USA003089
32.0/17.0

Space Flight Operations Contract

HAL/S COMPILER SYSTEM SPECIFICATION

PASS 32.0/BFS 17.0

November 2005

DRD - 1.4.3.8-a

Contract NAS9-20000

United Space Alliance

USA003089
32.0/17.0

HAL/S COMPILER SYSTEM SPECIFICATION

Approved by

Original Approval Obtained

Barbara Whitfield, Manager
HAL/S Compiler and Application Tools

Original Approval Obtained

Monica Leone, Director
Application Tools Build and Data Reconfiguration

DRD - 1.4.3.8-a

Contract NAS9-20000

Revision Log

The HAL/S-FC Compiler System Specification has been revised and issued on the following dates[1]:

Issue	Revision	Date	Change Authority	Sections Changed
29.0/14.0		03/10/1999	CR12935A	3.1.1.3 -p. 3-5, 3-6 App. B -p. B-5, B-8
			CR12940	2.6.1 -p. 2-6 2.6.2 -p. 2-6, p.2-7, 2.6.6 -p. 2-9 3.3 -p. 3-60, 3-61
			CR13043	5.2.3 -p. 5-2 5.2.7 -p. 5-11 5.3.3 -p. 5-168, 5-175 5.3.4 -p. 5-240
			DR109091	2.6.2 -p. 2-7
			DR109089	3.1.1.3 -p. 3-5
			Cleanup	-p. 2-4 -p. 3-4 - pp. 5-9, 5-10, 5-12, 5-34, 5-35, 5-37, 5-42 thru 5-45, 5-47, 5-50, 5-55, 5-61, 5-63, 5-64, 5-67, 5-68, 5-70, 5-72, 5-74, 5-75, 5-80, 5-82, 5-85, 5-93, 5-94, 5-96, 5-99, 5-101, 5-103, 5-106, 5-111, 5-113, 5-114, 5-116, 5-118 thru 5-125, 5-128 thru 5-131, 5-134 thru 5-136, 5-139 thru 5-143, 5-145, 5-147 thru 5-149, 5-151 thru 5-163, 5-166, 5-167, 5-169 thru 5-188, 5-190, 5-192, 5-194 thru 5-221, 5-226, 5-227, 5-230, 5-234 thru 5-236, 5-240, 5-242, 5-246, 5-247, 5-250, 5-257, 5-264, 5-267, 5-271, 5-272, 5-274, 5-278, 5-285, 5-287, 5-289, 5-293, 5-295, 5-296, 5-299, 5-300, 5-307, 5-315, 5-317, 5-319, 5-322, 5-323, 5-325 thru 5-342 -p. 6-4 -p. 7-5 -p. B-5

1. A table containing the Revision History of this document prior to the USA contract can be found in Appendix B.

30.0/15.0		06/12/00	CR12214	4.2 -p. 4-2 App. A -p. A-3
			CR12620	App. B -p. B-4
			CR13211	App. A -p. A-3
			CR13217	1.3 -p. 1-2 3.1.15.3 -p. 3-60, 3-61 7.5 -p. 7-6 App. B -pp. B-5, B-8
			CR13222	5.2.4.3 -pp. 5-3 thru 5-8 5.3.1 -pp. 5-39 5.3.2 -pp. 5-59, 5-79, 5-82, 5-83, 5-116, 5-118 5.3.3 -pp. 5-133 thru 135, 5-141 thru 5-144, 5-146, 5-149, 5-161, 5-213 5.3.4 -pp. 5-241 5.3.6 -pp. 5-294 5.3.7 -pp. 5-330, 5-331
			DR111314	7.2.3 -p 7-2
			Cleanup	Preface
31.0/16.0		09/07/01	CR13454	2.6.3 -p 2-7 3.1.1.1 -p 3-2 3.1.1.3 -p 3-5 3.1.1.5 -p 3-9 3.1.10.4 -p 3-47 3.1.12.5 -p 3-56 3.1.14.1 -p 3-62 3.1.14.2 -p 3-63 3.1.15.2 -p 3-64 3.1.17 -p 3-70 6.3 -p 6-28 6.3 -p 6-34
			CR13462	sec 4.0 -p 4-1 to 4-10
			CR13372	3.1.1.2 -p 3-3, 3-4
			DR111359	3.1.15.4 -p 3-69
			DR111364	3.1.5.3 -p 3-31 3.1.7.4 -p 3-39
			Cleanup	Preface
32.0/17.0		11/2005	CR13538	3.1.1.3 -pp. 3-5 thru 3-7 3.1.14.1 -pp. 3-62, 3-63 3.1.14.2 -pp. 3-63, 3-64 3.1.15.2 -p 3-66 3.1.15.4 -p 3-70, 3-71

			CR13570	3.1.15.5 -pp. 3-71, 3-72 3.3.4 -p 3-75 3.3.6 -p 3-75 8.3.2 -p 8-3 8.4 -p 8-4
			CR13652	3.1.1.3 -pp. 3-6, 3-7 3.1.14.1 -p 3-62 3.1.14.2 -p. 3-64, 3-65 3.1.15.3 -p 3-69 3.1.17 -p 3-72 4.2.3.3 -p. 4-6 8.3.1 -p 8-3 8.4 -p 8-4
			CR13811	3.1.1.3 -p 3-5
			CR13833A	2.10 -p 2-11
			CR14216A	Preface 1.3 -p 1-2 4.2.1.2 -p 4-2 7.1.2 -p 7-2

List of Effective Pages

The current status of all pages in this document is as shown below:

Page No.	Change No.
All	32.0/17.0

Preface

The *HAL/S FC Compiler System Specification* was developed by Intermetrics, Inc. and is currently maintained by the HAL/S project of United Space Alliance. The manual identifies the informational interfaces with the HAL/S-FC compiler and between the compiler and the external environment.

Over the years, numerous changes have been made to the HAL/S-FC compiler design and this document has been modified to reflect these changes. However only a small number of these changes have been incorporated into the HAL/S-360 compiler and now this document is only an approximation of that compiler's design. Earlier versions of the predecessor of this document (IR-182) contain a more accurate representation of the design of the HAL/S-360 compiler's components. The earliest of these versions should be referenced when attempting to understand the design of the HAL/S-360 compiler, and this document should be referenced when attempting to understand the design of the HAL/S-FC compiler.

The primary responsibility is with USA, Department, 01635A7.

Questions concerning the technical content of this document should be directed to Danny Strauss (281-282-2647), MC USH-635L.

This page intentionally left blank.

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

This page intentionally left blank.

List of Tables

List of Tables

This page intentionally left blank.

List of Figures

This page intentionally left blank.

1.0 Introduction

1.1 Scope of Document

This document specifies the informational interfaces within the HAL/S-FC compiler, and between the compiler and the external environment. An overall description of the compiler, and the hardware and software compatibility requirements between compiler and environment are detailed in the *HAL/S-FC Compiler System Functional Specification*.[2] Familiarization with the Functional Specification is presumed throughout this document.

This Compiler System Specification is for the HAL/S-FC compiler and its associated run time facilities which implement the full HAL/S language.[3] The HAL/S-FC compiler is designed to operate "stand-alone" on any compatible IBM 360/370 computer and within the Software Development Facility (SDF) at NASA/JSC, Houston, Texas.

1.2 Outline of the Document

The HAL/S-FC compiler system consists of:

1. a seven phase language processor (compiler) which produces object modules compatible with AP-101 Space Shuttle Support Software and a set of simulation tables to aid in run time verification.
2. a comprehensive run-time library which provides an extensive set of mathematical, conversion, and language support routines.

The organization of this document is based upon the organization of the compiler system. Each part of the system is considered as a separate entity with its own specific function and interfaces to other parts. Hence, there are four sections which cover the parts of the system as follows:

Section 2 - describes Phase 1 and the syntax analysis phase of the compiler.
Section 3 - describes Phase 2 and the code generation phase and specifies in detail the code patterns for specific HAL/S constructs.
Section 4 - describes Phase 3 and the operation of the Simulation Data File generator.
Section 5 - describes the Runtime Library and the concepts used in the library and also gives specific information about each library routine including size, speed, and algorithm.

In addition to this part-by-part documentation, the compiler system, taken as a whole, exhibits properties and interfaces which are not specific to any one of the pieces. General information about such topics as the compiler's operating environment and user-written interfaces to emitted object code are contained in Section 6. Several Appendices are included which deal with tabular data used in the compiler system.

2. HAL/S-FC Compiler System Functional Specification, 24 July 1974, IR-59
3. HAL/S Language Specification, USA003088

1.3 Status of Document

This document, plus the *HAL/S-FC Compiler System Functional Specification for the AP-101* comprise the complete *HAL/S-FC Compiler System Specification*.

The HAL/S-FC compiler inherits some of its operational features from the HAL/S-360 compiler system for which a similar Specification exists. In addition, many features of the HAL/S-FC system are under control of Interface Control Documents which are subject to update. When appropriate within this document, references are made to these companion documents as sources of supplementary material and in some cases as primary sources of detailed information.

The following list of documents represents the set of additional documents which reflect design and control of the HAL/S-FC compiler system:

- HAL/S-FC Compiler System Functional Specification for the AP-101, IR–59.
- Interface Control Document: HAL/FCOS, (a.k.a. HAL/FCOS ICD), USA001460.
- Interface Control Document: HAL/S-FC/SDL, (a.k.a. HAL/SDL ICD), USA001556.
- HAL/S-360 Compiler System Specification, USA001528.
- HAL/S Language Specification, USA003088.
- SPF Optical Disk Flight Software Products Indexing Requirements Interface Control Document, JSC-26622 Section 2.1.9.

2.0 Phase 1 - Syntax Analysis

The Syntax Analysis Phase performs syntactic and semantic analysis of the user's HAL/S source programs. It performs all functions necessary to allow an independent Phase 2 program to generate code for the target computer. The basic design of the HAL/S system includes use of a single Phase 1 for a variety of target machine Phase 2s. Thus, the Phase 1 used by the HAL/S-FC compiler is the same one used in the HAL/S-360 compiler. In this section on Phase 1, data which is supplied in detail in the *HAL/S-360 Compiler System Specification* is not repeated. Instead, reference is made to the proper section of that document.

This section deals with the following Phase 1 functions:

- Primary Source Input
- Secondary Source Input
- ACCESS System Implementation
- Compiler Directives
- Template Checking and Generation
- Printed Data
- Symbol Table Creation
- Statement Table Generation
- Literal Table Generation
- HALMAT Creation
- The Optimizer

2.1 Primary Source Input

Phase 1 accepts primary source input in the form of fixed length logical records. This input must be defined by the SYSIN DD statement in the JCL invoking the compiler. The first byte of each record is used to define the type of the record as follows:

M - main line

E - exponent line

S - subscript line

D - compiler directive

C - comment

(Other letters can be used if modified via the "CARDTYPE" compiler option to a legal type.)

For stand-alone operation the source records are 80 bytes in length and may contain data in columns 2-80. Optionally, the user may designate, via the "SRN" compiler option, that the source scanning is to stop at position 72 and also that positions 73-78 are to be printed on the listing as "Statement Reference Numbers".

When operating in the SDL environment, indicated by use of the "SDL" compiler option, the source records must still be all the same length but that length may be from 80 to

132 characters. When in the SDL mode, the compiler accepts source data from record positions 2 through 72. In addition, when the records are of sufficient length, the following fields are recognized and all except the change authorization field are printed on the primary source listing:

- Record Sequence Number - positions 73 through 78;
- Record Revision Indicator - positions 79 and 80;
- Change Authorization Field - positions 81-88
- Portions of records beyond position 88 are ignored.

The compiler's primary input may optionally be in a compressed source format as defined in the *HAL/SDL ICD*. No special notification of use of compressed source is needed. Phase 1 determines the type of input by examining the first record. Catenated datasets defined as primary compiler input must all be either in compressed or noncompressed format for one invocation of the compiler.

2.2 Secondary Source Input - The Include System

The user may direct the compiler to an alternate input source by use of an INCLUDE compiler directive in the primary input. The exact form of the INCLUDE directive may be found in Section 5.2 of the HAL/S-FC User's Manual.

The INCLUDE directive defines a member name in a partitioned dataset. Phase 1 uses a FIND macro to locate the member on the INCLUDE DD card. If the FIND is unsuccessful, an identical FIND is issued for the OUTPUT6 DD card. A member, when located, is read to its end by the compiler. The records are processed identically to primary (SYSIN) input with the exception that further INCLUDE directives within INCLUDE'd source are not allowed. The same source margins are applied to the INCLUDE'd source as are applied to the primary input. In addition, the compiler prints a line in the primary source listing indicating the catenation sequence number of the DD card on which the member was found and the RVL field from the PDS directory entry for the member. The RVL field is the first 2 bytes of user data after any TTRNs.

The individual members which are INCLUDE'd may be in either compressed or uncompressed format, independent of whether the primary input was compressed. The form of each INCLUDE'd member is determined by the compiler from the first record read.

Partitioned datasets may be catenated together in the JCL to form the INCLUDE DD sequence, but such datasets must have identical DCB attributes.

2.3 Access Rights Implementation

The HAL/S language allows managerial restrictions to be placed upon the usage of user-defined variables and external routines. The existence of such a restriction is indicated by the use of the ACCESS attribute as described in the *HAL/S Language Specification*. The manner in which the restrictions are enforced in the HAL/S-FC compiler system is described below.

Any variable in a COMPOOL template or any external routine to which the ACCESS attribute has been applied is considered to be restricted for the compilation unit which is being compiled. The restriction is slightly different for variables than for blocks:

a. Variables with the ACCESS attribute may not have their values changed.
b. Block names may not be used at all.

These restrictions may be selectively overridden for individual variable and block names. The selection of which ACCESS controlled names are to be made available to the unit being compiled is performed by processing an external dataset. The external dataset is known as the Program Access File (PAF). The PAF must have partitioned organization and is specified by the following JCL:

```
//HAL.ACCESS DD DSN=<PAF name>, <other parameters>
```

where the <PAF name> is the dataset name of the PAF without any member specification.

Each member of the PAF contains the information about ACCESS controlled names which are to be made available to one unit of compilation. The member name is defined by a Program Identification Name (PIN). The PIN is specified to the HAL/S-FC compiler by using the PROGRAM compiler directive in the primary input stream:

```
col    1
       D    PROGRAM ID = <id>
```

The <id> field of the directive is a 1 to 8 character identifying name which is used to select the member of the PAF to be processed for the current compilation's ACCESS information. The appearance of the PROGRAM directive in the compiler's input stream causes immediate processing of the PAF member specified.

The format of an individual PAF member is described below.

a. Column 1 of each record is ignored except when column 1 contains the character "C", in which case the entire record is ignored.
b. The portion of each record which is processed is the same portion which is processed in the primary compiler input (SYSIN).
c. COMPOOL elements which are to be made available to the compilation are specified as:

   <COMPOOL-name>($ALL)

   <COMPOOL-name>(<var-name>, <var-name>,…<var-name> or

   The first format specifies access to individual variables within the named COMPOOL. The second format specifies access to all variables within the named COMPOOL.

d. Access to external block names is specified as:

   $BLOCK(<ext-name>, <ext-name>,…<ext-name>)

e. Blanks are allowed anywhere in the record except that names may not be broken by a blank.

f. Either of the constructions (c) or (d), above, may span more than one record.

g. The name of the particular COMPOOL in the form (c) above may appear more than once; i.e. the variables in a particular COMPOOL do not have to be specified at one time. Similarly, the form $BLOCK may appear more than once.

Some validity checking is performed by the compiler while processing the PAF member. Warnings are issued for the following conditions:

1. A syntax error on a PAF record- the bad record is printed.
2. Names mentioned in the PAF are not defined.
3. Elements of $BLOCK in the PAF are not defined.
4. Requests for names which are not ACCESS protected.
5. Variables found, but not within the COMPOOL specified.
6. Names used in the context of a COMPOOL-name which are not COMPOOLs.

If, at the time the PROGRAM directive is encountered, there have been no ACCESS-controlled variables declared, the PAF is not opened. If a user does not require access to any, the PROGRAM directive and associated PAF members may be omitted.

2.4 Compiler Directive Parsing

When an input record is found which contains a "D" in column one, Phase 1 scans the remainder of the card for a valid compiler directive. A list of legal compiler directives and their function is listed in Section 5.2 of the HAL/S-FC User's Manual.

Directive processing is done independently of HAL/S source language parsing, i.e. words used on Directive cards are not necessary HAL/S language keywords. Similarly, HAL/S language keywords are not recognized as such on Directive cards.

2.5 Template Checking and Generation

Phase 1 assumes the task of source template verification and generation. Every compilation unit in the HAL/S-FC system has a source template. When the block header for a unit of compiler is encountered, Phase 1 begins to construct the source template for that unit as follows.

The member name for the template being created is determined. This is done by taking the "characteristic name" for the unit and preceding it by the characters '@@'. The characteristic name for any unit is created by taking the block name, removing any underscore characters, and then padding or truncating the result to 6 characters. An attempt is made to locate a member of this name on either the INCLUDE or OUTPUT6 DD cards. If such a member is found, the contents of the member are compared with an internal, temporary template created as the compilation proceeds. If the existing template and the internal one agree, a template update is not required, and the existing template remains intact. If the templates do not agree, the internal template is written to the OUTPUT6 DD card and STOW'ed with the current member name. If the initial search for an existing template fails, the generated template is automatically written and STOW'ed on the OUTPUT6 DD card. The PDS directory entry for a template member is created with two bytes of user data. The two bytes are initialized to X'F0F0'.

Phase 1 also sets appropriate bits in a field which is passed back to the caller of the compiler as the high order byte of register 15. The definitions of these bit settings is defined in the *HAL/SDL ICD*.

Generation of the internal template is performed during syntax analysis on a token by token basis. As statements are encountered which are required in the template, the tokens from the statements are added to an internal buffer. When a new token will no longer fit in the buffer, the buffer is written and cleared for continuation. Thus, the templates take the form of strings of HAL/S tokens separated by one block. The template statements are continued from one line to the next without regard for statement boundaries, thus producing the template in the most compact form possible.

For the comparison of existing templates with new, generated templates, the generated records are compared character for character with the existing records. Any mismatch is considered to indicate a change in the template.

Templates are never generated using the compressed source format mentioned in Section 2.1. The generated templates conform to the source margins in effect for the compilation (e.g. for an SDL mode compilation, templates are created with source in position 2 through 72 of the records. When template records are written to the OUTPUT6 DD card, the records are padded with blanks or truncated as necessary to conform to the LRECL specification for that DD card.

When a template has been found to have changed, the compiler updates a "Version number" associated with the template. For an existing template, the version number is found on a VERSION compiler directive card at the end of the existing template member. If a new template is needed, the version number is incremented by one and placed on a new VERSION directive card at the end of the generated template. The version number is limited to the range 1 to 255. Upon reaching 255, the next incrementation causes the number to begin again at 1. When no existing template can be located, the version is set to 1.

When templates produced by the compiler are referenced in subsequent compilations by use of an INCLUDE for the template, the version numbers from the referenced templates are emitted into the produced object code on special SYM records which indicate the versions of all external references. In addition, the emitted object code for any compilation unit contains a SYM record indicating the version number of the template created for that compilation unit. This information permits the checking, if desired, of proper integration of separately compiled units by providing information necessary for cross-checking of inter-module references.

2.6 Listing Generation

2.6.1 Options

All Type 1 and Type 2 options listed in the HAL/S-FC User’s manual except debug options and HAL/S-360 unique options are printed in alphabetical order. For Type 1 options, just the option is printed if the option is on. If the option is off, the option is prepended with a “NO”.

2.6.2 Primary Formatted Listing

The central printed output of the compiler is the primary source listing. This listing is designed to document the actions taken by the compiler during its generation of an executable form of the user's source program in an indented, annotated format. Additional information, such as block summaries and symbol table listings, are also part of the primary source listing.

The formatting of the primary source listing leads to the documentation of the users program in two ways: 1) variable annotation, and 2) logical indenting.

1. Variable annotation - Each user-defined data symbol, when printed on the primary source listing, receives "marks" appropriate to the type and organization of the symbol. This annotation is that which is defined by the *HAL/S Language Specification*.
2. Logical indenting - Each statement printed on the primary source listing is formatted and indented to show internal statement structure, and to show the statements' hierarchical and nesting relationships to other statements in the compilation. The indention increment is 2 spaces.

When operating in the SDL, additional information is provided on the primary source listing. The Record Sequence Number and Record Revision Indicator fields (see Section 2.1) are printed on the primary source listing next to the statements to which they apply. The revision level is printed to the right of the statement immediately following the vertical bar. Another vertical bar separates the revision level from the current scope. Additional details of the specific operations performed during SDL operation may be found in the *HAL/SDL ICD*.

All lines are single-spaced except for the following: there is a blank line before a group of one or more E-lines, C-lines or D-lines and after a group of one or more S-lines.

For D INCLUDES, the first statement number associated with the include is printed.

If there is an IF-THEN or IF-THEN-ELSE statement followed by a simple DO, the DO appears on the same line as the IF-THEN or ELSE except when the combination of the statements is too long for a single line. The combined IF-THEN and DO statements (including the semicolon of the DO) will be broken into multiple lines following regular compiler rules. The statement number for the IF-THEN will be printed as the statement number for each line. If the THEN and the DO or the ELSE and the DO are separated by a C-line or a D-line, the DO will be placed on its own line with its own statement number.

Normally, the current scope is printed to the right of each line in the compilation listing. The value in the scope field will be truncated if it exceeds the maximum line length. The following list indicates instances where the current scope will be replaced by a different value:

a. The scope field for END statements contains the statement number of the corresponding DO statement.
b. The scope field for the first statement line (that is not a label) of a case in a DO CASE group contains the case number.

c. The scope field for an IF-THEN followed by a simple DO is replaced by “DO=ST#”, where ST# is the statement number of the DO. Usually, the scope is replaced with the statement number of the DO for each line of a multi-line statement. However, because of certain compiler limitations or other uses of the scope field, the “DO=ST#” may not appear on all of the lines. Following are the known cases:

1. If the length of the statement exceeds a certain compiler-limited size, the statement number of the DO will not be printed for the first line(s) of the statement.
2. If a C-line or D-line is placed inside the IF-THEN statement, the “DO=ST#” will only be placed in the current scope for the lines following the last C-line or D-line.
3. If the multi-line IF-THEN-DO is the first statement of a case in a DO CASE group, the scope field of the first line will contain the case number.

Only one of a-c from above will be placed in the scope field for a given line, with the order of precedence as listed above.

Depending on the contents of the macro, the formatting of statements containing replace macros may vary from the requirements listed above.

2.6.3 Error Messages

When compilation errors are detected by Phase 1, an error message is printed in the primary listing at the point of detection. All error messages have an identifying code associating with them.

The code is assigned to messages according to a general system which groups errors according to a class and a subclass. Multiple errors within a class/subclass combination are assigned unique numbers within the group. Thus, every possible error in the HAL/S-FC compiler system has a unique identifying code.

The text of all error messages is maintained on a direct access dataset. The compiler retrieves error message text as needed from this dataset. During compilation, the ERROR DD card defines the error message dataset. This file has partitioned organization and contains one member for each error message. The member names are identical to the identifying code assigned to the errors.

The record format of the error library is FB and the logical record length is 80 bytes. The first record of each member defines the severity of that error. The severity is a single EBCDIC number in position one of the first record. The severities and their effects are:

Severity 0 messages will be warning messages (Severity 1) that have been downgraded. Processing will continue, and object code will be generated.

Severity 1 messages will be minor errors in which compilation will be allowed to continue. Since these errors could produce bad object code, compilation will abort and no object code will be generated.

Severity 2 messages will be major errors. These errors usually involve unimplemented features. Compilation will abort as results will be unpredictable. No object code will be generated.

Severity 3 messages will be severe errors that require user action. Compilation will abort immediately and no object code will be generated.

Severity 4 messages will be internal compiler errors. Compilation will abort immediately and no object code will be generated. Compiler support personnel should be notified, and a compiler DR usually results.

Within the text of an error message, locations into which specific descriptive information may be placed are denoted by the appearance of two question marks (??). For errors which have this feature, the compiler supplies additional description text (such as the name of an identifier) to make the printed error message as specific and informative as possible.

2.6.4 Block Summaries

The HAL/S-FC compiler provides additional information on the primary listing at the close of HAL/S code blocks. The blocks for which summaries are given are PROGRAM, TASK, FUNCTION, and UPDATE.

Information contained in block summaries consists of lists of labels or variable names used in various contexts within the block. The title "BLOCK SUMMARY" begins the list. For all potentially summarized contexts within the block, a descriptive heading is printed followed by the list of names involved. A "*" next to any name in the block summary indicates that the name appears in a context which changes its value. The headings are listed below.

PROGRAMS AND TASKS SCHEDULED
PROGRAMS AND TASKS TERMINATED
PROGRAMS AND TASKS CANCELED
EVENTS SIGNALED, SET, OR RESET
EVENT VARIABLES USED
PROGRAM OR TASK EVENTS USED
PRIORITIES UPDATED
EXTERNAL PROCEDURES CALLED
EXTERNAL FUNCTIONS INVOKED
OUTER PROCEDURES CALLED
OUTER FUNCTIONS INVOKED
ERRORS SENT
COMPOOL VARIABLES USED
COMPOOL STRUCTURE TEMPLATES USED
COMPOOL REPLACE DEFINITIONS USED
OUTER VARIABLES USED
OUTER REPLACE DEFINITIONS USED
OUTER STRUCTURE TEMPLATES USED

2.6.5 Compilation Layout Summary

Immediately preceding the Symbol Table printout at the CLOSE of the HAL/S program, there is a compilation layout map, indicating the way in which PROGRAMS, TASKS, PROCEDURES, FUNCTIONS, and UPDATE blocks were defined. The indent level in this printout indicates the nesting level definition of the block shown. This serves to give a quick overview of the compilation structure.

2.6.6 Symbol & Cross Reference Table Listing

The symbol and cross reference table printed at the end of a HAL/S compilation listing provides a detailed accounting of all programmer-defined symbols. The table listing is organized into two parts: a structure template listing and an alphabetized total listing. These parts are labeled appropriately and are separated by a page break.

Any structure templates defined in the compilation appear first in the symbol and cross reference table. The template names appear in alphabetical order. All structures declared using each template are listed alphabetically after “USED BY” under the template in the attributes and cross reference area. The body of each template (i.e. the levels defined under the template name) is also listed under the template name in the order of definition. This ordering provides a quick reference to the organization of the structure template.

Following any listing of the templates, an alphabetized listing of all programmer-defined symbols is printed. Symbols previously listed as element of a structure template are included in this list. However, the list is completely alphabetized and template organization is not shown. When a particular symbol is independently defined in more than one name scope, the symbol is multiply listed in order of definition.

2.6.7 Built-in Function Cross Reference

Phase 1 also produces a listing of any HAL/S built-in functions used in a compilation. The printout shows the statement numbers at which the references to the built-in functions occurred.

2.6.8 Replace Macro Text

If any HAL/S REPLACE statements were used in the compilation, the text of the macro is printed in the symbol table listing in the attributes and cross reference area.

2.6.9 Unformatted Source Listing

Under control of the "LISTING2" compiler option, Phase 1 will optionally produce, on the file defined by the LISTING2 DD card, a listing of the input (both SYSIN and INCLUDE) source records as read by the compiler. No special annotation, formatting, or indenting is performed. In the case of input in the SDL compressed format, the LISTING2 option produces the records in their uncompressed format.

2.7 Symbol Table Generation

Phase 1 is responsible for initial creation of the compiler's internal symbol table. The symbol table consists of a group of arrays which describe all of the properties of declared variables and labels. The capacity of the symbol table is under user control by means of the SYTSIZE compiler option. This table, as created by Phase 1, is located in an area common to all compiler phases. Thus, Phase 2 inherits the initialized table from Phase 1.

Design of the HAL/S-FC compiler includes, as a basic concept, the use of a Phase 1 and Phase 1/Phase 2 interface identical to that of the HAL/S-360 compiler. Thus, the description of the internal symbol table to be found in the *HAL/S-360 Compiler System Specification*, Appendix B.2 is sufficient to define the HAL/S-FC table.

2.8 Statement Table Generation

The statement table passes information about executable statements from Phase 1 of the compiler to Phase 3. This information allows Phase 3 to include statement type and target variable information in the Simulation Data Files.

Due to the use of a common Phase 1 in the HAL/S-360 and HAL/S-FC compiler systems, the Statement Table description in the *HAL/S-360 Compiler System Specification* document is sufficient to describe the HAL/S-FC table (See Appendix B.3 of that document).

The basic table description includes reference to an "extension" field in which statement memory addresses and/or SRN data is stored. Use of this field is activated by use of certain compiler options:

SRN data is included in the Statement Table if either of the SRN or SDL compiler options are used.

Beginning and ending addresses for individual HAL/S statements are included in the Statement Table when the ADDRS compiler option is used.

The Statement Table is produced on the file specified by the FILE6 DD card. No Statement Table data is communicated via in-memory tables.

2.9 Literal Table Creation

The format of the HAL/S-FC literal table is identical to that used by the HAL/S-360 compiler as described in Appendix B.1 of the *HAL/S-360 Compiler System Specification*.

The size of the area in which character literal data is stored is under user control via the LITSTRINGS compiler option. This character literal area is communicated to subsequent phases of the compiler through common memory locations.

The portion of the literal table which contains arithmetic literal, bit literal, and pointers to character literal is passed to later phases via the dataset defined by the FILE2 DD card.

2.10 HALMAT Creation

HALMAT is the intermediate code medium by which the structure of the compiled HAL/S-FC is passed to Phase 2 for code generation. The HAL/S-FC compiler uses a similar Phase 1 as the HAL/S-360 compiler. A description of HALMAT as used by the HAL/S-360 compiler can be found in Appendix A of the *HAL/S-360 Compiler System Specification* and a description of the HAL/S-FC HALMAT can be found in Appendix A of the *HAL/S-FC Compiler System Program Description Document*.

HALMAT is passed to Phase 2 through use of auxiliary storage as defined by the FILE1 DD card.

2.11 The Optimizer

The HALMAT produced by Phase 1 is a direct representation of the HAL/S program being compiled. A separate phase of the compiler exists between Phases 1 and 2 which examines and manipulates the HALMAT in order to produce an optimized HALMAT representation. This phase, known as Phase 1.5, is conceptually a part of Phase 1. Its operation is transparent to the user as it produces no standard printouts.

The Optimizer performs the following functions:

- Common subexpression elimination, including subscript computations
- Additional literal folding
- Pulling loop invariant subexpressions out of loops
- Replacement of unneeded divisions by multiplications
- Suppression of unnecessary matrix transpose operations
- Indicate linear operations on Vectors and Matrices to allow for in-line code.
- Indication of procedures which cannot be leaf procedures (as an aid to Phase 2)
- Indicate the next use of variables and subexpressions, to assist register allocation in the code generator

These operations are carried out by modifying the HALMAT, literal table, and symbol table.

While the Optimizer is a separate phase, it is conceptually a part of Phase 1 and is described in the *HAL/S-360 Compiler System Specification*.

This page intentionally left blank.

3.0 PHASE 2 - Code Generation

The code generation phase of the HAL/S-FC compiler has the primary function of producing machine language instructions for the AP-101. Phase 2 also performs other tasks which are also the subject of this chapter.

This section deals with the following Phase 2 functions:

- Code Generation
- Naming Conventions
- Printed Data
- Symbol Table Augmentation
- Statement Table Augmentation

3.1 Code Generation

3.1.1 Bases and Conventions

Phase 2 produces AP-101 machine language instructions which perform the operations indicated by each line of HALMAT received from the syntax and semantic analysis phase. This section describes in detail the ground rules which the code generation phase follows in producing object code. The following terms will be used throughout the ensuing text:

R	-	A general accumulator (integer or scalar);
X	-	An indexing register (for subscripting);
B	-	A base register containing a base address used to compute the effective address of a variable, constant, temporary, or program label.
OFFSET	-	The constant term which, when subtracted from the actual data address of a variable, yields the address of the 0'th item of the aggregate data collection (note that all HAL subscripts start counting from 1). This is 0 when the variable is a single item.
VAR	-	The address of a declared non-parameter HAL/S variable. For addressing purposes, it is actually the base address of the actual data minus the OFFSET. Single valued integer, scalar, or bit input parameters also will use this form.
PAR	-	The address of a formal parameter passed "by reference". This includes any assigned parameters, plus any input parameters which are not simple integer or scalar variables. Note that PAR actually contains an address.
DELTA	-	The constant indexing term in a subscript calculation. This term may also reflect the displacement of a structure terminal within a structure template.
OP	-	Any AP-101 machine instruction.

Note - When VAR or PAR appears in machine instruction constructions, it represents the displacement difference between the data address and the base address contained in the base register B.

3.1.1.1 Register Usage

The following register assignments are used by the code generator:

F0-F5	Used for floating point accumulators and parameters.
F6-F7	Used for floating point accumulators only.
R0	Stack register. This register points to the caller's register save area in the run time stack. In addition, all formal parameters, temporaries, and AUTOMATIC variables in REENTRANT procedures are based on this register.
R1	Global data addressing register. This register is used to address all of the declared variables and literals within a compilation unit.
R2	Work addressing register. This register is used to pass address parameters, dereference NAME variables, and set up any other dynamic addressing.
R3	Local addressing register. This register is used in SRS instructions only to address a certain subset of the local data in a block. When the DATA_REMOTE directive is in effect (see Section 4.0), register 2 can only be loaded with non-local data addresses (COMPOOL, etc.) and register 3 can only be loaded with local data addresses.
R4	Linkage register. This register records the return address for all subroutine linkages. It may also be used for an integer accumulator.
R5- R7	Used for integer accumulators, index registers, and parameter passage where applicable.

3.1.1.2 Storage Allocation

The HAL/S-FC compiler arranges data in memory such that the least number of base registers need be dedicated in addressing.

Data is grouped into three major categories: single value (constant offset=0), aggregate (character, vector-matrix, structure without copies), and array (including structure with copies). Within each group, data is ordered such that data requiring the same boundary alignment is adjacent, minimizing the storage lost due to hardware alignment requirements. Within the array group, ordering is further carried on such that multidimensional arrays (with larger offsets) come after single dimensional arrays. These above orderings are carried on independently for: 1) program data, and 2) each COMPOOL block contained in the compilation unit. Note that program data includes all variables within the compilation unit including those defined in procedures, functions, or any other block.

Structure templates, unless declared as RIGID, are internally ordered such that the minimum boundry alignment within any node level is required. Template matching requirements guarantee that templates exhibiting identical properties will be identically reordered.

After all groupings are complete, storage assignments are made, with the required base-displacement combinations being generated to properly access the data. The storage addresses assigned refer to the actual data beginning, but for arrayed data types, the base-displacement address includes a negative offset value (COMPOOL variables that are not referenced do not have base-displacement addressing generated). This negative offset value is commonly referred to as an imaginary *0th element.*

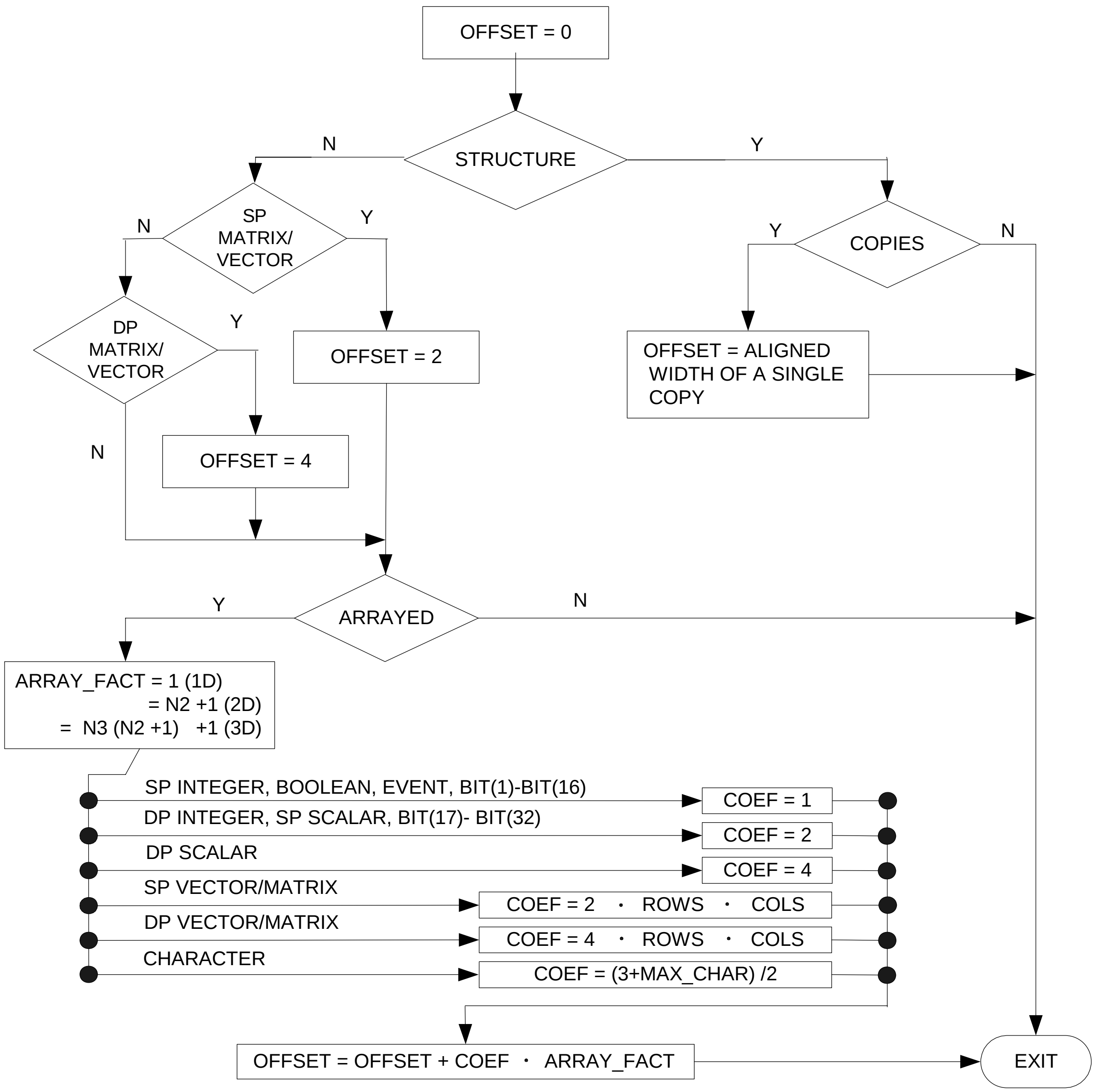


Figure 3-1 Algorithm for Calculating the 0^{th} Element Offset

Example:

```
   DECLARE A SCALAR,
           B INTEGER,
           C CHARACTER(7),
           D ARRAY(5) DOUBLE;
   DECLARE E ARRAY(5),
           F ARRAY(3,3) VECTOR,
           G MATRIX;
   DECLARE H DOUBLE,
           I ARRAY(5,5) INTEGER;
```

Alignment	Name	Location	Base	Displacement	(In Decimal) 0^{th} element
Halfword	B	00002	1	0002	0
Fullword	A	00004	1	0004	0
Doubleword	H	00008	1	0008	0
Halfword	C	0000C	1	000C	0
Halfword	I	00011	1	000B	-6
Fullword	E	0002A	1	0028	-2
Doubleword	D	00034	1	0030	-4
Fullword	G	00048	1	0046	-2
Fullword	F	0005A	1	0040	-26

Note that all formal parameters and all AUTOMATIC variables in a RENTRANT PROCEDURE or FUNCTION are based off the stack register (0).

3.1.1.3 Addressing Concepts

This section describes the general addressing rules for data. To the extent possible, data can be directly addressed via some combination of base register and bit displacement (eleven bits for indexed addressing). This is not possible whenever the data item is a formal parameter other than a simple integer or scalar, or any formal parameter scoped in from an outer to an inner procedure. The skeletal forms given in Section 3.2 assume the most commonly used addressing forms. The rules described here should be superimposed upon these skeletal forms to interpret all possible combinations of operations between operands.

Simple Addressing Forms

Simple Variable

```
OP   R,VAR(B)
```

Simple Aggregate Component (array or vector-matrix)

```
OP   R,VAR+DELTA(X,B)
```

Simple Integer-Scalar formal parameter

```
OP   R,VAR(0)
```

Simple Aggregate formal parameter

```
L     B,PAR(0)
OP    R,DELTA(X,B)
```

NAME Variable in dereference context

```
LH    B,VAR(B)
OP    R,DELTA(X,B)
```

NAME Variable in dereference context (ASSIGN formal parameter)

```
L     B,VAR(B)
LH    B,0(B)
OP    R,DELTA(X,B)
```

When the DATA_REMOTE directive is in effect (see Section 4.0), if register 1 or 3 is loaded with a new base address and used in operation OP as base B, then it will be restored immediately after OP to its original local data pointer value with:

```
LH    B,stack_location(0)
```

However, if the next instruction is a conditional branch, then it will be restored with:

```
LH    B,stack_location(0)
SLL   B,16
```

REMOTE Variable

```
      OP@#  R,ZCON(X,1)
ZCON DC    Z(0,VAR,0)
```

NAME REMOTE variable in dereference context is basically the same as a REMOTE variable, except the NAME variable is used in place of the ZCON

```
OP@#  R,VAR(X,1)
```

NAME REMOTE variable in dereference context that lives REMOTE

```
L@#   R,ZCON(X,1)
ST    R,stack_location(0)
OP@#  R,stack_location(X,0)
```

REMOTE formal pass-by-reference (address) parameter

```
OP@#  R,stack_location(X,0)
```

Address Passage Addressing Forms

For parameter passing to procedures, functions, and library routines, it is often necessary to pass address pointers instead of data. The following sequences could be used anywhere the instruction LA appears in the generated code sequence.

Unsubscripted variable:

```
LA    R,VAR(B)
```

Subscripted variable:

```
SLL   X,<index alignment>
LA    R,VAR(X,B)
```

Unsubscripted REMOTE variable:

```
L    R,ZCON(1)
```
[4]

Subscripted REMOTE variable:

```
SLL  X,<index alignment> or SLL R,<index alignment>
L    R,ZCON(1)
AR   R,X
```
[4]

Non-aggregate variable to REMOTE library or to REMOTE parameter in HAL/S procedure or function:

```
LA   R,VAR(B)
OHI  R,x'8000' (PASS only)
IAL  R,x'0800'
```

Subscripted variable to REMOTE library or to REMOTE parameter in HAL/S procedure or function:

```
SLL  X,<index alignment>
LA   R,VAR(X,B)
OHI  R,x'8000' (PASS only)
IAL  R,x'0000'
```

Unsubscripted aggregate variable to REMOTE library or to REMOTE parameter in HAL/S procedure or function:

```
LA   R,VAR(B)
OHI  R,x'8000' (PASS only)
IAL  R,x'0000'
```

Non-aggregate variable to REMOTE library or to REMOTE parameter in HAL/S procedure or function through a NAME dereference:

```
LH   R,Name_Var
IAL  R,x'0800'
SRA  R,1
SRR  R,31
OHI  R,x'8000' (PASS only)
```

Subscripted variables to REMOTE library or to REMOTE parameter in HAL/S procedure or function through a NAME dereference:

```
LH   B,Name_Var
LA   R,<INDEX>(B)
SRA  R,1
SRR  R,31
OHI  R,x'8000'(PASS only)
```

Unsubscripted aggregate variable to REMOTE library or to REMOTE parameter in HAL/S procedure or function through a NAME dereference:

```
LH   R,Name_Var
SRA  R,1
SRR  R,31
OHI  R,x'8000' (PASS only)
```

4. ZCON DC Z(0,VAR,0).

Subscripted variable to REMOTE library.

(stack variable only)

```
SLL   X,<index alignment>
LA    R,VAR(X,B)
IAL   R,x'0400'
```

Note that the compiler emits an RLD card that informs the linkage editor to insert the proper CSECT value into the last four bits inserted by the IAL instruction for non-NAME non-stack variables, to conform to the ZCON format.

For stack variable (B=0), this cannot be done because the stack CSECT is undefined at compile time. The '0400' sets the ZCON's C bit which will allow correct address expansion for either CSECT 0 or 1, which is where the stack is located.

Indexing:

The computation for all indexing is done as follows. All constant index terms are factored out from the variable terms. The variable terms are computed according to the natural sequence of unwinding aggregate data. The constant terms are similarly computed to form a DELTA. The variable subscript in register X is shifted according to the halfword width of the data being indexed, except for those instructions which perform automatic index alignment. The DELTA is similarly shifted at compile time. If $0 \leq DELTA < 2048$, it is used in the variable displacement. Otherwise, it is added to X if X is non-zero, or loaded into a newly created X if X is zero (i.e. the subscript contains no variable terms).

3.1.1.4 Condition Codes

The following table lists the allowable relational operations and the resultant condition code - referred to as COND throughout the remainder of this section. Note that the AP-101 conditional branch instructions branch on the "not true" condition.

<OP>	COND
=	3
¬=	4
<	5
>	6
¬< or >=	2
¬> or <=	1

3.1.1.5 ZCONs and the Calling Mechanisms

Throughout the descriptions of generated code of Section 3.1, branches to other CSECTs (comsub or library) are generally indicated as:

ACALL <routine name>

The actual implementation of this linkage is to go not directly to the named routine, but instead to branch indirectly through a long address constant (ZCON) located in sector 0 of the machine.

When the target of the branch is a compiler-generated CSECT (a COMSUB), the ZCON referenced will be one created during compilation of the COMSUB. The long indirect address will be in a CSECT named #Znnnnnn (see Section 3.2) which will in turn refer to the real code CSECT.

When the target of the branch is a library routine, the ZCON referenced will be one provided with the library. Its name will be #Qnnnnnn and it will in turn refer to the proper library code CSECT. Certain library routines, for reasons of execution speed, are referenced directly by compiler-emitted code without going through a ZCON. These routines are designated in the BANK0 column of the library documentation. This direct addressing requires that these routines reside either in sector zero or in the same sector as the compiler code which references them.

The use of ACALL in the descriptions implies an external call. In actuality, the instruction generated may be either:

```
    SCAL    0,<routine name>
or
    BAL     4,<routine name>
```

depending on whether the library routine has been designated as PROCEDURE or INTRINSIC type.

Some of the parameter setups show the use of P1, P2, and P3 for parameter registers. The following table shows the actual register values for P1, P2, and P3 depending upon the nature of the library routine (see library documentation for specific details).

	P1	P2	P3
Intrinsics	1	2	3
Procedure-			
P1 used	2	4	7
P1 not used	X	2	4

When the DATA REMOTE directive is in effect (see Section 4.0), the ACALL will be preceded by the instruction LDM $ZDSECLR to clear the DSE registers, and will be followed by the instruction LDM $ZDSESET to set the DSE registers upon return.

3.1.1.6 The Runtime Stack

The HAL/S-FC compiler system employs a runtime stack mechanism as an integral part of its operation. The stack mechanism is used to provide subroutine linkage areas, temporary work areas, error environments, and to provide reentrancy of code blocks when needed. The actual memory used as a stack space for a given HAL/S process is provided by the flight computer operating system (FCOS). The determination of the size required for a particular stack is made by the flight computer support software linkage editor. The linkage editor determines stack size (and upon special request will create a stack CSECT) from information provided on SYM cards in the modules being link edited. The HAL/S-FC compiler emits the SYM cards as part of its object modules. The runtime library uses a system of macros to generate the properly named DSECTs and SYM entries for stack size computation.

The details of formats and requirements relating to stack generation can be found in the *HAL/SDL ICD*. That document also contains the detailed description of the "stack frame", that portion of a total stack which is used by an individual subroutine when that subroutine has been invoked. The description of the basic stack frame is reproduced here for reference.

The active stack frame is pointed to by the pointer in register R0. The back link to the previous stack frame is established when a new level is entered. A pointer, NEW R3, is established for any block with a local data area. If a local data area is not present, e.g. in the case of a HAL/S-FC library routine, NEW R3 is set to zero. See Section 3.1.1.7 for a definition of the local data area.

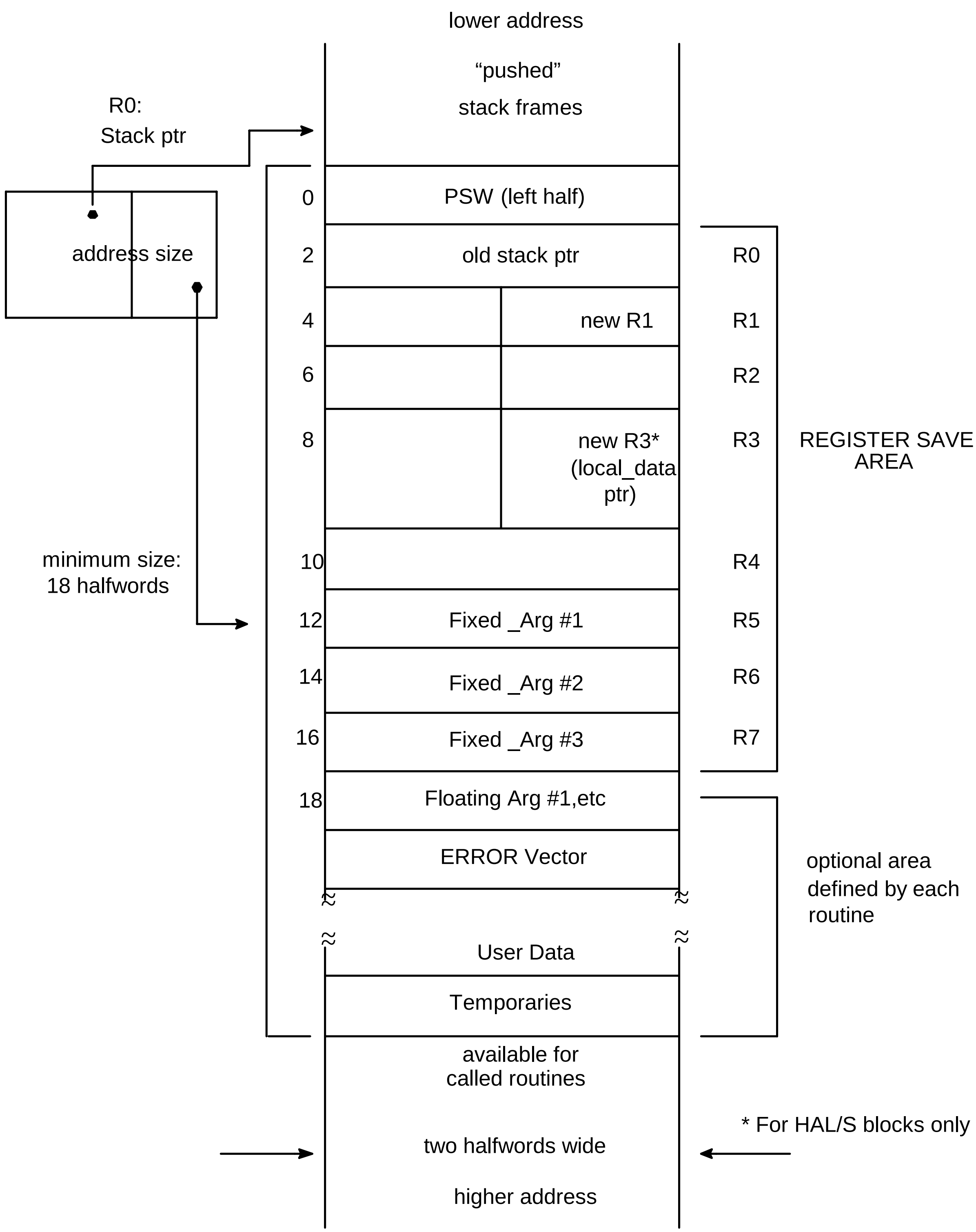


Figure 3-2 Stack Layout

3.1.1.7 Local Block Data Areas

During execution of a HAL/S-FC program, certain machine registers have dedicated uses as described in Section 3.1.1.1. In particular, register R3 is a local addressing register which points to the local Block Data Area for the block in execution. These R3 values are saved on the runtime stack as indicated in Section 3.1.1.6. The format of a local Block Data Area is the subject of this section. The HAL/SDL ICD contains the controlling definitions of these areas.

Block Data Areas are created by the compiler and are part of the #Dnnnnnn CSECT generated for a compilation unit. A Block Data Area may exist for any Program, Procedure, Function, Update Block, or Task. The compiler-emitted code for block entry (as defined in Section 3.1.1.6) loads R3 with the address of the Block Data Area for the block being entered. The format of such an area is shown in the following diagram.

<u>Fields</u>

BL	1	Block ID			2	
	2	XU	ONERRS	ERRDISP	2	
	3	TYPE	UNUSED	RESERVE SVC#	2	only required if XU=1
	4	UNUSED		RELEASE SVC#	2	
	5	LOCK ID			2	

Field	Definition
1. Block ID	A 16 bit field uniquely identifying the HAL block. The first 9 bits are a "compilation number" supplied by the user via the COMPUNIT compiler option. The last 7 bits are a block count generated internally for each new block within a compilation unit.
2. XU	EXCLUSIVE/UPDATE flag (1 bit). Set to one if block is either an UPDATE block or has the EXCLUSIVE attribute.
ONERRS	(6 bits). The number of discrete errors for which an ON ERROR statement exists in the block.
ERRDISP	(9 bits). The displacement in half words from the stack frame pointer register (R0) to the error vector.
3. TYPE	(1 bit). Set to zero for EXCLUSIVE procedure or function. If an UPDATE block, set to one if shared data variables are read only. Set to zero if shared data variables are to be written.
Reserve SVC#	(8 bits). SVC number for the reserve SVC: 15 for a code block 16 for a data area.
4. Release SVC#	(8 bits). SVC number for release SVC: 17 for a code block 18 for a data area.

5. Lock ID (15 bits). An indicator of which code block or data areas are being used. For a code block this is the address of the EXCLUSIVE DATA CSECT of the procedure/function. For a data area this is a bit pattern indicating which data areas (by lock groups) are involved. If the "master lock" was specified, the bit pattern will be all ones.

3.1.1.8 Parameter Passing Conventions for User-Written Routines

To the extent possible, HAL/S parameters are passed via registers. Scalar parameters are passed in floating point registers. All others are passed in general registers. The following rules describe how the registers are designated, and what they contain for each type of parameter.

General purpose registers 5-7 and floating point registers 0, 2, 4 are available for parameter passing. If the supply of registers is exhausted before the parameter list, the balance of the parameters are passed in memory locations. All parameters are located via the stack register (0).

Allocation of general and floating registers is carried on in parallel. If no scalar parameters exist, no floating point registers will contain parameters.

General purpose registers 5 through 7 are automatically contained in the stack beginning at displacement 12_{10}. Floating point registers are not automatically saved, and it is the responsibility of the called program to do so. Storage locations are reserved in the stack for this purpose as described below. Parameters which cannot be passed in registers are automatically stored in the called procedure's stack by the caller. The allocation of these stack locations is identical to the allocation for floating point values. Note that, unlike ordinary HAL/S variable allocation, parameter allocation is <u>not</u> subject to reordering to minimize alignment conflicts.

The first available stack location is at 18_{10} off the stack register. All parameters are assigned storage in order starting at this point (the exception being parameters contained in general registers 5 through 7, which are allocated space in the register save area as described above). Any necessary alignment is performed as needed.

Arguments are either input type or ASSIGN type (Input types are those whose values will not be changed by the called routine). The actual information which is passed for a particular argument is dependent upon the following factors:

- whether the argument is input or ASSIGN;
- whether the HAL/S data type of the argument is an aggregate (i.e. more than one element, as in a matrix);
- whether the argument has any arrayness or structure copies to be passed; and
- whether any arrayness or structure copies are defined via an ARRAY(*) or -STRUCTURE(*) specification.

The following table and list show the information which is passed for an argument with particular attributes.

Data Type / Argument Type	Integer	Scalar	Bit	Character(*)	Vector	Matrix	Structure
Input (no arrayness or copies)	1	2	3	4	5	6	7
ASSIGN (no arrayness or copies)	8	8	8	4	5	6	7
Input or Assign (with arrayness or copies)	9†	9†	9†	10†	9†	9†	11†

† If the parameter is declared as ARRAY(*) or -STRUCTURE(*), an additional parameter word is passed containing the value of the unspecified dimension.

<u>Key</u> <u>Information Passed</u>

1 A halfword or fullword of data.
2 A single or double precision floating point value.
3 Up to 32 bits of data (halfword or fullword depending upon declared size).
4 Address of the max-size byte of the character string.
5 Address of the 0th item in the VECTOR (i.e. 1 item width ahead of the actual vector).
6 Address of the 0th item as if the MATRIX were a VECTOR of length m x n.
7 Address of the first location in the structure as defined by its template. (Note that item position within a template is subject to compiler reordering unless RIGID is used).
8 Address of the data item.
9 Address of the 0th item of the array.
10 Two items are passed. The first is the address of the 0th array item. The second is the number of halfwords of memory occupied by one character string element (including the halfword containing the max and current size bytes).
11 The address of the first data in the 0th copy.

For all cases where auxiliary values are allocated for a single parameter (i.e. CHARACTER(*) ARRAY or ARRAY(*)), the parameters (up to 3) <u>must</u> be contiguous. Thus, if more pointers are required than registers are available, then the whole parameter sequence will be pushed into the stack.

Example:

```
P:     PROCEDURE(X,Y,I,J,K,Z,C,L);
       DECLARE SCALAR,X,Y,Z DOUBLE;
       DECLARE INTEGER,I,J ARRAY(*),K,L;
       DECLARE CHARACTER(*) ARRAY(*),C;
```

Upon entry to this procedure, the stack and registers are as follows:

	1 word		
$R0+12_{10}$	I	unused	also in R5
+14	address of 0^{th} array element of J	unused	also in R6
+16	size of array J	unused	also in R7
+18	X		also in F0
+20	Y		also in F2
+22	K	unused	
+24	1st word of Z		also in F4, F5
+26	2nd word of Z		
+28	address of 0^{th} array element of C	unused	
+30	# HW occupied by one element of C	unused	
+32	size of array C	unused	
+34	L	unused	

Figure 3-3

3.1.2 Integer and Scalar Operations

Nomenclature

The register R is any of the available set of accumulators. The terms I, I_2, S, and S_2 refer to the single and double precision versions of Integer and Scalar values respectively. It is assumed that any implicit precision or type conversions have been accomplished prior to generating the code sequences shown below.

3.1.2.1 Arithmetic Operators

Integer and scalar arithmetic operators generally employ two operands, denoted as X and Y. X is assumed to be loaded into register R_x unless otherwise noted. If Y is also in a register, it is represented by the form R_y.

Operation	Type	Code		Alternate Code	
X + Y:	I	AH	R_x,Y	AHI	R_x,Y†
	I_2	A	R_x,Y	AR	R_x,R_y
	S	AE	R_x,Y	AER	R_x,R_y
	S_2	AED	R_x,Y	AEDR	R_x,R_y
X- Y:	Similar to X + Y except that the subtract operator is used (For example, SH in place of AH in the above list).				
(Multiply)					
X Y:	I	MH	R_x,Y	MIH	R_x,Y†
		SLL	R_x,15		
	I_2	M	R_x,Y	MR	R_x,R_y
		SRDA	R_x,1		
	S	ME	R_x,Y	MER	R_x,R_y
	S_2	MED	R_x,Y	MEDR	R_x,R_y

† Used if Y is a literal.

Note that the shift operations used in the integer multiplications are required to correctly normalize the result in the proper registers

Certain constant multipliers are optimized to avoid using actual multiply instructions. They are described below.

Operation	Type	Code		Alternate Code	
	I 2^n	SLL	R_I,n,n>1		
		AR	R_I,R_I,n=1		
	I_2 2^n	SLL	R_I,n,n>1		
		AR	R_I,$R_{I,}$n=1		
	X 1	no code for any type			
	S 2	AER	R_s,R_s		
	S_2 2	AEDR	R_s,R_s		
X/Y	S	SER	R_x+1,R_x+1	SER	R_x+1,R_x+1
		DE	R_x,Y	DER	R_x,R_y
	S_2	DED	R_x,Y	DEDR	R_x,R_y

The DED and DEDR instructions are broken on the AP-101S machine. Thus, the compiler emits the following code sequences in place of these instructions:

Operation	Type	Code		Alternate Code	
X/Y	S_2	LER	R_{a+1}, R_{x+1}	LER	R_{a+1}, R_{x+1}
		LER	R_a, R_x	LER	R_a, R_x
		DE	R_X,Y	DER	R_X, R_y
		LER	R_{b+1}, R_{x+1}	LER	R_{b+1}, R_{x+1}
		LER	R_b, R_x	LER	R_b, R_x
		MED	R_b,Y	MEDR	R_b, R_y
		SEDR	R_b, R_a	SEDR	R_b, R_a
		DE	R_b,Y	DER	R_b, R_y
		SEDR	R_X, R_b	SEDR	R_X, R_b

where the result resides in the R_x,R_{x+1} register pair.

In the special case where a double precision result is divided by itself, a double precision "1" is loaded directly into the result register rather than executing the DED/DEDR workaround sequence. The code sequence for this case is:

Operation	Type	Code	
X/X	S_2	LFLI	R_x,1
		LFLI	R_{x+1},0

where the result resides in the R_x,R_{x+1} register pair.

X**Y: The exponentiation is performed by subroutine. The patterns shown for I and S are identical to those which will be generated for I_2 and S_2, except for the obvious differences:

```
I**I   LH      5,X          )
       CVFL    0,5*         |
       LH      6,Y          |
S**I   LH      6,Y (see note) }  Argument Setup
       LE      0,X          |
S**S   LE      2,Y          |
       LE      0,X          )
       ACALL   αPWRβ        }  Actual Call
```

where α and β represent the types of operands X and Y respectively:

Type of X	α	Type of Y	β
Single precision integer	E	single precision integer	H
double precision integer	E	double precision integer	I
single precision scalar	E	single precision scalar	E
double precision scalar	D	double precision scalar	D
single precision integer	H*		
double precision integer	I*		

Return is in F0 for α of E or D; in R5 for α of H or I.

* if Y operand is a positive integer literal, the CVFL conversion is eliminated and the PWR routine invoked is αPWRH or αPWRI.

Note: Scalar expressions raised to integer literal powers from 1 to 16 are performed in line via repeated multiplication, using the binary powers algorithm. The following examples should serve to illustrate the method.

Operation	Type	Code	Alternate Code
X**1:		No code generated.	
X**2:	S	MER R_x, R_x	
X**3:	S	LER R_T, R_x	
		MER R_x, R_x	
		MER R_T, R_x	
		(result in R_T)	
X**6:	S	MER R_x, R_x	
		LER R_T, R_x	
		MER R_x, R_x	
		MER R_T, T_x	
		(result in R_T)	

For type S_2, the instruction MEDR is used in place of MER. Two LERs must be used in place of one.

Operation	Type	Code
+X		No code generated
-X	I, I_2	LACR R_x, R_x
	S	LECR R_x, R_x
	S_2	LED R_x, X
		LECR R_x, R_x

3.1.2.2 Comparison Operators

The full complement of relational operators is allowed for Integer or Scalar operations between single quantities. Only equal or not equal operators are allowed for arrayed comparisons. No logical variables are created by comparisons. Instead, branching to one of two points is used for true/false relations.

Operation	Type	Code		Alternate Code	
X <OP> Y:	I	CH	R_x,Y		
		BC	COND,not-true-label		
	I_2	C	R_x,Y	CR	R_x,R_y
		BC	COND,not-true-label		
	S	CE	R_x,R_y	CER	R_x,R_y
		BC	COND,not-true-label		
	S_2	SED	R_x,Y	SEDR	R_x,R_y
		BC	COND,not-true-label		

Note: For comparisons to the literal 0, the condition code is used directly. If the condition code is not valid, the instruction LR or LER is used to set it.

3.1.2.3 Conversions

Where necessary, conversions are performed in intrinsic or library functions. Some conversions do not require any generation of code.

Integer Conversions

Operation	Code	
I TO S	LH	R_x,X
	CVFL	F_x,R_x
I TO S_2	LH	R_x,X
	CVFL	F_x,R_x
	SER	F_{x+1},F_{x+1}
I_2 TO S	L	5,X
	ACALL	ITOE
I_2 TO S_2	L	5,X
	ACALL	ITOD
I,I_2 TO BIT	No code necessary	
I TO CHAR	LH	5,X
	LA	2,temp-string-area†
	ACALL	HTOC
I_2 TO CHAR	L	5,X
	LA	2,temp-string-area†
	ACALL	ITOC
I TO I_2	SRA	R_x,16
I_2 TO I	SLL	R_x,16

† temp-string-area contains converted string.

Scalar Conversions

Operation	Code		Alternate Code
S TO I, I_2	LE	0,X	LER 0,R_x
	ACALL	αTOβ	
S_2 TO I, I_2	LED	0,X	LEDR 0,R_x
	ACALL	αTOβ	

TYPE OF SCALAR	α	TYPE OF INTEGER	β
Single Precision	E	Single Precision	H
Double Precision	D	Double Precision	I

```
S,S2 TO BIT  Same as for scalar to integer

S TO CHAR    LE       0,X
             LA       2,temp-string-area†
             ACALL    ETOC

S2 TO CHAR   LED      0,X
             LA       2,temp-string-area†
             ACALL    DTOC

S  TO  S2    LE       Rx,X
             SER      Rx+1,Rx+1
```

† temp-string-area contains converted string.

3.1.2.4 Assignments

For all assignments, type conversion may take place across the assignment operator. For multiple assignments, the left hand side operands are grouped by data type to minimize the number of conversions performed. The order in which the groups are processed is determined by the following table:

Right Hand Operand Type

Left Hand Type Ordering	I	I_2	S	S_2
First	I	I_2	S	S_2
	I_2	Char	Char	Char
	Char	S_2	S_2	S
	S_2	S	I_2	I_2
	S	I	I	I
Last	Vector-Matrix			

Character is always performed before any right hand side conversion is performed.

The following sequences assume that R_x has already had the required integer or scalar conversions performed as described in Section 3.1.2.3.

Operation	Type of Y	Code	
Y = X;	I†	STH	R_x,Y
	I_2	ST	R_x,Y
	S	STE	R_x,Y
	S_2	STED	R_x,Y

† If X is an integer literal of value 0 or-1, then the following code will be generated:

Y = 0;	I	ZH	Y
Y = -1;	I	SHW	Y

R_x is also marked as now containing the value Y. Subsequent usages of Y may use this register in lieu of the copy of Y in memory until such time as the contents of this register are destroyed or a label is generated.

3.1.3 Bit String Operations

3.1.3.1 Bit String Operators

Bit string operators include the following: AND (&), OR (|), and CAT (||). They generally employ two operands, denoted here as X and Y (of lengths N_x and N_y respectively). X is assumed to be loaded into register R_x unless otherwise noted. If Y is also in a register, it is represented as R_y. Note that the & and | operations will pad the bit length of the shorter bit string to the length of the longer bit string.

Operation	Bit Length	Code		Alternate Code	
X&Y	N_x,N_y≤16	NR	R_x,R_y	NHI	R_x,'Y'†
	N_x,N_y>16	N	R_x,Y	NR	R_x,R_y
X|Y	N_x,N_y≤16	OR	R_x,R_y	OHI	R_x,'Y'†
	N_x,N_y>16	O	R_x,Y	OR	R_x,R_y
X||Y	N_y≤16	SLL	R_x,N_y		
		OR	R_x,R_y		
	N_y>16	SLL	R_x,N_y		
		O	R_x,Y	OR	R_x,R_y

† Used only when Y is a bit literal.

3.1.3.2 Bit String Comparisons

The only possible relational operators for bit strings, as with bit operators, are = or Ø = (see Section 3.1.1.4). The bit strings are padded to be of equal lengths. No logical variables are created by comparisons. Instead, branching to the "not-true-label" occurs with the "not true" condition.

Operation	Bit Length	Code		Alternate Code	
X<OP>Y	$N_x, N_y \leq 16$	CH	R_x,Y	CHI	R_x,'Y'†
		BC	COND,not-true-label		
	$N_x, N_y > 16$	C	R_x,Y	CR	R_x, R_y
		BC	COND,not-true-label		

† Used only when Y is a bit literal.

3.1.3.3 Component Subscripting

Component subscripting for bit strings consists of shifting and &'ing out unwanted components of the subscripted bit string. The resultant bit string length, N_r, determines a binary mask, whose decimal value is $2^{N_r}-1$, and bit number "I" of the original bit string is the last component of the resultant bit string.

Operation	Bit Length	Code	
$X_{subscript}$	N_x	SRL	R_x, N_x-I
		N	R_x,mask†
$X_{variable}$ subscript	N_x	LACR	R_I, R_I
		AHI	R_I, N_x
		SRL	R_x,0(R_I)
		N	R_x,mask†

† The mask value is equal to (2^{N_r}-1)

Examples of Subscript Forms		
Subscript	I	N_r
3 TO 10	10	8
6 AT 11	16	6
9	9	1
8 AT J	J + 7	8
K	K	1

3.1.3.4 Bit Conversions

When necessary, conversions are performed in intrinsic or library functions. Some conversions do not require any generation of code.

Operation	Bit Length	Code		Alternate Code	
BIT TO I		No code necessary			
BIT TO I_2		LH	R_x,X	SRA	R_x,16
		SRA	R_x,16		
BIT TO S,S_2	$N_x \le 16$	LH	5,X	LR	5,R_x
		CVFL	0,5		
	$N_x > 16$	L	5,X	LR	5,R_x
		ACALL	ITOE		
BIT TO CHAR	$N_x \le 16$	LH	5,X		
		SRL	5,16	set up of bit-type argument	
	$N_x > 16$	L	5,X		
		LA	2,temp-string-area†	actual calling sequence	
		LHI	6,N_x		
		ACALL	BTOC		
BIT TO CHAR@<radix>		Same as BIT TO CHAR except call to BTOC is replaced as follows:			
		<radix>	routine		
		BIN	BTOC		
		OCT	OTOC		
		DEC	KTOC		
		HEX	XTOC		
BIT TO BIT	$\begin{bmatrix} N_x > N_y \\ N_y \le 16 \end{bmatrix}$	NHI	$R_x,2^{N_y}-1$		
	$N_y > 16$	N	R_x,mask††		

† temp-string-area contains converted string.

†† The value of the mask is $2^{N_y}-1$.

3.1.3.5 Bit Assignments

The following sequences assume that R_x has already had the required conversions performed as described in Sections 3.1.3.3 or 3.1.3.4**.**

Operation	Length of Bit String Y	Code	
Y = X	$N_y \le 16$	STH	R_x,Y†
	$N_y > 16$	ST	R_x,Y†

† Note: If $N_x > N_y$ and N_Y is not exactly 16 or 32, then the following instruction must be added: NR$_x$, F'2^{N_r}-1'.

If the right hand side of the assignment (X) is a BIT literal as described below, and $N_y \leq 16$, then the following code is generated:

```
Y = BIN'0';        Ny<=16         ZH    Y
Y = BIN(16)'1';    Ny=16          SHW   Y
```

3.1.3.6 Partitioned Bit Assignments

The following sequences assume that R_x has already had the required conversions performed as described in Section 3.1.3.3 or 3.1.3.4. Definitions of I, N_y, and N_r are as described in Section 3.1.3.3.

Operation	Length of Bit String Y	Code	
$Y_{subscript}$=X;	$N_y \leq 16$	LH	R_x,X
		LH	R_y,Y
		SLL	R_x,N_y-I
		XR	R_x,R_y
		NHI	R_x,mask†
		XR	R_x,R_y
		STH	R_x,Y
$Y_{subscript}$=X;	$17 \leq N_y \leq 32$	L	R_y,Y
		L	R_x,X
		SLL	R_y,N_y-I
		XR	R_y,R_x
		N	R_y,mask††
		XR	R_y,R_x
		ST	R_y,Y

† Mask: The mask used in a bit store is computed as follows: $(2^{N_r}-1)(2^{N_x-I})$. In other words, the mask is a sequence of N_r bits shifted left N_x-I bits.

†† The value of the mask is 2y-1.

If the right hand side of the assignment (X) is a BIT literal containing either BIN'0' or BIN(N_y) '1' then if $N_y \leq 16$ and Y is addressable in SRS format, then the following code is generated:

$Y_{11\ TO\ 13}$=BIN'0';	N_y=16	ZB	Y,B'111000'
$Y_{10\ TO\ 12}$=BIN'111';	N_y=16	SB	Y,B'1110000'

If N_y>16 then the following code is generated:

$Y_{13\ TO\ 20}$=BIN'0';	N_y=32	L	R_x,=X'FFF00FFF'
		NST	R_x,Y
$Y_{17\ TO\ 20}$=BIN'111';	N_y=32	L	R_x,=X'00007000'
		OST	R_x,Y

3.1.3.7 Bit Tests

```
IF X          Nx=1                    TH    X
                                      BZ    <not true label>

IF X10        Nx=16                   TB    X,B'1000000'
                                      BZ    <not true label>
                      or
                                      LH    Rx,X
                                      SRL   R,6
                                      NHI   R,B'1'
                                      BZ    <not true label>

IF ¬ X        Same as IF X except BZ changed to BNZ instruction.
```

3.1.4 Character String Operations

3.1.4.1 Character String Operators

The only character string operator is the CAT (||) operator employing two character string operands denoted here as X and Y (of lengths N_x and N_y respectively). Unless otherwise noted, X is assumed to be loaded into register R_x. If Y is also in a register, it is represented as R_Y.

```
Operation  Code
X || Y     LA      P3,Y
           LA      P2,X
           LA      P1,temp-string-area
           ACALL   CATV
```

3.1.4.2 Character String Comparisons

The full set of relational operators are allowed for character strings (see Section 3.1.1.4 for condition codes). Characters with different lengths are always unequal. No logical variables are created by comparisons. Instead, branching to the "not-true-label" occurs with the "not true" condition.

Operation: X <OP> Y

<OP>	α
=, ¬=	
<,>, <=, >=	C

```
Code
LA      P3,Y
LA      P2,X
ACALL   CPRα
BC      COND,not-true-label
```

3.1.4.3 Component Subscripting

Component subscripting for character strings consists of setting the initial, N_i, and final, N_f, index values of the subscripted components into registers 5 and 6 respectively, and then branching to the CASP intrinsic.

Operation	Code	Alternate Code	
$Y=X_{subscript}$;	LA	P1,Y	
	LH	5,N_i	
	LH	6,N_f	LR 6,5} if only one component
	ACALL	CASP	

3.1.4.4 Character String Conversions

Where necessary, conversions are performed in intrinsic or library functions.

Operation	Code
CHAR TO I	LA 2,char ACALL CTOH
CHAR TO I_2	LA 2,char ACALL CTOI
CHAR TO S	LA 2,char ACALL CTOE
CHAR TO S_2	LA 2,char ACALL CTOD
CHAR TO BIT	LA 2,char ACALL CTOB
CHAR TO $BIT_{@<radix>}$	Same as CHAR TO BIT except call to BTOC is replaced as follows:

<radix>	routine
BIN	CTOB
OCT	CTOO
DEC	CTOK
HEX	CTOX

3.1.4.5 Character String Assignments

Either the receiver variable or the assigned variable in a character string assignment may be subscripted. The possible forms are shown below. When subscripting is used, a partitioning of a character string results. The initial element of this partitioned character string is signified by its index: N_i. Similarly the final element has the index N_f. Some examples of HAL/S subscript forms and the resulting N_i and N_f values are:

Subscript Form	N_i	N_f
1 TO 3	1	3
5 AT 2	2	6

Operation	Code	
Y=X	LA	P2,X
	LA	P1,Y
	ACALL	CAS†
$Y_{subscript}$=X	LA	P2,X
	LA	P1,Y
	LHI	5,N_{iy}
	LHI	6,N_{fy}
	ACALL	CPAS†
Y=$X_{subscript}$	LA	P2,X
	LA	P1,Y
	LHI	5,N_{ix}
	LHI	6,N_{fx}
	ACALL	CASP†
$Y_{subscript}$=$X_{subscript}$	LA	P2,X
	LA	P1,Y
	LHI	5,N_{ix}
	LHI	6,N_{fx}
	L	7,H'N_{iy},,N_{fy}'
	ACALL	CPASP[15]

† For REMOTE data, CASR is called instead of CAS, CASRP for CASP, etc.

3.1.5 Vector Matrix Operations

3.1.5.1 Vector-Matrix Operators

Vector Matrix operators usually operate on two arguments according to the conventions stated in Section 5.2. Since 3-vectors, and 3x3-matrices have special library routines, their code is listed in the column labeled "3-code", while the code for any other vectors or matrices is listed in the "n-code" column.

Operation	Type	n-code			3-code	
V1+V2	single	L	R,=H'1,n-1'		L	R,=H'1,2'
	loop:	LE	FR,V2(R)	loop:	LE	FR,V2(R)
		AE	FR,V1(F)		AE	FR,V1(R)
		STE	FR,temp-area(R)		STE	FR,temp-area(R)
		BIX	R,loop		BIX	R,loop
V1+V2	double	L	R,=H'1,n-1'		L	R,=H'1,2'
	loop:	LED	FR,V2(R)	loop:	LED	FR,V2(R)
		AED	FR,V1(R)		AED	FR,V1(R)
		STED	FR,temp-area(R)		STED	FR,temp-area(R)
		BIX	R,loop		BIX	R,loop
V1-V2	Same as V1+V2 except that an SE instruction is used instead of an AE instruction for single precision. For double precision, an SED is used instead of an AED.					
-V1	single	L	R,=H'1,n-1'		L	R,=H'1,2'
	loop:	LE	FR,V1(R)	loop:	LE	FR,V1(R)
		LECR	FR,FR		LECR	FR,FR
		STED	FR,temp-area(R)		STED	FR,temp-area(R)
		BIX	R,loop		BIX	R,loop
-V1	double	Same as -V1 single, except that an LED is used in place of the LE instruction.				
V1 V2	single	LA	P3,V2		LA	P3,V2
V1: length n		LA	P2,V1		LA	P2,V1
V2: length m		LA	P1,temp-area		LA	P1,temp-area
result is nxm		LHI	5,n		ACALL	VO6S3
matrix		LHI	6,m†			
		ACALL	VO6SN			
V1 V2	double	Same as for single precision, except that the routines branched to are VO6DN and VO6D3 for n-vectors and 3-vectors respectively.				
V1*V2	single	(illegal operation)			LA	P3,V2
					LA	P2,V1
					LA	P1,temp-area
					ACALL	VX6S3
V1*V2	double	Same as for single precision, except that VX6D3 is branched to, rather than VX6S3.				
V1•V2	single	LA	P3,V2		LA	P3,V2
		LA	P2,V1		LA	P2,V1
		LHI	5,n		ACALL	VV6S3††
		ACALL	VV6SN††			

† If both V1 and V2 are the same size, then this instruction will be: LR 6,5.

†† The scalar result of the dot product is left in register F0.

Operation	Type	n-code	3-code
V1•V2	double	Same as for single precision, except that the routines branched to are VV6DN and VV6D3 for n-vectors and 3-vectors respectively.	
M1+2 OR M1-2	Same code as that for adding or subtracting two vectors of length equal to the product of the row size and the column size of M1 and M2.		

V1 M2
V1:length n
M2:nxm
	single	`LA P3,M2`
		`LA P2,V1`
		`LA P1,temp-area`
		`LHI 5,n`
		`LHI 6,m`†
		`ACALL VM6SN`
			`LA P3,M2`
			`LA P2,V1`
			`LA P1,temp-area`
			`ACALL VM6S3`

V1 M2	double	Same as for single precision, except that the routines branched to are VM6DN and VM6D3 for the general case and the size 3 cases respectively.	

M1 V1
M1:nxm
V1:m
	single	`LA P3,V1`
		`LA P2,M1`
		`LA P1,temp-area`
		`LHI 5,n`
		`LHI 6,n`
		`ACALL MV6SN`
			`LA P3,V1`
			`LA P2,M1`
			`LA P1,temp-area`
			`ACALL MV6S3`

M1 V1		double Same as for single precision, except that routines branched to are MV6DN (n code) and MV6D3 (3 code).	

V1 I††,
V1 I2††,
V1 S
	single
	
	loop:
		`L R,=H'1,n-1'`
		
		`LE FR,V1(R)`
		`ME FR,S`
		`STE FR,temp-area(R)`
		`BIX R,loop`
			`L R,=H'1,2'`
			
			`loop: LE FR,V1(R)`
			`ME FR,S`
			`STE FR,temp-area(R)`
			`BIX R,loop`

V1 S2	double
	loop:
		`L R,=H'1,n-1'`
		`LED FR,V1(R)`
		`MED FR,S2`
		`STED FR,temp-area(R)`
		`BIX R,loop`
			`L R,=H'1,2'`
			`loop: LED FR,V1(R)`
			`MED FR,S2`
			`STED FR,temp-area(R)`
			`BIX R,loop`

† If M2 is of size nxn, then this instruction is: LR 6,5.

†† Note that in the case of single and double precision integers, they are first converted to scalar form whose value is in F0.

Operation	Type	n-code		3-code	
V1/I… V1/S…	Same as for V1 I, etc., except that a DE instruction is used instead of ME. (DED[†] is used instead of MED for double precision.)				
I V1,I2 V1, S V1,S2 V1		Exactly the same as for V1 I, etc.			
M1 I,M1 I2, M1 S,M1 S2		Same as for V1 I, etc., except that the length value (n) is the product of the row size and the column size of M1.			
M1/I,M1/I2 M1/S,M1/S2		Same as for V1/I, etc., except that the length value (n) is the product of the row size and the column size of M1.			
I M1,I2 M1 S M1,S2 M1		Exactly the same as for V1 I, etc., except that the length specified in R5 is equal to the product of the row size and the column size of M1.			
M1**i (where i is either a literal or a constant integer)	single	LHI LA LA LA LHI ACALL	6,i P3,temp-storage-area P2,M1 P1,temp-storage-area 5,n MM17SN	Same as for "n-code" where n=3.	
M1**I	double	Same as for single precision, except branches to the MM17DN.			
M1**0	single	LA LA LHI ACALL	P2,M1 P1,temp-storage-area 5,n MM15SN		
M1**0	double	Same as for single precision, except branches to MM15DN.			
M1**T M1: m x n	single	LA LA LA LA ACALL	P2,M1 P1,temp-storage-area 5,n 6,m MM11SN	LA LA ACALL	P2,M1 P1,temp-storage-area MM11S3
M1**T	double	Same as for single precision, except that the routine branched to is either MM11DN or MM11D3 for n x n matrices and 3 x 3 matrices respectively.			

[†] See Section 3.1.2.1 for important information regarding the DED instruction.

Operation	Type	n-code		3-code	
M1 M2	single	LA	P3,M2	LA	P3,M2
M1: k x m		LA	P2,M1	LA	P2,M1
M2: m x n		LA	P1,temp-area	LA	P1,temp-area
		LHI	5,k	ACALL	MM6S3
		LHI	6,m[†]		
		LHI	7,n[†]		
		ACALL	MM6SN		
M1 M2	double	Same as for single precision, except that the routines branched to are MM6DN and MM6D3 for the general case and the 3 x 3 case respectively.			

† Either of the instructions may be of the form: LR 6,5 if n=k, etc.

3.1.5.2 Conditional Operators

The only comparison operators allowed for comparing vector and matrices are = or ¬=. Since these comparisons are done on an element-by-element basis, the same routines that are used for size-n vectors are also used for size n x m matrices which are considered to be vectors of length n x m. No logical variables are created by comparisons. Instead, branching to the "not-true-label" occurs with the "not true" condition.

Operation	Type	n-code		3-code	
V1 <OP> V2	single	LA	P3,V2	LA	P3,V2
		LA	P2,V1	LA	P2,V1
		LHI	5,n	ACALL	VV8S3
		ACALL	VV8SN	BC	COND,not-true-label
		BC	COND,not-true-label		
V1<OP>V2	double	Same as for single precision, except that the routines branched to are VV8DN and VV8D3 for n-vectors and 3-vectors respectively.			
M1<OP>M2	single	LA	P3,M2	LA	P3,M2
M1,M2:mxn		LA	P2,M1	LA	P2,M1
		LHI	5,mxn	LHI	5,9
		ACALL	VV8SN	ACALL	VV8SN
		BC	COND,not-true-label	BC	COND,not-true-label
M1<OP>M2	double	Same as for single precision, except that the routine branched to is VV8DN.			

3.1.5.3 Component Subscripting

Possible components of matrices include submatrices, vectors, column vectors, and single components. Possible components of vectors include subvectors and single components. The resultant type of component is determined by the subscripts used. Note that double precision operations are not shown - their code is identical except that: a) the called routines will be VV1DN rather than VV1SN, etc.; b) the index multiplier is 4 instead of 2. Register 7, when used, contains skip values between elements in partitioned matrices (see Section 3.1.1.3).

Operation[†]		n-code	3-code	
$Y=V_{x_i}$	LE	R_x,V_x+2i	N.A.	
	STE	R_x,Y		
$Y=V_{x_i}$	LH	R_I,I		
	LE	R_x,V_x(R_I)	N.A.	
	STE	R_x,Y		
V_{Y_i}=X;	LH	R_I,I		
	LE	R_x,X	N.A.	
	STE	R_x,V_Y(R_I)		
$V_{Y_{n\ AT\ I}}=V_{x_{n\ AT\ I}}$;	LH	R_I,I	LH	R_I,I
	AR	R_I,R_I	AR	R_I,R_I
	LA	P2,V_x(R_I)	LA	P2,V_x(R_I)
	LA	P1,V_y(R_I)	LA	P1,V_y(R_I)
	LHI	5,n	ACALL	VV1S3
	ACALL	VV1SN		
$M_y=M_{x_{m\ AT\ I,n\ AT\ J}}$				
assumes M_y is an m by n MATRIX				
	LH	R_I,I	<same>	
	MHI	R_I,<column size of M_x>		
	SLL	R_I,15		
	AH	R_I,J		
	AR	R_I,R_I		
	LA	P2,M_x(R_I)		
	L	7,F'delta,0'		
	LA	P1,M_y		
	LHI	5,m		
	LHI	6,n		
	ACALL	MM1SNP		

† i indicates integer literal, I indicates integer variable.

Operation[†]			n-code	3-code
$M_{x*,I}=V_x$;	LH	R_I,I	LH	R_I,I
	AR	R_I,R_I	AR	R_I,R_I
	LA	P2,V_x	LA	P2,V_x
	LHI	6,0	LHI	6,0
	LHI	7,delta	LHI	7,delta
	LA	P1,M_x(R_I)	LA	P1,M_x(R_I)
	LHI	5,n	ACALL	VV1S3P
	ACALL	VV1SNP		

† i indicates integer literal, I indicates integer variable.

3.1.5.4 Conversions

MATRIX/VECTOR conversions are done by considering matrices as vectors, and assigning the required components to the receiver variable. More than 1 argument requires multiple calls to the vector assign routine (as shown in the second sequence below). Use of double precision operands will cause branches to VV1DN. Otherwise, the code is unchanged.

Operation	n-code	
VECTOR(M_x)	LA	P2,M_x
Produces vector of size equal to product of dimension of matrix: n x m.	LA	P1,temp-area
	LHI	5,nxm
	ACALL	VV1SN
MATRIX(V_x,V_y,V_z)[†]	LA	P2,V_x
	LA	P1,temp-area
	LHI	5,n_x
	ACALL	VV1SN
	LA	P2,V_y
	LA	P1,temp-area+DELTA1
	LHI	5,n_y
	ACALL	VV1SN
	LA	P2,V_z
	LA	P1,temp-area+DELTA2
	LHI	5,n_z
	ACALL	VV1SN

† This is An example using several vectors to illustrate the multiple calling of the VV1SN (or VV1S3) routine. It also applies to the VECTOR shaping functions.

3.1.5.5 Assignments

Vectors and matrices may be assigned to other vectors and matrices of the same dimensions. In addition, they may have all elements set to zero by a statement of the form:

$\overset{*}{M}$=0; or $\overline{V}$=0;

Operation	Type	n-code			3-code	
V_x=V_y	single	L	R,=H'1,n-1'		L	R,=H'1,2'
	loop:	LE	FR,V_y(R)	loop:	LE	FR,V_y(R)
		STE	FR,Vx(R)		STE	FR,Vx(R)
		BIX	R,loop		BIX	R,loop
V_x=0	single	L	R,=H'1,n-1'		L	R,=H'1,2'
	loop:	SER	FR,FR		SER	FR,FR
		STE	FR,V_x(R)		STE	FR,V_x(R)
		BIX	R,loop		BIX	R,loop
M_x=M_y		Same as for V_x=V_y, except that the loop count, n-1, is replaced by (m n)-1.				
M_x=0		Same as for V_x=0, except that the loop count, n-1, is replaced by (m n)-1.				
V_x=V_y M_x=M_y	double	Same as single, but use LED, STED sequence instead of LE, STE instructions.				
V_x=0 M_x=0	double	Same as single, but use SEDR, STED sequence instead of SER, STE instructions.				

For the following operations the compiler will attempt to generate in-line code sequences, including as many operations within a single loop as possible:

```
VECTOR/MATRIX ADD
VECTOR/MATRIX SUBTRACT
VECTOR/MATRIX NEGATE
VECTOR/MATRIX-SCALAR PRODUCT
VECTOR/MATRIX-SCALAR DIVIDE
VECTOR/MATRIX ASSIGNMENT
```

In many cases, the stores into temp-areas, as shown in the prototype instruction sequences, will not be necessary, unless the resultant VECTOR or MATRIX needs to be passed from one loop to another, or to a library routine. For example:

```
HAL/S                                  Code
DECLARE
VECTOR,V,W,X,Y,Z;
V=V+(W+X)*Y-Z;                  L      R,=H'1,2'
                            L1  LE     FR,W(R)
                                AE     FR,X(R)
                                STE    FR,temp1(R)
                                BIX    R,L1
                                LA     P3,4
                                LA     P2,temp1
                                LA     P1,temp2
                                ACALL  VX6S3
                                L      R,=H'1,2'
                            L2  LE     FR,V(R)
                                AE     FR,temp2(R)
                                SE     FR,Z(R)
                                STE    FR,V(R)
                                BIX    R,L2
```

In those cases where in-line code is not generated, the temporary area used to store the result of the last HALMAT operation before an assignment can be eliminated if the vector-matrix statement is of a suitable "form" for optimization and one of four conditions holds. The statement may not have multiple receivers; the single receiver must be a consecutive partition or be nonpartitioned. The precision of the right-hand-side of the statement must match the precision of the receiver. The receiver cannot be a remote variable, and neither the receiver nor the operand(s) of the final HALMAT operation can be name variables, or the terminal of a subscripted structure. Also, variable subscripts on any variables do not allow optimization processing to continue.

Statements that meet these basic requirements can then be checked for the occurrence of a necessary and sufficient condition for optimization. The result of the final operation before the assignment will be stored directly in the receiver if at least one of the following conditions is true:

1. The receiver is nonpartitioned and the last operation before the assignment HALMAT is a "Class 3" operation. Class 3 operations include matrix-scalar and vector-scalar multiplication and division, vector-matrix addition and subtraction, vector and matrix negation and the built-in function, UNIT.

   The last operation is a "Class 1" operation. The class contains only "matrix raised to 0th power." The result, the identity matrix, can be stored directly in any consecutive receiver.

2. The operand(s) are in temporary work areas. Nonconsecutive partitions are moved to work areas when the operands are processed. The result of a previous operation is also in a work area. Operands in work areas are disjoint from the receiver. This is important for "class 2" operations that use the elements of the vector or matrix, vector-vector, and matrix-matrix arithmetic, and matrix transpose and exponentiation (also, the built-in functions, TRANSPOSE and INVERSE). This condition can also hold for class 1 and class 3 operations. If the operation has two operands, <u>both</u> must be in work areas for this condition to be true.
3. The operand(s) are nonidentical to the receiver. A receiver-operand pair is nonidentical if the operand is in a work area, or if neither variable is a formal parameter and the variables have different symbol table references, or if <u>only one</u> of the variables in a formal parameter and the NEST level of the nonparameterized variable is greater than or equal to the NEST level of the parameterized variable (again, symbol table reference cannot be the same).

```
EXAMPLE1: PROGRAM;
   DECLARE MATRIX(3,3),S,T;
   PROC: PROCEDURE(A) ASSIGN(B);
         DECLARE MATRIX(3,3),A,B,C;
         SUBPROC: PROCEDURE(X) ASSIGN(Y);
                  DECLARE MATRIX(3,3),X,Y,P,Q;
                  Y2 TO 3,* = X2 TO 3, * + C2 TO 3,*;
                  B2 TO 3,* = P2 TO 3, * + Q2 TO 3,*;
         CLOSE SUBPROC;
   CALL SUBPROC(A) ASSIGN(C);
   CLOSE PROC;
   CALL PROC(S) ASSIGN(T);
CLOSE EXAMPLE1;
```

where

```
   X&Y are parameters, C is not
       NEST_LEVEL(Y)=2,
       NEST_LEVEL(C)=1.
   Y can be C - cannot assign directly.
   P&Q not parameters - ok to assign directly
       NEST_LEVEL(P)=2,
       NEST_LEVEL(A)=1.
```

4. The operand(s) are disjoint with the receiver. A receiver-operand pair can be disjoint in two ways. If the pair is nonidentical it is, by default, disjoint. If <u>both</u> the receiver and the operand are consecutively partitioned, they are disjoint if the partitions do not overlap in any way. If the receiver and the operand have the same symbol table reference (are identical) then the two partitions can be disjoint in either "direction".

   For example, let A be a 4-by-4 matrix. Then,

   $A_{1\ TO\ 2,*} = A_{3\ TO\ 4,*} + \ldots$ and
   $A_{3\ TO\ 4,*} = A_{1\ TO\ 2,*} + \ldots$ and both disjoint pairs.

If the receiver and operand are possibly identical, then the pair can only be disjoint if all of the operand partition comes after the receiver partition.

```
EXAMPLE2: PROGRAM;
   DECLARE MATRIX(6,3),A,D,E;
   PROC: PROCEDURE(B,C);
      DECLARE MATRIX(4,3),B,C;
      A1 TO 2,* = B3 TO 4,* + C3 TO 4,*;   Pairs A-B & A-C disjoint
      A3 TO 4,* = B1 TO 2,* + C3 TO 4,*;   Pair A-B not necessarily disjoint
   CLOSE PROC;
   CALL PROC(A3 TO 6,*,D3 TO 6,*);      (B1 TO 2,* is really A3 TO 4,*)
   A3 TO 4,* = D3 TO 4,* + E1 TO 2,*;    A,D,E are, by default, disjoint
                                        because they are nonidentical
     CLOSE EXAMPLE2;
```

If the operation has two operands, both receiver-operand pairs must be disjoint for this condition to be true. The nonidentical and disjoint checks are made at the same time, so this condition also holds if one pair is disjoint by disjoint partitioning and one pair is disjoint by being nonidentical.

3.1.6 Structure Operations

3.1.6.1 Structure Comparisons

Structure comparisons may only be = or ¬=. The comparisons are done by comparing corresponding terminal elements of the two structure operands in order of their natural sequence. Each terminal element is referenced by adding the displacement of the element to the address of the structure (see Section 3.1.1.3). No logical variables are created. Instead, branching to the "not-true-label" occurs with the "not-true" condition.

```
Operation                      Code
X<OP>Y                         LA    2,X
                               LA    3,Y
                              {LA   2,terminal#1(X)
                              |LA   3,terminal#1(Y)
          for each terminal   {LHI  5,width
                              |BAL  4,CSTRUC
                              {BC   COND,not-true-label
                               .
                               .
                               .
                               <same for all terminals>
                               .
                               .
                               .
                               BC  7,true-label
```

3.1.6.2 Structure Assignment

The assignment of both major and minor structures is done via the BLOCK MOVE algorithm. (Generally, this is an MVH code sequence.)

Operation		Code	
Y=X		L	R_y,Const1
(neither X nor Y REMOTE)		L	R_x,Const2
		MVH	R_y,R_x
		.	
		.	
		.	
		DC	Y(Y)
	const1	DC	H'n'†
		.	
		.	
	const2	DC	Z(x)
Y=X		LA	P2,X
(X or Y REMOTE)		LA	P1,Y
		LHI	5,width
		ACAL	MSTR
		L	

† n is width of x in halfwords.

3.1.7 Indexing and Arrayed Statements

3.1.7.1 Linear Array Indexing

Linear array indexing is the use of subscripts, on an arrayed data type, to produce a one-dimensional resultant array. In the generated code, only one register - R_a - is needed to keep track of the index value. An initial entry to the array loop (see Section 3.1.7.4), R_a is initialized to a value of 1. On each pass through the loop, R_a is used to define a DELTA value to index the arrayed data (see Section 3.1.3.3). Following this, at the end of the loop R_a is incremented by 1, and is tested to determine if all of the data has been utilized, as described in Section 3.1.7.4. R_a is any available indexing register. Its contents may not be altered during the course of an arrayed statement. If the index in R_a must be shifted to access the word or doubleword data, it must be moved to another register to perform this shift.

3.1.7.2 Non-Linear Array Indexing

Non-linear array indexing has more than one index which can change values to produce a multi-dimensional resultant array. The actual code generated, though, can only utilize one register - R_a - for indexing. Thus, temporary storage is needed to store all but the inner-most index. As with linear indexing, all index values (both in R_a and temporary storage) are initialized to 1. The DELTA value defining the index of each arrayed data item is then computed on the basis of the value of R_a and the index values stored in memory (see Section 3.1.3.3). Following this, each index value is tested against the size of the corresponding dimension (of the resultant array) to determine if all of the data has been utilized and/or which indices are incremented for the next iteration. An example of this is given in Section 3.1.7.4.

3.1.7.3 Array Indexing

Arrays may be used in their entirety in HAL/S without explicit subscripting (for example assignment of two equally dimensioned arrays). However, the code generated is very similar to that for non-linear indexing, except that the indices are tested against the size of the corresponding declared dimensions of the arrays, rather than against the size of the corresponding dimensions of the subscripted array. An example of this is shown in the next section.

3.1.7.4 Arrayness and Loop Generation

This section has an example of each possible form of array loops, and how indexing is achieved within them. In general, an array loop consists of the following sections:

a. initialization of index values;
b. computation of address of array element from index value (see Section 3.1.3.3);
c. actual operation to be performed on the array element(s) (i.e. assignment, comparison, etc.);
d. incrementing and testing index values.

It should be noted that non-linear and array indexing produce multiple loops and indices. Since only a single register is available for indexing, temporary storage of index values for outer loops is employed.

Operation	Type	Code	
Linear Indexing:		L 7,=H'1,2'	}(1)
[X]=[Y]$_{3\ AT\ 2}$	[X]: ARRAY(3) SCALAR loop:	LED 2,Y+4(7)	}(2)
	[Y]: ARRAY(5) SCALAR DOUBLE	STE 2,X(7)	
		BIX 7,loop	}(3)

Notes on above example:

1. initialize
2. assignment
3. increment and test index

```
Operation      Type                      Code
Non-Linear Indexing:
[I]=[V]1,2 TO 3,*:2
     [I]:  ARRAY(2,4) INTEGER
     [V]:  ARRAY(2,3,4)VECTOR  L        7,=H'1,1'       } (1)
                  outer-loop:  ST       7,temp1

                               L        7,=H'1,3'       }(2)
                  inner-loop:  LH       6,templ         }
                               SLL      6,2             }
                               AR       6,7             }(3)
                               MHI      6,H'3'          }
                               SLL      6,15            }
                               LH       5,templ         }
                               SLL      5,2             }(4)
                               AR       5,7             }
                               LE       0,V+100(6)      }
                               STH      5,temp2         }
                               ACALL    ETOH            }(5)
                               LH       6,temp2         }
                               STH      5,I(6)          }
                               BIX      7,inner-loop    }(6)
                               L        7,templ         }(7)
                               BIX      7,outer-loop    }
```

Notes on above example:

1. initialization and storage of first index value
2. initialization of second index value
3. indexing of [V]
4. indexing of [I]
5. assignment of scalar value to an integer value
6. incrementing and testing second index value
7. incrementing and testing first index value

```
Operation    Type            Code
 *   *        *
[M]=[N]     [M],[N]:                L     RM,const1
            ARRAY(2,3)              L     RN,const2
            MATRIX(2,4)             MVH   RM,RN
                                     .
                                     .
                                     .
                            const1  DC    Y(M)
                                    DC    H`96'
                            const2  DC    Z(N)
```

Note: 96 is the size of N, (i.e. N is 2x3x2x4x2 = 96 halfwords long).

3.1.8 PROCEDURE/FUNCTION Calls

The PROCEDURE/FUNCTION calling process consists of two parts:

a. argument set up; and
b. the actual branch to the subroutine.

Argument set up uses registers 5-7 as needed for passing integers or bit strings, and/or pointers to vectors, matrices, character strings, arrays or structures. Floating point registers 0, 2, and 4 are similarly used to pass scalar arguments. Once all of these registers are utilized, all remaining arguments are placed in a run time stack for the procedure or function.

The actual code generated sets up the arguments in the order that they appear in the HAL/S PROCEDURE or FUNCTION block definition statement. For example, if the function is:

```
F: FUNCTION(integer_1,scalar_1, scalar_2,vector_1,integer_2);
```

then the registers are loaded in the order:

```
register 7  using LH or L
register 6  using LA to load the pointer to vector_1
register F2 using LE or LED
register F0 using LE or LED depending on the precision of scalar_1
register 5  using LH or L depending on the precision of integer_1
```

Once all arguments are set up, the actual branch is a BAL or SCAL instruction to the CSECT defined for the procedure or function.

A leaf PROCEDURE/FUNCTION is one which has no stack requirements (i.e. no parameters, no stack temporaries, no local addressable data, no ON ERROR statements, and no intrinsic library calls). Such procedures may be called via BAL R4, <routine name>. These routines are exited using BCR 7,R4.

Operation	Args	Code	Alternate Code
Argument Setup	≤3 non-scalar	LH 7,arg3	L 7,arg3 or LA 7,arg3
	and ≤3 scalar	LH 6,arg2	L 6,arg2 or LA 6,arg2
		LH 5,arg1	L 5,arg1 or LA 5,arg1
		LE 4,scalar-arg3	LED 4,scalar-arg3
		LE 2,scalar-arg2	LED 2,scalar-arg2
		LE 0,scalar-arg1	LED 0,scalar-arg1

```
Actual Call                    ACALL csect-name
```

Operation	Args	Code		Alternate Code
Argument	>3 non-scalar	LH	R,argn	
Setup	and/or >3 scalar	STH	R,stack	
		.		
		.		1) non-scalar
		.		stores stack
		LH	R,arg4	
		STH	R,stack	
		LE	FR,scalar-argn	
		STE	FR,stack	
		.		2) scalar
		.		stores
		.		into stack
		LE	FR,scalar-arg4	
		STE	FR,stack	
		LH	5,arg1	
		.		
		.		3)
		.		
		LE	2,scalar-arg2	
Actual Call		ACALL	csect-name	

Notes on above example:

1., 2. Any additional arguments are generally loaded into any unused register and stored. The actual load op codes may be: L, LH, LA, LE, or LED, depending on the type of argument. Similarly, the stores op codes may be ST, STH, STE, or STED. If the argument already exists in a register, then the code generated will be only a store from that register into the stack.

3. Loading of the first 3 non-scalar, and the first 3 scalar arguments. This is identical to the code shown in the first example above.

3.1.9 Block Definition

3.1.9.1 PROGRAM and TASK Definition

Operation		Code	
PROGRAM or TASK definition	block-name:	LA	0,stack-start†
		LA	1,program-data-csect
		NHI	1,x'7FFF'††
		STH	1,5(0)
		IAL	0,stack-size
		LA	3,local-data-area(1)
		LDM	$ZDSESET††
		STH	3,9(0)

† Omitted if SDL option is turned on.

†† Only occurs if the DATA_REMOTE directive is in effect (See Section 4.0).

3.1.9.2 PROCEDURE and FUNCTION Definition

Both PROCEDURE and FUNCTION definitions are similar to PROGRAM and TASK definitions. However, floating point store instructions are needed to save any scalar argument passed via registers.

Operation	Code†	[Alternate Code]
PROCEDURE or FUNCTION definition		
		[COMSUB only: LA 1,Program-data-csect]
block-name:	NHI 1,x'7FFF'††	
	STH 1,5(0)	
	IAL 0,stack-size	
	LA 3,Local-data-area(1)	
		[COMSUB only: LDM $ZDSESET††]
	STH 3,9(0)	
		[EXCLUSIVE only: SVC Local-dataarea+2(1)]
	STE 0,stack	[STED 0,stack]
	STE 2,stack	[STED 2,stack]
	STE 4,stack	[STED 4,stack]

† If PROCEDURE or FUNCTION is leaf (has no calls and no stack requirements) then no code is generated for the block entry.

†† Only occurs if the DATA_REMOTE directive is in effect (See Section 4.0).

3.1.10 Flow of Control Statements

3.1.10.1 IF … THEN … ELSE

The code shown below is for the most general form of the IF…THEN…ELSE statement. It is assumed that the condition code from the conditional expression has been generated (see previous subsections on conditional operations).

```
Operation                         Code
IF<cond exp>THEN<…>ELSE <…> BC      cond, else-label
                            ⎧executable code for THEN clause ⎫
                            ⎨                                ⎬
                            ⎪                                ⎪
                            ⎩                                ⎭
                            BC 7,next-statement
                            ⎧executable code for ELSE clause ⎫
              else-label:   ⎨              .                 ⎬
                            ⎪              .                 ⎪
                            ⎩              .                 ⎭

           next-statement:
                                            .
                                            .
                                            .
IF<cond exp>THEN<…>         BC  cond, next-statement
                            ⎧executable code for THEN clause ⎫
                            ⎨              .                 ⎬
                            ⎪              .                 ⎪
                            ⎩              .                 ⎭
         next-statement:
                                           .
                                           .
                                           .
```

3.1.10.2 DO FOR…Loops

The DO FOR loop has two forms: the iterative, and the discrete. They may also cause termination of the loop by use of the clause UNTIL < >, or WHILE < >. The use of these clauses is shown for the case of the iterative DO FOR forms where the additional code needed has been labeled "UNTIL code" and "WHILE code". This same additional code is generated for the discrete DO FOR and is placed immediately before the executable code within the DO group (the same process as is illustrated with the iterative DO FOR). Note that the code only shows the use of a single precision integer index; double precision integers, and single or double precision scalars follow the same algorithm with the exception that the corresponding full word, or floating point instructions are used when dealing with the index variable.

```
Operation                                     Code
DO FOR I = a TO b BY c;†                      LHI   7,a
                                 loop-begin:  BC    7,test-label
                                               .
                                               .    executable code within DO group
                                               .
                      repeat††:               LH    7,I†††
                                              AHI   7,c
                                 test-label:  STH   7,I
                                              CHI   7,b
                                              BC    6,loop-begin
                                 exit-label:   .
                                               .    code for statement
                                               .    following DO group

DO FOR I = a TO b BY c                        ZH    temp-area   UNTIL code
       UNTIL <cond exp>;
    .                                         LHI   7,a
    .
    .                                         BC    7,test-label
END;                             loop-begin:  TS    temp-area
                                              BC  4,first-statement††††  ⎫
                                               .  cond for exp           ⎪
                                               .                         ⎬UNTIL code
                                               .                         ⎪
                                              BC  cond,exit-label        ⎭
                            first-statement:   .
                                               .    executable code within DO group
                                               .
                                   repeat††:  LH    7,I
                                              AHI   7,c
                                 test-label:  STH   7,I
                                              CHI   7,b
                                              BC    6,loop-begin
                                 exit-label:   .    code for statement
                                               .    following DO group
                                               .
```

† Assumes a, b, and c are literal values.

†† This is referenced by the REPEAT STATEMENT (see Section 3.1.10.5).

††† This instruction may be omitted if the REPEAT label is not actually used, and the loop index I is already in the designated register.

†††† This is done to avoid testing the <cond exp> until after executing through the loop at least once.

```
Operation                                       Code
DO FOR I = a TO b BY c
       WHILE <cond exp>
    .                                           LHI   7,a
    .                                           BC    7,test-label
    .                            loop-begin:    .
END;                                            .     code for cond exp
                                                .                        } WHILE code
                                                BC    cond, exit-label
                                                .
                                                .     executable code within DO group
                                                .
                                                LH    7,I
                                                AHI   7,c
                                test-label:     STH   7,I
                                                CHI   7,b
                                                BC    6,loop-begin
                                exit-label:     .     code for statement
                                                .     following DO group
                                                .
DO FOR I = a1,a2,…,an              label-1:     LHI   7,a1
    .
    .                                           BAL   4,test-label
    .
END;                               label-2:     LHI   7,a2
                                                BAL   4,test-label
                                                .
                                                .
                                                .
                                   label-n:     LHI   7,an
                                                LA    4,exit-label
                                test-label:     ST    4,temp-area
                                                STH   7,I
                                                .
                                                .     executable code within DO group
                                                .
                             repeat†:           L     4,temp-area
                                                BCR   7,4
                                exit-label:     .     code for statement
                                                .     following DO group
                                                .
DO FOR I = I1 TO I2 BY I3                       LH    5,I2
    .
    .                                           STH   5,temp-test
    .
END;                                            LH    6,I3
```

† This is referenced by the REPEAT statement (see Section 3.1.10.5).

```
Operation                                Code
                                         LH    7,I1
(I1,I2,I3: variables)                    STH   6,temp-incr
                                         BC    7,test-label
                           loop-begin:    .
                                          .    executable code within DO group
                                          .
                             repeat†:    LH    7,I
                                         AH    7,temp-incr
                          test-label:    STH   7,I
                                         LH    5,temp-incr
                                         LA    5,loop-begin
                                         BC    5,positive-test††
                                         CH    7,temp-test
                                         BCR   5,5
                                         BC    7,exit-label
                       positive-test:    CH    7,temp-test
                                         BCR   6,5
                          exit-label:     .    code for statement
                                          .    following DO group
                                          .
```

† This is referenced by the REPEAT statement (see Section 3.1.10.5).

†† This branch is determined by the condition code set by the previous LH 5, temp-incr instruction.

3.1.10.3 DO WHILE/UNTIL

Both of these forms of DO groups are essentially the same except that the DO UNTIL does not test its conditional expression until it has finished executing the code once. In both cases, the condition is tested as detailed in preceding subsections.

```
Operation           Code
DO WHILE <cond exp>
                  repeat:       .
                                . code for conditional expression
                                .
                        BC     cond,exit-label
                                .
                                . code for statements within DO group
                                .
                        BC     7,repeat
              exit-label:       .
                                . code for statement following DO group
                                .
```

<u>Operation</u> <u>Code</u>

```
DO UNTIL <cond exp>         BC    7,first-statement
                   repeat:        .
                                  . code for conditional expression
                                  .
                            BC    cond,exit-label
          first-statement:        .
                                  . code for statements within DO group
                                  .
                            BC    7,repeat
               exit-label:        .
                                  . code for statement following DO group
                                  .
```

3.1.10.4 DO CASE

The DO CASE statement is used to select one of a collection of statements for processing.

<u>Operation</u> <u>Code</u>

```
DO CASE I;          LH       Rc,I
  <statement 1>     BC       6,else-case-label†
  <statement 2>     LA       2,case-vector
   .                CH       Rc,0(2)†
   .                BC       1,else-case-label†
   .                LH       2,0(Rc,2)
  <statement n>     BCR      7,2
END;
```

```
┌code above is replaced by the following if the
│DATA_REMOTE directive is in effect (see Section 4.0)
│LH   Rc,I
│BC   6,else-case label†
│LA   3,case-vector
│CH   Rc,0(3)†
│IHL  3,9(0)†
│SLL  3,16†
│BC   1,else-case-label†
│LA   1,case-vector
│LH   2,0(Rc,1)
│LH   1,5(0)
└BCR  7,2
```

† Bounds checks on case number. Omitted if ELSE not specified.

```
Operation           Code
else-case-label:
                    <else statement code>
                    BC 7,exit-case-label
                    <statement 1>
                    BC 7,exit-case-label
                    <statement 2>
                    BC 7,exit-case-label
                      .
                      .
                      .
                    <statement n>
exit-case-label:
                    Data
    case-vector     DC     H'n'
                    DC     Y(statement 1)
                    DC     Y(statement 2)
                      .
                      .
                      .
                    DC     Y(statement n)
```

3.1.10.5 GO TO, REPEAT, EXIT

All of these statements take the form of unconditional branches. It should be noted that REPEAT and EXIT statements may only be used inside DO groups. See Sections 3.1.10.2 and 3.1.10.3 for the locations of the "repeat" and "exit-label" within a DO group.

Operation	Code		
GO TO label	BC	7,label	
REPEAT REPEAT label	BC	7,repeat	"repeat" is the location of the code which determines whether DO group iteration is finished or not.
EXIT EXIT label	BC	7,exit-label	"exit-label" is the location of the code immediately following the end of the DO group.

3.1.10.6 RETURN

The RETURN statement will branch back from the code for a function to the code immediately following the function's invocation.

```
Operation            Code
Procedures & Functions
RETURN               EXCLUSIVE only:   SVC    Local-data-area+3(1)
                     normal:           LDM    $ZDSECLR†
                                       SRET   7,0
                     leaf:             BCR    7,4
Programs & Tasks
```

† Only occurs if the DATA_REMOTE directive is in effect (see Section 4.0) and the procedure/function is a COMSUB.

3.1.10.7 ON ERROR/OFF ERROR/SEND ERROR

```
Operation                                 Code
ON ERRORn:m<stmt>                         LA   4,<stmt>
                                          STH  4,error table entry 1
                                          LHI  4,<action>†
                                          STH  4,error table entry 0
                                          BC   7,next-statement
                   <stmt>:                     <code for stmt>
                   next-statement:         .
                                           .   code for next statement
                                           .
                   SIGNAL
ON ERRORn:mSYSTEM[AND SET <event>]
                   RESET
                                          LA   4,<event>†             only if event
                                          STH  4,error table entry 1  action phase
                                                                      present
                                          LHI  4,<action>
                                          STH  4,error table entry 0
                   SIGNAL
ON ERRORn:mIGNORE[AND SET <event>]
                   RESET
                                          LA   4,<event>              only if event
                                          STH  4,error table entry 1  action phase
                                                                      present
                                          LHI  4,<action>†
                                          STH  4,error table entry 0
SEND ERRORn:m                             SVC  =X'0014nnmm'
OFF ERRORn:m                              ZH   4,error table entry 0
```

† <action> contains action code, error code, and the error group as defined in HAL/FCOS ICD.

3.1.11 Built-In Functions

3.1.11.1 Inline Built-in Functions

The following built-in functions emit the inline code shown in the following sequences. In all cases, it is assumed that Rx contains the argument except when a specific load instruction is shown. The results will always be in register Ry.

Operation	Type	Code	
ABS(arg)	scalar, single	LE	Ry,arg
		LECR	Ry,Ry
		BC	2,*-1
	scalar, double	LED	Ry,arg
		LECR	Ry,Ry
		BC	2,*-1
	integer, single	LH	Ry,arg
		LACR	Ry,Ry
		BC	2,*-1
	integer, double	L	Ry,arg
		LACR	Ry,Ry
		BC	2,*-1
LENGTH (char)	character string	LH	Ry,char
		NHI	Ry,255
SIGN (arg)	scalar, single	LE	Rx,arg
		LFLI	Ry,1
		LER	Rx,Rx
		BC	5,continue
		LECR	Ry,Ry
			.
	continue:		.
			.
	scalar, double	LED	Rx,arg
		LED	Ry,D'4110000000000000'
		LEDR	Rx,Rx
		BC	5,continue
		LECR	Ry,Ry
			.
	continue:		.
			.
	integer, single	LH	Rx,arg
		LFXI	Ry,1
		LR	Rx,Rx
		BC	5,continue

Operation	Type	Code	
		LACR	Ry,Ry
			.
	continue:		.
			.
	integer, double	L	Rx,arg
		L	Ry,=F'1'
		LR	Rx,Rx
		BC	5,continue
		LACR	Ry,Ry
			.
	continue:		.
			.
SIGNUM(arg)	scalar, single	LE	Rx,arg
		LFLI	Ry,1
		LER	Rx,Rx
		BC	1,continue
		BC	4,equal
		LECR	Ry,Ry
		BC	7,continue
	equal:	SER	Ry,Ry
			.
	continue:		.
			.
	integer, single	LH	Rx,arg
		LFXI	Ry,1
		LR	Rx,Rx
		BC	1,continue
		BC	4,equal
		LACR	Ry,Ry
		BC	7,continue
	equal:	SR	Ry,Ry
			.
	continue:		.
			.
	integer, double	L	Rx,arg
		L	Ry,=F'1'
		LR	Rx,Rx
		BC	1,continue
		BC	4,equal
		LACR	Ry,Ry
		BC	7,continue

Operation	Type	Code	
	equal:	SR	Ry,Ry
			.
	continue:		.
			.
SUBBIT$_{m\ TO\ n}$(arg)	integer, single or bits of length ≤16	SRL	Ry,16-n
		NHI	Ry,mask†
-or-			
SUBBIT$_{n-m+1\ AT\ m}$(arg)			
	integer double, or bits of length >16, or scalar single	SRL	Ry,32-n
		N	Ry,F'mask'†
SHL(x,n)	integer	SLL	Rx,n
SHR(x,n)	integer	SRA	Rx,n
XOR(X,Y)	Bit,n≤16	LH	Ry,Y
		XR	Rx,Ry
	Bit,n>16	X	Rx,Y
		or XR	Rx,Ry
MIDVAL(X,Y,Z)	scalar	LE	F0,X
		LE	F1,Y
		MVS	F0,Z

† The mask value is: $2^{(n-m+1)}-1$.

3.1.11.2 Out of Line Functions

Out of line functions require branches to the runtime library.

The registers needed for parameter passing, and the name of the library routine branched to, are specified in the tables of Section 5. Examples are given for representative argument types.

Operation	Type of X	Code	
COS(X)	scalar,single	LE	0,X
		ACALL	COS
SQRT(X)	scalar,double	LED	0,X
		ACALL	DSQRT
ABVAL(X)	vector(3),double	LA	2,X
		ACALL	VV9D3
TRANSPOSE(X)	matrix(m,n),double	LA	P2,X
		LA	P1,temp-area

Operation	Type of X	Code	
		LA	5,n
		LA	6,m
		ACALL	MM11DN
	matrix(3,3),single	LA	P2,X
		LA	P1,temp-area
		ACALL	MM11S3
UNIT(X)	vector(3),single	LA	P2,X
		LA	P1,temp-area
		ACALL	VV10S3
RANDOMG		ACALL	RANDG
TRIM(X)	character	LA	P2,X
		LA	P1,temp-area
		ACALL	CTRIMV
MAX(X)	array(n)	LA	2,X
		LHI	5,n
		ACALL	EMAX

3.1.11.3 Shaping Functions

Shaping functions are explicit invocations of type conversion. The generated code for shaping functions has been described in previous subsections where conversions have been described (see Sections 3.1.2.3, 3.1.3.4, 3.1.4.4, and 3.1.5.4).

In addition, when conversion functions are used in a true "shaping" sense, (e.g. MATRIX(<integer array>)), a subroutine is used to move contiguous elements, with possible conversion, to a result location of the desired shape.Example:

MATRIX(A) where A is a 9 element integer array

```
LA        P2,A1
LA        P1,<result loc>
LHI       6,X'0002'flags[5] size
LHI       5,9
ACALL     QSHAPQ
```

5. Flags: 1st 8 bits indicate input data type.
2nd 8 bits indicate output data type.

```
Values:  0 = H
         1 = I
         2 = E
         3 = D
```

3.1.12 Real Time Statements

All REAL TIME statements are implemented by means of a supervisor call (SVC) instruction which has as its address a pointer to a parameter list. The first halfword of this parameter list contains a number which identifies the type of real time call. The remainder of the parameter list varies with the service being requested.

The specific forms of the SVC parameter lists are those described in the *HAL/FCOS ICD*.

For real time statements in non-REENTRANT blocks, the SVC parameter lists are in the block's data area. Any invariant portions of the parameter lists are implemented by initialized data. Parts of the parameter lists which are runtime-dependent are created by execution of in-line code preceding the SVC instruction.

For real time statements in REENTRANT blocks, the SVC parameter lists are dynamically created in the stack by executable code preceding the SVC instruction.

3.1.12.1 WAIT Statement

The WAIT statement may use registers 0, 1 to contain a double precision time value specified in seconds. If the UNTIL option is specified, the time value is expressed as mission elapsed time. Any other times are "Delta Time" from the current mission elapsed time. If a time value is not specified in the WAIT statement, then the registers will not be affected.

Operation	Type	Code	
WAIT n	n: literal	LED	0,D'floating point form of n'
		SVC	parameter-list
WAIT X	X: scalar double	LED	0,X
		SVC	parameter-list
WAIT FOR DEPENDENT		SVC	parameter-list
WAIT FOR X	X: event value	SVC	parameter-list
WAIT UNTIL X	X: scalar double	LED	0,X
		SVC	parameter-list

3.1.12.2 CANCEL, TERMINATE Statements

Operation	Code
CANCEL	
CANCEL <task id>	
TERMINATE	SVC parameter-list
TERMINATE <task id>	

3.1.12.3 SIGNAL, SET, RESET Statements

```
Operation            Type                        Code
SIGNAL<event var>    latched or unlatched  ⎫
SET<event var>       latched               ⎬   SVC parameter-list
RESET<event var>     latched               ⎭
```

3.1.12.4 UPDATE PRIORITY Statement

```
Operation                          Type               Code
UPDATE PRIORITY TO i              ⎫
UPDATE PRIORITY<taskid> TO i      ⎬  i: integer   SVC parameter-list
                                  ⎭
```

3.1.12.5 SCHEDULE Statement

In the following code generation sequences, a schematic representation of possible SCHEDULE statement forms has been used. The symbol [ ] means that one of the contained elements may appear in the statement form without affecting the generated code. The symbol { } means that one of the contained elements <u>must</u> be included in the statement form - but which one does not affect the code generated.

In general, the code differs only when time values are specified in the SCHEDULE statement. This requires that the time values be specified in double precision format in certain registers as shown below.

```
Operation                                                                  Code
SCHEDULE<label>[ON<eventexp>]PRIORITY(I)[DEPENDENT] ⎡WHILE<event exp>⎤     SVC  parameter-list
                                                    ⎣UNTIL<event exp>⎦

                         ⎧      ⎫                                          LED  0,D'X'
SCHEDULE <label>⎨ AT X ⎬  PRIORITY(I)[DEPENDENT] ⎡WHILE<event exp>⎤
                         ⎩ IN X ⎭                  ⎣UNTIL<event exp>⎦
                                                                           SVC  parameter-list
                                                              ⎧         ⎫
SCHEDULE<label>[ON<eventexp>]PRIORITY(I)[DEPENDENT],REPEAT⎨AFTER X ⎬     LED  2,D'X'
                                                              ⎩EVERY X ⎭
                                                                           SVC  parameter-list
SCHEDULE<label>[ON<eventexp>]PRIORITY(I)[DEPENDENT]UNTIL X                 LED  4,D'X'
                                                                           SVC  parameter-list

                 ⎧      ⎫                          ⎧         ⎫             LED  0,D'X'
SCHEDULE<label>⎨ AT X ⎬PRIORITY(I)[DEPENDENT],REPEAT ⎨AFTER y ⎬ ⎡WHILE<event exp>⎤  LED  2,D'Y'
                 ⎩ IN X ⎭                          ⎩EVERY y ⎭ ⎣UNTIL<event exp>⎦  SVC  parameter-list

                 ⎧      ⎫                                                  LED  0,D'X'
SCHEDULE<label>⎨ AT X ⎬PRIORITY(I)[DEPENDENT] UNTIL Y                      LED  4,D'Y'
                 ⎩ IN X ⎭                                                  SVC  parameter-list

                                                              ⎧        ⎫
SCHEDULE<label>[ON<eventexp>]PRIORITY(I)[DEPENDENT],REPEAT⎨AFTER X⎬ UNTIL Y   LED  2,D'X'
                                                              ⎩EVERY X⎭       LED  4,D'Y'
```

Operation	Code
	SVC parameter-list
SCHEDULE<label>{AT X / IN X}PRIORITY(I)[DEPENDENT], REPEAT {AFTER y / EVERY y} UNTIL Z	LED 0,D'X' LED 2,D'Y' LED 4,D'Z' SVC parameter-list

3.1.13 I/O Statements

The READ statement will compile successfully, but will generate incorrect results for BFS and produce an error in the linkage editor for PASS. The error is not generated for BFS because it uses a different linkage editor. The user will see the following message in the map file for PASS:

```
  IEW0264 - TABLE OVERFLOW - INPUT LOAD MODULE CONTAINS TOO MANY
EXTERNAL SYMBOLS IN ESD
```

This is accepted as a known error due to the fact that neither the BFS nor PASS flight software use either the READ or READALL statements. (DR 102959, 10/22/90)

3.1.13.1 Initiation

Initiation of either READ, READALL, or WRITE statements consists of a branch to the IOINIT library routine. Register 1 contains the I/O channel number and register 0 indicates the type of I/O to be initiated.

Operation	Type	Code
READ(n)…		LHI 6,n LHI 5,0 ACALL IOINIT
READALL(n)…		LHI 6,n LHI 5,1 ACALL IOINIT
WRITE(n)…		LHI 6,n LHI 5,3 ACALL IOINIT

3.1.13.2 Input

In all cases, the code sequences below follow the I/O initiation process described in the previous subsection. It is assumed that any conversions have been done previous to the code sequences shown; the resultant type determines which type of code sequence is generated. Note that vector and matrix partitioning require that the first element of the partition be known; additionally, matrices require a DELTA value to be known to skip over those elements (in the "natural sequence") which are not part of the resulting partitioned Matrix (see Section 3.1.2.3).

```
Operation         Type                   Code
READ()…,I,…       integer, single        .
                                         .      initiation
                                         .
                                         LA     2,I
                                         ACALL  HIN
                                         .
                                         .      initiation
                  integer, double        .
                                         LA     2,I
                                         ACALL  IIN
                                         .
                                         .      initiation
READ(),…,S,111    scalar, single         .
                                         LA     2,S
                                         ACALL  EIN
                                         .
                                         .      initiation
                  scalar, double         .
                                         LA     2,S
                                         ACALL  DIN
READ()…,V,…       vector(n);single       .
                                         .      initiation
                                         .
                                         LA     2,V
                                         XR     7,7
                                         LHI    5,1
                                         LHI    6,n
                                         ACALL  MMRSNP
READ()…,V,…       partitioned vector     .
                  of length n whose      .      initiation
                  first element is       .
                  located at 'V+         LA     2,V+displacement
                  displacement'          XR     7,7
                                         LHI    5,1
                                         LHI    6,n
                                         ACALL  MMRSNP
                  vector(n); double      same except branches to
                  (partitioned or        MMRDNP
                  not partitioned)
```

Operation	Type	Code	
READ()…,M,…	matrix(m,n); single	.	
		.	initiation
		.	
		LA	2,M
		XR	7,7
		LHI	5,m
		LHI	6,n
		ACALL	MMRSNP
READ()…,M,…	partitioned matrix	.	
	whose resultant	.	initiation
	size is mxn, first	.	
	element is M + displacement.	LA	2,M+displacement
		LHI	7,DELTA
		LHI	5,m
		LHI	6,n
		ACALL	MMRSNP
	matrix(m,n); double	Same except branches to MMRSNP	
	(partitioned or not partitioned)		
READ()…,C,…	character string	.	
or READALL()…,C,…		.	initiation
		.	
		LA	2,C
		ACALL	CIN
READ()…,	partitioned	.	
$C_{m\ TO\ n}$,…	character string	.	initiation
or READALL()…,		.	
$C_{m\ TO\ n}$,…		LA	2,C
		LHI	5,m
		LHI	6,n
		ACALL	CINP
READ()…,C_n,…	single partitioned	.	
or READALL()…,	character string	.	initiation
C_n,…		.	
		LA	2,C
		LHI	5,n
		LR	6,5
		ACALL	CINP

```
Operation        Type                Code
READ()…,B,…      bit string(of       .
                 length n)           .      initiation
                                     .
                                     LHI    6,n
                                     ACALL  BIN†
                                     BC     4,around
                                     ST     6,B
                           around:   .
                                     .
                                     .
```

Arrayed Input — The actual code generated depends on the type of array. Thus, the code will consist of an array loop (see Section 3.1.7.4) which contains the proper code for inputting of each array element using the code shown above (corresponding to the array element type).

† BIN returns the bit string input in register R6.

3.1.13.3 Output

In all cases, the code sequences below follow the I/O initiation processes described in Section 3.1.13.1. It is assumed that any conversions have been done previous to the code sequences shown; the resultant type determines which type of code sequence is generated. Note that vector and matrix partitioning require that the first element of the partition be known; additionally, matrices require a "delta" value be known to skip over those elements (in the "natural sequence") which are not part of the resulting partitioned matrix.

```
Operation        Type                Code
WRITE()…,I,…     integer, single     .
                                     .      initiation
                                     .
                                     LH     5,I
                                     ACALL HOUT
                 integer, double     .
                                     .      initiation
                                     .
                                     L      5,I
                                     ACALL IOUT
WRITE()…,S,…     scalar, single      .
                                     .      initiation
                                     .
                                     LE     0,S
                                     ACALL EOUT
```

Operation	Type	Code
	scalar, double	LED 0,S ACALL DOUT
WRITE()…,V,…	vector(n); single	. . initiation . LA 2,V XR 7,7 LHI 5,1 LHI 6,n ACALL MMWSNP
WRITE()…,V,…	partitioned vector of length n whose first element is located at 'V+displacement'	. . initiation . LA 2,V+displacement XR 7,7 LHI 5,1 LHI 6,n ACALL MMWSNP
	vector(n); double (partitioned or non-partitioned)	same except branches to MMWDNP
WRITE()…,M,…	matrix(m,n); single	. . initiation . LA 2,M XR 7,7 LHI 5,m LHI 6,n ACALL MMWSNP
WRITE()…,M,…	partitioned matrix of resultant size mxn whose first element is M +	. . initiation . LHI 7,delta LHI 5,m LHI 6,n ACALL MMWSNP

Operation	Type	Code
	matrix(m,n);double (partitioned or not partitioned)	same except branches to MMWDNP
WRITE()…,C,…	character string	. . initiation . LA 2,C ACALL COUT
WRITE()…,$C_{m\ TO\ n}$	partitioned character string	. . initiation . LA 2,C LHI 5,m LR 6,n ACALL COUTP
WRITE()…,C_n,…	single partitioned character string	. . initiation . LA 2,C LHI 5,m LR 6,5 ACALL COUTP
WRITE()…,B,…	bit string (of length n)	. . initiation . L 5,B LHI 6,n ACALL BOUT
Arrayed Output	The actual code generated depends on the type of array. Thus, the code will consist of an array loop (see Section 3.1.7.4) to cause iterative outputting of each array element using the code shown above (corresponding to the array element type).	

3.1.14 NAME Operations

3.1.14.1 NAME Comparisons

NAME comparisons may only be = or ¬ =.

```
Operation                 Code
NAME(X) <OP> NAME(Y)     X and Y are NAME variables
                            [LH  Rx,X                  ]
                            [LH  Ry,Y                  ]
                            -or-
                         X and Y are NAME REMOTE variables
                            [L   Rx,X                  ]
                            [L   Ry,Y                  ]
                         the ZCON index inhibit bits are
                         ignored for the comparison
                            [LHI Ri,x‘FFFF0000’        ]
                            [IAL Ri,x‘F7FF’            ]
                            [NR  Rx,Ri                 ]
                            [NR  Ry,Ri                 ]
                            -or-
                         X is a declared variable, Y is a NAME
                         variable
                            [LA  Rx,X                  ]
                            [LH  Ry,Y                  ]
                            -or-
                         X is a declared REMOTE variable,
                         Y is a NAME REMOTE variable
                            [L   Rx,ZCON(X)            ]
                            [L   Ry,Y                  ]
                         the ZCON index inhibit bits are
                         ignored for the comparison
                            [LHI Ri,x‘FFFF0000’        ]
                            [IAL Ri,x‘F7FF’            ]
                            [NR  Rx,Ri                 ]
                            [NR  Ry,Ri                 ]
                            -or-
                         X is a declared local variable, Y is
                         a NAME REMOTE
                            [LA  Rx,X                  ]
                            [OHI Rx,x'8000'            ](PASS only)
                            [IAL Rx,x'0800' or x'0000']
                            [L   Ry,Y                  ]
```

```
Operation              Code
                         the ZCON index inhibit bits are
                         ignored for the comparison
                            [LHI Ri,x'FFFF0000'       ]
                            [IAL Ri,x'F7FF'           ]
                            [NR  Rx,Ri                ]
                            [NR  Ry,Ri                ]
                            -or-
                         X is a NAME variable, Y is a NAME REMOTE
                            [L   Ry,Y                 ]
                            [LH  Rx,X                 ]
                            [IAL Rx,x'0800'           ]
                                 (non-aggregate variables only)
                            [SRA Rx,1                 ]
                            [SRR Rx,31                ]
                            [OHI Rx,X'8000'           ](PASS only)
                         the ZCON index inhibit bits are ignored
                         for the comparison
                            [LHI Ri,x'FFFF0000'       ]
                            [IAL Ri,x'F7FF'           ]
                            [NR  Rx,Ri                ]
                            [NR  Ry,Ri                ]
The following apply to all of the above examples:
                            CR   Rx,Ry
                            BC   COND, not-true-label
                            BC   7, true-label
Note that the compiler emits an RLD card that informs the linkage
editor to insert the proper CSECT value into the last four bits
inserted by the IAL instruction for non-NAME non-stack variables,
to conform to the ZCON format.
```

3.1.14.2 NAME Assignment

The variable Y in the following examples may only be a NAME variable. The variable X may be either an actual or NAME variable having declared properties identical to Y (refer to Language Specification).

```
Operation              Code
NAME(Y)=NAME(X)          X and Y are NAME variables
                            [LH  Rx,X                 ]
                            [STH Rx,Y                 ]
                            -or-
                         X and Y are NAME REMOTE variables
                            [L   Rx,X                 ]
                            [ST  Rx,Y                 ]
                            -or-
```

<u>Operation</u> <u>Code</u>

```
X is a declared local variable, Y is not
REMOTE
   [LA  Rx,X                ]
   [STH Rx,Y                ]
    -or-
X is a declared REMOTE variable, Y is a
NAME REMOTE
   [L   Rx,ZCON(X)          ]
   [ST  Rx,Y                ]
   -or-
X is a declared local variable, Y is a
NAME REMOTE
   [LA  Rx,X                ]
   [OHI Rx,x'8000'          ](PASS only)
   [IAL Rx,x'0800' or x'0000']
   [ST  Rx,Y                ]
   -or-
X is a NAME variable, Y is a NAME REMOTE
   [LH  Rx,X                ]
   [IAL Rx,x'0800'          ]
        (non-aggregate variables only)
   [SRA Rx,1                ]
   [SRR Rx,31               ]
   [OHI Rx,x'8000'          ](PASS only)
   [ST  Rx,Y                ]
```

Note that the compiler emits an RLD card that informs the linkage editor to insert the proper CSECT value into the last four bits inserted by the IAL instruction for non-NAME non-stack variables, to conform to the ZCON format.

-or-

<u>Operation</u> <u>Code</u>

```
X and Y are declared local variables, but
X was previously converted to a ZCON to
accomodate a NAME REMOTE left-hand side
variable in a multiple assignment
   [LA  Rx,X                    ]
   .
   .
   [YCON to ZCON conversion     ]
   .
   .
   [LR   Ry,Rx                  ]   ZCON
   [SLL  Ry,31                  ]   to YCON
   [OHI  Ry,x'7FFF'             ]   con-
   [NR   Rx,Ry                  ]   version
   [ST   Rx,Y                   ]
```

3.1.15 %MACROS

The following %MACROS are recognized by the HAL/S-FC compiler and produce the indicated code.

3.1.15.1 %SVC

The %SVC statement generates a true SVC instruction in which the address portion points to the operand specified.

Operation	Code
`%SVC (α)`	`SVC α`

3.1.15.2 %NAMECOPY

This operation works in the same manner as NAME assignments except that the operands must be structures, but not necessarily having identical properties. See Section 3.1.14.2 for more examples.

Operation	Code
`%NAMECOPY(Y,X);`	`LA Rx,X`
`where X is actual variable`	`STH Rx,Y`

3.1.15.3 %COPY

The %COPY statement is used to move data from one location to another without regard to data types. This operation uses the block move algorithm. See section 3.1.17 for examples of the variances of that algorithm. The general form is:
%COPY(dest, source, count);

where:

source is the variable name from which data will be moved;

dest is the variable name into which data will be moved; and

count is optional and if present indicates the number of halfwords to be moved from source to dest. If count is omitted, the size of the source operand is used to determine a count.

NAME variables are dereferenced in all cases. Use of a NAME variable as dest or source operand will refer to the data pointed to by the NAME variable. The count may also be a NAME variable in which case the count is taken from the storage location pointed to by the NAME variable.

Error checking for %COPY statements is performed when the third argument is a literal or omitted. The following errors may be emitted:

FN105: When the source or destination dereferences a Name variable.

FN106: When the element boundary is exceeded for non-remote locally declared variables. For example, if the source or destination is a scalar (size of scalar is two halfwords), and the number of halfwords to copy is 4, then the boundary of the variable has been exceeded and an FN106 is emitted. If the source or destination is an array, or element of an array, an FN106 error message is emitted when the location of the array element plus the number of halfwords to copy exceeds the ending position of the array.

FN107: When runtime addressing is generated for a non-remote locally declared variable involved in a %COPY statement. Runtime addressing means the compiler cannot determine the address of a variable during compilation.

FN108: When the source or destination dereferences a pass-by-reference formal parameter.

The move halfword instruction (MVH) is used to implement the %COPY statement when feasible. If possible, the compiler will optimize the %COPY sequence by performing LED, STED, L, ST or LH, STH sequences instead of using the MVH instruction. This optimization occurs when the compiler is able to determine that no more than eight halfwords need to be moved and that the data alignments match for both source and receiving data areas. When the destination operand of a %COPY statement is REMOTE, a call to MSTR is used to implement the move. Since MSTR expects ZCON inputs, the compiler will perform a YCON to ZCON conversion if the source operand is LOCAL data. The following examples show some of the possible sequences.

A %COPY Warning:

Since the %COPY macro moves data without regard to data type, it can be used to move one character string over another character string even if the character string sizes do not match. A HAL/S CHARACTER string is defined as one halfword descriptor, aligned on a halfword boundary, followed by the data (two characters packed into each halfword of memory). The string descriptor is organized into two one-byte values: the upper byte contains the maximum number of characters the string can hold and the lower byte contains the actual number of characters in the string.

When the %COPY source is a smaller string than the destination, the destination string's descriptor gets overwritten causing the string to internally become smaller and causing unexpected results in all subsequent uses of the destination. Conversely, when the source is bigger than the destination and the 3rd argument is a literal count or is omitted, the compiler correctly emits a FN106 error ("ELEMENT OR CSECT BOUNDARY EXCEEDED FOR DESTINATION OF %COPY") if the character strings are declared locally and are non-REMOTE.

Examples:

1. Strings A and B are declared as CHARACTER(20). When the following %COPY statement:

   %COPY(A,B);

is executed, 11 halfwords (10 halfwords (20 bytes of character data / 2) + 1 halfword of descriptor) are copied in to the memory location containing A. This %COPY statement does not pose any problems because the length of the source and destination strings are the same.

2. String X is declared as CHARACTER(20) and string Z is declared as CHARACTER(15). When the following:

   %COPY(X,Z);

   is executed, 9 halfwords (8 halfwords ((15 bytes of character data + 1 byte of pad) / 2) + 1 halfword of descriptor) are copied into the memory location containing CHARACTER X. The new maximum character size of string X would be 15. This occurs because the %COPY macro moves the 9 halfwords representing string Z over the first 9 halfwords of string A, and in the process, overwriting X's string descriptor. After this statement, and for the duration of the program's execution, the internal character size of X is 15, not 20, as originally declared! Furthermore, this situation could lead to some unexpected execution results. For example: after the above %COPY statement character B is assigned into character X ("X = B;") by calling the CASR run-time library, the character assignment would not take place. This happens because CASR's checks would detect the size of character X is internally smaller than string B.

3. Using the values in example 2 above, if the user codes the following %COPY statement:

   %COPY(Z,X);

   the compiler would correctly emit a FN106 error when the 3rd argument is omitted because the destination operand is smaller than the source operand.

Note: The compiler can allow an assignment to exceed the bounds of the receiving data. There are two methods of doing this that could cause problems, and thus should be avoided:

a. A %COPY statement with a variable halfword count field larger than the size of the destination when the destination is local data (this may also go beyond the bounds of a CSECT). The compiler performs bound checking when a literal count is provided but cannot perform these checks when a variable count is used. Local data can be rearranged whenever a compilation unit is modified.

b. Overindexing an arrayed variable (subscript is greater than the receiving data's declared arrayness).

The compiler assumes that arrayed assignments and %COPY are not used in this manner and does not update registers that may have been modified as a result of a violation of these rules.

Operation	Code	
%COPY(X,Y)	L	Rx,YCON(X, size of Y)
	L	Ry,ZCON(Y)
	MVH	Rx,Ry
%COPY(X,Y,n)	L	Rx,YCON(X,n)
	L	Ry,ZCON(Y)
	MVH	Rx,Ry
%COPY(X,Y,5)	L	Ry,Y
	ST	Ry,X
	L	Ry,Y+2
	ST	Ry,X+2
	LH	Ry,Y+4
	STH	Ry,X+4
%COPY(X,Y);	L	R4,ZCON(Y)
where X and Y are	LFXI	R5,size of Y
REMOTE	L	R2,ZCON(X)
	SCAL@#	MSTR
%COPY(X,Y);	LA	R4,Y
where X is REMOTE	OHI	R4,'8000' (PASS only)
and Y is local	IAL	R4,'0000'
	LFXI	R5,size of Y
	L	R2,ZCON(X)
	SCAL@#	MSTR

3.1.15.4 %NAMEADD

The %NAMEADD statement adds an integer to a given NAME variable and assigns the result into a separate NAME variable. It consists of three operands as seen below:

%NAMEADD(X, Y, Z)

where:

- X is any NAME variable.
- Y can be either any NAME variable or any HAL variable which is legal in the context NAME(V).
- Z is either an integer literal or variable which specifies the amount to be added (which may be negative) from the second operand. (Note: literals must not be signed.)

The source (Y) cannot be REMOTE if the destination (X) is non-REMOTE.

The result of the %NAMEADD statement which is placed into the NAME variable, X, is equal to the NAME value of Y plus the integer Z.

Operation	Code	
`%NAMEADD(X,Y,5);`	`LA`	`Rx,Y+5`
`where Y is a non-NAME variable`	`STH`	`Rx,X`
`and X is non-REMOTE`		
`%NAMEADD(X,Y,Z);`	`LA`	`Rx,Y`
`where Y is a non-NAME variable`	`AH`	`Rx,Z`
`and X is non-REMOTE`	`STH`	`Rx,X`
`%NAMEADD(X,Y,8);`	`LH`	`Rx,Y`
`where Y is a NAME variable`	`LA`	`Rx,8(Rx)`
`and X is non-REMOTE`	`STH`	`Rx,X`
`%NAMEADD(X,Y,8);`	`L`	`Rx,ZCON(Y)`
`where Y is a non-NAME REMOTE`	`AHI`	`Rx,8`
`variable and X is NAME REMOTE`	`ST`	`Rx,X`
`%NAMEADD(X,Y,8);`	`L`	`Rx,Y`
`where Y is NAME REMOTE`	`AHI`	`Rx,8`
`and X is NAME REMOTE`	`ST`	`Rx,X`

In either of the last two examples above, code may be inserted immediately before the ST instruction to clear the ZCON index inhibit bit according to the type of X.

Operation	Code		
`%NAMEADD(X,Y,8);`	`LA`	`Rx,Y+8`	
`where Y is a non-NAME variable`	`OHI`	`Rx,x'8000'`	`(PASS only)`

```
 and X is NAME REMOTE       IAL      Rx,x'0800' or x'0000'
                                     (linker fills in sector num-
                                     ber)

                            ST       Rx,X

%NAMEADD(X,Y,8);            LH       Rx,Y
 where Y is a NAME          LA       Rx,8(Rx)
variable
 and X is NAME REMOTE       STH      Rx,x'0800'
                                     (non-aggregate variables
                                     only)

                            SRA      Rx,1
                            SRR      Rx,31
                            OHI      Rx,x'8000'  (PASS only)
                            ST       Rx,X
```

If X is of aggregate type (i.e. ARRAY(3) SCALAR) and Y is of REMOTE singular type (i.e. SCALAR) then the following code is inserted to clear the index inhibit bit:

```
LFXI     Ri,-1
IAL      Ri,x"F7FF"
NR       Rx,Ri
```

3.1.15.5 %NAMEBIAS

By convention, the compiler uses an address for aggregate data that is offset a certain number of halfwords (depending on data type) before the actual beginning of the data (see Section 3.1.1.2). The %NAMEBIAS statement performs this zeroth element calculation by determining the positive value needed to point to the first element of the data. It consists of two operands as seen below:

%NAMEBIAS(X,Y)

where,

X	is a destination variable of integer type
Y	is a source variable of any data type (unsubscripted)

The result of the %NAMEBIAS statement is placed into the variable X. Please note that the offset of a NAME variable is the offset of the variable to which it points.

Operation	Code	
%NAMEBIAS(X,Y)	LFXI/LHI/L	Ry,zeroth element offset of Y
where X is a non-NAME variable	STH/ST	Ry,X

%NAMEBIAS(X,Y) where X is a NAME variable	LFXI/LHI/L LH STH/ST	Ry,zeroth element offset of Y Rx,X Ry,disp(Rx)
%NAMEBIAS(X,Y) where X is REMOTE	LFXI/LHI/L STH@#/ST@#	Ry,zeroth element offset of Y Ry,X
%NAMEBIAS(X,Y) where Y has zero offset	ZH/ZH@#	Rx

3.1.16 NONHAL References

Definition and use of the NONHAL construct in the HAL/S-FC compiler system results in an unimplemented feature message from the code generator.

3.1.17 Block Move Algorithm

For assignments involving arrays, structures, or vectors and matrices, the block move algorithm will be applied when:

1. both source and destination data specifications occupy contiguous storage;
2. the data types for source and destination match in type and precision; and
3. neither source nor destination variables are NAME variables.

The default block move sequence is as follows:

```
 X=Y;                          L    Rx,=Y(X,size of Y)
                               L    Ry,=Z(Y)
                               MVH  Rx,Ry

When X is declared REMOTE:
                               LA   P2,Y
                               OHI  P2,x'8000'  (PASS only)
                               IAL  P2,x'0800' or x'0000'
                               LFXI 5,size of Y
                               L    P1,ZCON(X)
                               ACALL MSTR

When both X and Y are declared REMOTE:
                               L    P2,ZCON(Y)
                               LFXI size of Y
                               L    P1,ZCON(X)
                               ACALL MSTR
```

```
When the DATA_REMOTE directive is in effect (see Section 4.0)
and the destination variable X is local data:
                              L    3,=Y(X, size of Y)
                              ZRB  3,x'8000'
                              L    Ry,=Z(Y)
                              MVH  3,Ry
                              LH   3,9(0)
```

In certain block move applications where the following is true:

a. variable indexing is not specified;
b. no NAME or ASSIGN parameters are specified;
c. source and data word alignments match and 8 or fewer halfwords are being moved, or two or fewer halfwords are being moved;

then the MVH sequence is replaced by the appropriate number of LED/STED (4 halfwords apiece), L/ST (2 halfwords apiece), and LH/STH instruction pairs to accomplish the movement.

The following examples illustrate:

```
DECLARE ARRAY (3),
        A,B,C INTEGER,
        D INTEGER;
DECLARE ARRAY (3) INTEGER,
              E,F;/*E,F on odd boundary*/
              A=B;
                       LED     0,B
                       STED    0,A
                       LE      0,B+4
                       STE     0,A+4
             C=D;
                       L       4,D
                       ST      4,C
                       LH      4,D+2
                       STH     4,C+2
             E=F;
                       LH      5,F
                       STH     5,E
                       L       5,F+1
                       ST      5,E+1
             B=E;
```

uses MVH since >2 halfwords and alignments not matching.

3.2 Object Code Naming Conventions

Each successful HAL/S compilation produces several named control sections (CSECTs). The CSECT names are derived according to the following rules:

a. HAL/S compilation unit names are transferred to the emitted object code by using only the first six characters of the HAL/S name. The name will be padded or truncated to six characters where necessary.

b. Any occurrence of the underscore character (_) in the first six characters of a PROGRAM, PROCEDURE, FUNCTION, TASK, or COMPOOL is eliminated. The resulting characters are joined together to produce the characteristic name of the compilation unit (e.g. A_B_C becomes ABC). Additional characters are placed on the front of the resultant name to form the final name for each of the individual situations in which the name is used. All CSECT names therefore take the form:

ccNNNNNN

where the value of cc for individual cases is:

```
PROGRAM                      :  $0
TASKs                        :  $c c=(1-9,A-Z)
COMSUBs                      :  #C
Internal procs               :  an a=(A-Z),n=(0-9)
DECLAREd data                :  #D
COMPOOL                      :  #P
Process Directory Entries    :  #E
Z-con to comsub              :  #Z
REMOTE data                  :  #R
Exclusive data               :  #X
```

In addition to CSECTs produced by the compiler, the HAL/S-FC system defines other CSECTs, some of which are referenced by compiler-emitted code. These CSECT types and their naming conventions are:

Z-con to library routine: #Q

Data for library routines: #L

3.3 Printed Data from Phase 2

3.3.1 Formatted Assembly Listing

Under control of the LIST compiler option, Phase 2 will produce a formatted, mnemonic listing of the object code produced for the compilation unit. This includes showing the statement number in which a label is located when a branch to a label is done. Also, the registers are identified with one of the following letters in front of the register number: “R” for general or “F” for floating point. The nominal execution time for each instruction (as defined in the “*Space Shuttle Model AP101-S Principles of Operation*”, Chapter 17) is provided. When a literal value is referenced, the register contents and decimal value for floating point instructions or the HEX and decimal values for non-floating point instructions are shown. For RS instructions with a base register of 3, the displacement is used instead of the base register. In this case, nothing is printed instead of “R3”. For BC instructions, alternate mnemonics are printed in the symbolic operand field to clarity the intent of the branch. In addition to the assembler-type mnemonic instruction listing, a full hexadecimal listing of the emitted code is also produced.

This object code listing is normally appended to the Phase 1 primary source listing as defined by the SYSPRINT DD card. However, use of the SDL compiler in addition to the LIST option causes the object code listing to be produced through the OUTPUT7 DD card. The listing thus produced is compatible with the ABSLIST function of the AP-101 Link Editor. The *HAL/SDL ICD* contains the detailed description of the ABSLIST format.

3.3.2 Symbol Information

Included in the listing is a table containing the symbol, type, ID, address, length (in both HEX and decimal) and block name.

3.3.3 RLD Information

The RLD information is printed in a table containing the position pointer, relocation pointer, flags, address and CSECT name for both position and relocation pointers. This section includes a title and a legend explaining each of the columns.

3.3.4 Variable Offset Table

The Variable Offset Table contains the location, base register, displacement, bias (zeroth element offset), name and length for each variable. This section includes a title and a legend explaining each of the columns.

3.3.5 Memory Map Table

A Memory Map table for local data is printed for the current compilation unit. It contains the variable name, length, offset, base register, displacement and scope. It also contains local block data information and alignment gap information.

3.3.6 Structure Template Layout Table

The Structure Template Layout Table contains the displacement from the root node, the length, the bias (zeroth element offset) and the name of each terminal and node in the template. Alignment gap information is also provided.

3.4 Symbol Table Augmentation

Phase 2 inherits an initialized symbol table from Phase 1. In the course of generating code, Phase 2 makes additions to the symbol table which are inherited, in turn, by Phase 3. These additions are generally in the area of data addressing.

Information is added in two of the symbol tables parallel arrays:

- The SYT_ADDR array is filled with data offset information indicating the relative location of data items within CSECTs
- The EXTENT array is filled with information about the size of the storage allocated to individual data items.

3.5 Statement Table Augmentation

Phase 3 inherits, in a secondary storage device, the statement table produced by Phase 1. If the ADDRS compiler option is in effect, Phase 1 leaves room in the statement table for beginning and ending addresses of individual HAL/S statements. This information is filled in by Phase 2 after the generation of the executable code has been performed. The completed statement table is then left for use by Phase 3.

4.0 Incremental #D (DATA_REMOTE Directive) REQUIREMENTS AND CODE DESIGN

4.1 Introduction

Incremental #D was a major enhancement to the compiler implemented in Phase 1 (Syntax Analysis), Phase 2 (Code Generation), and Phase 3 (SDF Generation), with the majority of the implementation in Phase 2. The following is taken from detailed requirements for Incremental #D as originally written.

The purpose of this chapter is to identify the detailed requirements, and to review the compiler source code design that satisfies these requirements, for the implementation of Support Software Change Request (SSCR) 11096 titled “Implement the Incremental #D Option”. The HAL/S FC compiler that is modified to implement the features of the Incremental #D option will be Release 24.0, and will also be known as the #D compiler.

4.2 Requirements and Code Design

The four primary requirements for the Incremental #D option are given in this section. Following each primary requirement is an interpretation of what that requirement means from the compiler’s perspective, the implied or derived detailed requirements which the design must satisfy, and a high-level picture of the compiler source code design.

When applicable, the code sections in the design pictures have been labeled with the number of the requirement that is satisfied (without the chapter number; i.e., requirement 4.2.1.2.1 is shown as 1.2.1). Note that after requirement 4.2.1.2.1 is met (see section 4.2.1.3), all remaining code segments shown throughout the document are only executed if the #D Directive was used. The if-then constructs that each of these code segments are imbedded in are not shown in order to avoid clutter and repetition.

No formal requirement exists for the compiler with regard to constraints on execution performance or Flight Software object code growth impacts incurred upon implementing and using the Incremental #D option.

4.2.1 Provide for Selective Migration of #D Data

- Provide for selective migration of program data CSECTs into upper memory via use of Extended Addressing hardware feature.

4.2.1.1 Interpretation

The HAL/S FC compiler will have some type of mechanism which will allow the HAL/S programmer to specify that all program local data declared within a single compilation unit (#D CSECT), will be referenced using Data Sector Extension (DSE) Addressing techniques. This type of addressing will allow the #D CSECT to be located in any memory sector.

4.2.1.2 Detailed Implied/Derived Requirements

4.2.1.2.1 The HAL/S FC Compiler shall {1} support a #D directive, “DATA REMOTE”, which will allow the #D CSECT to be placed into any memory sector (where previously it was restricted to sector 0 or 1). A single upper memory sector shall {2} be reserved for #D data and will be referred to as the **DSE Sector**. The data in this CSECT will be referred to as **Remote #D data**.

4.2.1.2.2 The #D Directive shall {1} have the same placement restrictions as the “D INCLUDE TEMPLATE” directive, and the coding of more than one #D Directive shall {2} produce the same result as coding one.

4.2.1.2.3 The #D Directive shall be illegal for COMPOOL compilation units.

4.2.1.2.4 When the #D directive is used, the REMOTE attribute shall be ignored for all locally declared data <u>except</u> NAME variables. Such data will be processed as Remote #D data.

4.2.1.2.5 The compiler shall set a flag (known as the DATA_REMOTE flag) in the Simulation Data File (SDF) to indicate when the #D directive is used, for the benefit of post processing tools.

4.2.1.2.6 When the #D directive is used, the REMOTE attribute of the module shall {1} be set in order to provide for auto placement of that module’s #D CSECT in a REMOTE bank by the PRELINK tool. After code generation, the REMOTE attribute of the module shall {2} be cleared so that the SDF flags for the module are not changed.

4.2.1.2.7 The requirements, restrictions, and impacts of Incremental #D described in this chapter shall {1} be in effect <u>only</u> when the #D directive is used. When the #D directive is <u>not</u> used, the #D compiler shall {2} produce object code which is identical to that produced by its predecessor compiler (except for changes due to any implemented DR fixes).

4.2.1.3 Compiler Implementation Design

COMMON DATA

NEW DATA_REMOTE Boolean 1.0
symbol table REMOTE attribute flags

PASS1	PASS2	PASS3
DWNTABLE CERRDECL add new error class CLASS_XR (1.0)		
STREAM.PROCESS_COMMENT if DATA_REMOTE directive used then if COMPOOL then error XR2 (1.2.3) if wrong placement then error XR1 (1.2.2) set DATA_REMOTE Boolean (1.2.1)		

IMPORTANT: All subsequent code segments are only executed if DATA_REMOTE is set. The check for this Boolean being set has been left out to avoid repetition (1.2.7)

PASS1	PASS2	PASS3
SYNTHESIZE in processing declared data: if REMOTE attribute then error XR3 (1.2.4)	NEW CSECT_TYPE a support routine to indicate if a data item is #D data: return ‘#D’ for locally declared data return ‘#R’ for locally declared data with REMOTE attribute set	##DRIVER use old PATCH_FLAG as the new DATA_REMOTE_FLAG for SDFs (1.2.5)
		INITIALISE set DATA_REMOTE_FLAG (1.2.5)
	INITIALISE.PROCENTRY IF NOT EXTERNAL then set module’s REMOTE attribute (1.2.6)	
	INITIALISE.SET_NEXT_AND_LOKS if CSECT_TYPE=#R then clear REMOTE attribute (1.2.4) (4.2.1)	
	EMIT_ESD_CARDS if module’s REMOTE attribute then set bit in ESD card for PRELINK and clear module’s REMOTE attribute (1.2.6)	

Figure 4-1 Provide for Selective Migration of #D Data

4.2.2 Provide for Management of Extended Addressing Feature

- Provide for management of the Extended Addressing feature at the appropriate points in the code:

  a. *Prolog*

  b. *Common Subroutine CALL locations.*

  c. *Runtime Library CALL locations.*

  d. *Epilog*

4.2.2.1 Interpretation

DSE management is defined as the techniques used to either clear or load the Data Sector Extension (DSE) of a register which is used to address the DSE sector.

DSE management shall be used only in modules compiled with the #D directive.

DSE management shall occur upon entry to a module, around all calls to external procedures and calls to Runtime Library (RTL) routines, and immediately before exit from a module.

4.2.2.2 Detailed Implied/Derived Requirements

4.2.2.2.1 Two fullword constants, $ZDSESET and $ZDSECLR, shall be supplied by FCOS to provide the compiler a means of setting and clearing DSE registers with the proper values.

4.2.2.2.2 For compilation units using the #D directive, the compiler shall generate object code in the prolog of each PROGRAM/COMSUB, which will initialize the DSEs of registers R1 and R3 with the Remote #D CSECT.

4.2.2.2.3 When the #D directive is used, the DSEs associated with the Remote #D registers shall be set to zero immediately prior to any external procedure and prior to any RTL call which utilizes R1 or R3 as a base register.

4.2.2.2.4 Immediately upon return from an external procedure or any RTL call which utilizes R1 or R3 as a base register, the DSEs associated with the Remote #D registers shall be restored to their previous values.

4.2.2.2.5 Before the exit from a COMSUB compilation unit using the #D directive, the DSEs associated with the Remote #D registers shall be set to zero. This is not done for PROGRAMs, as the SVC instruction that ends a PROGRAM needs the DSEs set to work properly.

4.2.2.3 Compiler Implementation Design

COMMON DATA	NEW DATA_REMOTE Boolean 1.0	
PASS1	PASS2	PASS3
	##DRIVER add new LDM instruction (2.0)	
	INITIALISE.SETUP_DATA set up external recognition of data contents $ZDSESET and $ZDSECLR (2.2.1)	
	GENERATE.EMIT_CALL clear DSEs with LDM before call (2.2.3) set DSEs with LDM after call (2.2.4)	
	GENERATE.BLOCK_OPEN set DSEs with LDM in prolog (2.2.2)	
	GENERATE.EMIT_RETURN clear DSEs with LDM in epilog of COMSUBS (2.2.5)	

Figure 4-2 Provide for Management of Extended Addressing Feature

4.2.3 Enforce Compiler Restrictions on #D Data

- Have all program data items take on the attribute “lives REMOTE”; enforce normal compiler restrictions on those data items:
  - *a. No assignments of Remote #D address into (16 bit) NAME variable.*
  - *b. No declaration of Remote #D data as EVENT type.*

4.2.3.1 Interpretation

For the purposes of syntactical and semantic analysis only, all data which is declared locally, and is within a single compilation unit compiled with the #D directive, shall be processed as if it had been declared with the REMOTE attribute. Again, this data will be referred to as **Remote #D data**.

All restrictions and limitations which currently exist for REMOTE data references will also exist for Remote #D data references. Refer to Section 8.10 of the *HAL/S-FC User's Manual* for more information.

Specifically, these restrictions are: a) no assignments of Remote #D addresses into (16-bit) NAME variables; b) not allowing the declaration of Remote #D EVENT variables.

4.2.3.2 Detailed Implied/Derived Requirements

4.2.3.2.1 The compiler shall internally "turn on" the REMOTE attribute for all local variables which are declared within a compilation unit containing the #D directive.

4.2.3.2.2 Once the REMOTE attribute is "turned on" for Remote #D data, the compiler shall process the HAL source code using existing error analysis techniques and error messages.

4.2.3.2.3 Because the REMOTE attribute must be "turned off" during code generation (see section 4.2.4.2**)**, supplemental error checking shall be used for parameter checking.

4.2.3.2.4 It shall be legal to assign Remote #D addresses into (32-bit) NAME REMOTE variables. This requires a conversion of a YCON (16-bit offset) plus associated DSE into ZCON format.

4.2.3.2.5 In order for Remote #D data to be passed by reference to REMOTE Runtime Library routines the way current REMOTE data is handled, the YCON to ZCON conversion of 4.2.3.2.4 shall be used.

4.2.3.2.6 The HAL/S WRITE statement shall be the means by which Remote #D data can be output during testing.

4.2.3.3 Current Error Message Usage

The following existing errors will be used to indicate when the use of Remote #D data has violated a compiler restriction:

DI107 - SEVERITY 1
ATTEMPT TO INITIALIZE A NON-REMOTE NAME WITH A REMOTE VARIABLE

XQ102 - SEVERITY 2
ATTEMPT TO ASSIGN NAME OF REMOTE DATA ITEM TO A 16 BIT NAME VARIABLE

FT111 - SEVERITY 2
MISMATCHED ARGUMENTS IN %NAMECOPY STATEMENT. REMOTE SOURCE NOT ALLOWED WITH NON-REMOTE DESTINATION IN %NAMECOPY STATEMENT

FT110 - SEVERITY 2
MISMATCHED ARGUMENTS IN %NAMEADD STATEMENT. REMOTE SOURCE NOT ALLOWED WITH NON-REMOTE DESTINATION IN %NAMEADD STATEMENT

FT112 SEVERITY 2
PARAMETER #?? MAY NOT BE NAME(NAMEVAR) IF NAMEVAR LIVES REMOTE

DA9 - SEVERITY 2
ILLEGAL ATTRIBUTE SPECIFIED FOR EVENT DATA TYPE

4.2.3.4 FCOS Restrictions on #D Data

Due to FCOS restrictions, certain real time operations cannot occur with DATA_REMOTE in effect (i.e. outside of sector 0/1). An XR4 error message (severity 2) will be generated for any of the following conditions when DATA_REMOTE is in effect:

- EQUATE EXTERNAL to a #D symbol;
- SCHEDULE statement with an ON, WHILE or UNTIL event option in a non-reentrant module;
- WAIT FOR statement in a non-reentrant module.

4.2.3.5 Compiler Implementation Design

COMMON DATA

NEW DATA_REMOTE Boolean

symbol table REMOTE attribute flags

PASS1

SYNTHESIZE
in processing declared data:
set REMOTE attribute (3.2.1)

PASS2

new GENERATE.PARM_STAT
if CSECT_TYPE of parameter is #D (3.2.2)
then error FT108 (3.2.3)

GENERATE.VMCALL
CHAR_CALL, CTON,
COMPARE_STRUCTURE
GEN_CLASS0.USE_MSTRUC
GEN_CLASS0
setup to call REMOTE RTL routine if
CSECT_TYPE of parameter is #D (3.2.5)

GENERATE.BIT_STORE
CHAR_CALL, CTON,
GEN_CLASS0
GEN_CLASS1
(3.2.3)

GENERATE.STRUCTFIX,
GET_OPERAND
if CSECT_TYPE of data is #D
then set Boolean LIVES_REMOTE to trigger
NAME error-checking
(3.2.2)

GENERATE.CLASS0.SET_IO_LIST
if CSECT_TYPE of aggregate is #D
then copy to stack before calling, then
copy to stack before calling output RTL
routine
(3.2.6)

GENERATE.FORCE_ADDRESS
if data assigned to NAME_REMOTE does
not have REMOTE attribute
then convert to YCON+DSE to ZCON
(3.2.4)

PASS3

Figure 4-3 Enforce Compiler Restrictions on #D Data

4.2.4 Manipulate #D Data Using Extended Addressing Techniques

Manipulation of program data addresses should make use of Extended Addressing techniques where appropriate.

4.2.4.1 Interpretation

The HAL/S FC compiler normally uses ZCON (32-bit indirect address constant) addressing to manipulate (load and store) REMOTE data. Remote #D data will be manipulated using YCON (16-bit base register) addressing techniques.

The DSEs of the registers used to address the Remote #D data shall contain the DSE sector number. This allows standard YCON addressing to be used to point to REMOTE data. Although all eight general registers have associated DSE registers, it is only valid for the compiler to use the DSEs associated with registers R0, R1, R2, and R3, as FCOS only preserves these four during context switches.

There <u>may</u> be some circumstances where DSE addressing can not be appropriately used, and ZCON addressing must be used instead; for instance, in assignments of Remote #D addresses to NAME REMOTE variables (see section 4.2.3.2).

4.2.4.2 Detailed Implied/Derived Requirements

4.2.4.2.1 Once the Remote #D registers and their DSEs have been initialized, the Remote #D data shall be referenced and manipulated in the same manner as local #D data is manipulated today. This requires the REMOTE attribute of Remote #D data to be “turned off”, which in turn ensures that the SDF flags for local #D data are not changed.

4.2.4.2.2 When the #D directive is used, general purpose registers R1 and R3 shall be reserved for addressing Remote #D data, and only Remote #D data. These registers will be referred to as the Remote #D registers.

4.2.4.2.3 When the #D directive is used, the #D compiler shall enforce restrictions such that when base addressing is used, only general register R2 (and no others) is used to reference COMPOOL data. Otherwise, immediate addressing (no-base) is used for referencing COMPOOL data.

4.2.4.2.4 When the #D directive is used, special handling of the Remote #D registers and their DSEs shall be required for the MVH instruction to function properly.

4.2.4.3 Compiler Implementation Design

COMMON DATA

NEW DATA_REMOTE Boolean

symbol table REMOTE attribute flags

PASS1 PASS2 PASS3

INITIALISE.SET_NEST_AND_LOCKS
if CSECT_TYPE = #R then (1.2.4)
clear REMOTE attribute (4.2.1)

new GENERATE.CHECK_RESTORE in R1/R3 for base addressing, or the next instruction is a branch
then restore R1/R3 (4.2.2)

GENERATE.EMITRR.EMITRX,
EMITSI, EMITP, EMITSP
if R1/R3 has been changed from the original#D pointer value
then call CHECK_RESTORE (4.2.2)

new GENERATE.REG_STAT
perform register restriction:
four basic cases:
1) loading MVH source operand:
no change in register allocation
2) loading address for RTL call:
no change in register allocation
3) target register is R1 or R3:
if loading NAME or formal parameter
then use R2 instead if loading non #D data
then use R2 instead
4) target register is R2:
if loading #D data (4.2.2)
then use R3 instead (4.2.3)

GENERATE.GET_R,
RELOAD_ADDRESSING,
GUARANTEE_ADDRESSABLE,
FORCE_ADDRESS, CHECK_SI,
ADDRESS_STRUCTURE,
REF_STRUCTURE,
FORCE_ADDRESS_LIT
call REG_STAT after current allocation methods select a target register (4.2.2) (4.2.3)

GENERATE.USE_MVH,
if CSECT_TYPE of destination is #D
then call GET_R to use R3 after a load of R3, clear its msb (4.2.4)

Figure 4-4 Manipulate #D Data Using Extended Addressing Techniques

5.0 PHASE 3 - SIMULATION DATA FILE GENERATION

Phase 3 of the HAL/S-FC compiler has the primary function of providing Simulation Data Files (SDFs) for each unit of compilation. Phase 3 also produces user-oriented printouts upon special request. This section deals with the following Phase 3 functions:

- SDF generation
- Printed data

5.1 SDF Generation

Phase 3 synthesizes the SDF for a compilation unit from data received from previous Phases of the compiler. This data is primarily in two areas: a) The symbol table, created by Phase 1 and augmented by Phase 2, and b) The statement table similarly created by Phase 1 and 2.

The detailed format of an SDF is controlled by the *HAL/SDL ICD*. The reader is referred there for details of SDF design beyond the overview presented in the next section.

5.1.1 Overall SDF Design

A Simulation Data File (SDF) is an organized and directoried collection of block, symbol, and statement data which is created by the HAL/S compiler from a single unit of compilation and stored in a permanent form for later use by simulation processors.

There are basically three types of information contained in an SDF. These are:

1. Symbol Data - contains the attributes of HAL/S symbols (labels and variables) such as name, class and type, relative core address, number of bytes in core occupied, etc. Also contains arrayness and dimensionality for arrayed variables, template linkages for elements of structures, and cross-reference information listing all statements within the compilation unit that may assign values to the symbol.
2. Statement Data - contains the attributes of HAL/S statements such as type, Statement Reference Numbers (SRNs) if specified by the user, indices for all labels attached to each statement, and indices for all variables which may be assigned values by that statement. Also may optionally contain the relative core addresses of the first and last executable instructions emitted for that statement.
3. Block and Directory Data - contains information about each HAL/S block and the symbols and statements contained within that block, plus information concerning the layout and organization of the SDF which minimizes the time needed to access desired data entries.

An SDF is produced for all compilation units unless suppressed by the user (the TABLES/NOTABLES option). In the case of COMPOOL compilations, the SDF becomes somewhat simplified, having no executable statements and, consequently, no cross-reference data for its symbols.

SDFs are created as members of Partitioned Data Sets (PDSs) and are assigned names o

f the form ##CCCCCC, where CCCCCC is the first six characters of the compilation unit name with any and all underscore characters removed. (Example: the SDFs for the compilation units SAMPLER and TEST_SAMPLE would be assigned the names ##SAMPLE and ##TESTSA, respectively). The members are written in fixed record format with a block size and logical record length of 1680.

The structure of the SDF will support three efficient types of access:

1. Given the name of a symbol, and the name of the block in which it was declared, obtain the attributes of the symbol.
2. Given a Statement Reference Number (SRN), obtain the attributes of the statement.
3. Given an Internal Statement Number (ISN), obtain the attributes of the statement.

In access methods 1 and 2, the SDF directory plays a key role. When the symbol name and its block are given, the directory will identify which particular physical record of the SDF contains the corresponding fixed-length Symbol Node. Once this record has been read into core, a simple and fast binary search will locate the symbol node which in turn "points" directly to the attributes of the symbol which are contained within a variable-length Symbol Data Cell. A virtually identical procedure can be used to locate statement data when the SRN is given. In this case, the fixed-length nodes involved in the binary search are called Statement Nodes, and their corresponding variable-length data cells are called Statement Data Cells.

In contrast to access methods 1 and 2, which require directory help followed by binary searches, method 3 is direct. This is because there is a one-to-one correspondence between the ISN (compiler-generated Internal Statement Number) and the order of the Statement Nodes. The *HAL/SDL ICD* contains detailed descriptions of the SDF organization.

5.2 Phase 3 Printed Data

For each invocation of Phase 3, a set of tabular data is printed. The information presented deals with parameters relating to the SDF produced, such as number of SDF pages, numbers of block, symbol, and statement nodes, etc.

In addition to the information which is always printed, two optional printouts are available. Under control of the TABLST compiler option, the user may request that symbolic, structured dump of the SDF be provided. In addition, under control of the TABDMP compiler option, the user may request that the contents of the SDF be displayed in a hexadecimal format, page by page.

Immediately following the Phase 3 output, but before any optional output from TABDMP or TABLST, an advisory section is printed starting on a new page. This section contains information for the programmer about improvements that could be made to the source code of the program.

This page intentionally left blank.

6.0 RUNTIME LIBRARY

6.1 Introduction

This section describes the HAL/S-FC runtime library as used to support the HAL/S-FC compiler. The Primary Avionics Software System (PASS) and Backup Flight Software (BFS) versions of the HAL/S-FC compiler use the same source code to build their respective runtime libraries. The material is organized to present both general design concepts and detailed interface and algorithm information. Following an introductory discussion of general conventions used throughout the library, descriptions of the individual routines are grouped according to the basic type of the routine. Each group is introduced by a quick-reference chart containing basic interface data.

6.2 Basics and Conventions

6.2.1 Origin and Format

The HAL/S-FC compilers are supplied with runtime libraries. The library for PASS is a partitioned dataset (PDS) containing members in AP-101 load module format. The library for BFS is a PDS that contains object modules in the Eclipse format.

The runtime library objects are built by assembling the identically named members of a source library that consists of statements written in AP-101 Basic Assembly Language (BAL). Each source library member is assembled with the value of the &SYSPARM system variable set to 'PASS' and 'BFS' respectively. The &SYSPARM variable is used in the macro library routines to isolate code sequences that are unique to either PASS or BFS. A macro library is used to standardize frequently used sequences of source code.

The runtime libraries are built from these runtime library objects using methods that differ substantially between PASS and BFS. For PASS, some source library members have more than one entry point, in which case library ALIAS names are generated for each entry using the FIXOBJ tool. The AP-101 link editor is then invoked to generate the library in load module format. For BFS, the object modules are converted from AP-101 to Eclipse format using the SATSOBJ tool. The input commands to SATSOBJ specify that members beginning with '#L' are to be tagged as DATA type and all other library members as CODE type. The type assigned to a library member is used by the PILOT (Program Integration and Loading Tool) program to determine where the member should be placed in memory. The BFS runtime library is also marked with Version 0 to ensure compatibility with other objects generated by flight software.

Also included with the HAL/S-FC compilers are the ZCON libraries associated with PASS and BFS. The PASS ZCON library is created by assembling its associated source code using the same procedures that are used for the PASS runtime library. The BFS ZCON library is created using a special tool, BLDQCON, that simply requires a member list from the runtime library PDS as its input. The BFS ZCON library is also marked with Version 0 to ensure compatibility with other objects generated.

6.2.2 Purpose

The runtime library is used to supply routines, data and interfaces which are needed to execute a HAL/S program or group of programs which are not produced by the compiler's code generator. Most of the library consists of subroutines which are called from compiler generated code in a HAL/S statement.

6.2.3 Intrinsics and Procedure Routines

The library routines are divided into two groups: intrinsics and procedures. The main distinction is that procedure routines save the passed contents of all fixed point registers, while intrinsics do not. For this reason, a procedure can call another routine (e.g., vector (VV10S3) magnitude calls SQRT), but an intrinsic cannot. Intrinsics do not have a new stack level and therefore do not have any stack work areas. Because intrinsics do not save all passed contents of fixed point registers, they cannot restore them and must not destroy any register contents that must be returned to the calling program. Expansions of the macros within intrinsic routines are different from the expansions within procedure routines.

6.2.4 Register Conventions in Runtime Library Routines

6.2.4.1 General Purpose Registers R0-R7.

R1-R3,	R5-R7	:	free use;
	R4	:	return address during calling and exiting intrinsics, otherwise free use;
	R0	:	stack base;
Parameters		:	Intrinsics: any or all of R1, R2, R3, R5, R6, R7 can be used for parameter passing. Procedures: any or all of R2, R4, R5, R6, R7 can be used for parameter passing.

6.2.4.2 Floating Point Registers F0-F7.

Internal compiler tables indicate which floating point registers are used by each RTL routine. Any register which is used in an RTL routine will be reloaded after returning from that routine before further use. The only exception to this rule is registers which are not flagged in the compiler's internal tables, but are instead saved and restored by the RTL routine upon entry and exit from the routine.

6.2.4.3 Interface Conventions.

In addition to the parameter passing conventions summarized in general form in the previous two sections and given in detail in the individual library routine descriptions, the compiler has information defining the linkage conventions and register usage for each routine.

This section contains that information in a list formatted in four columns as follows:

NAME	The primary or secondary entry point name.
CALL TYPE	Either PROCEDURE or INTRINSIC to distinguish between routines which must be called via the SCAL instruction and those that must be called using BAL.
BANK0	YES indicates that the routine will always reside in Sector 0 of the GPC and may therefore always be called directly (no ZCON needed). NO indicates that the routine may reside in a sector other than 0 and must therefore be called via a long indirect address constant (ZCON).
Registers assumed to be modified	A list of registers which the compiler assumes to be modified across a call to the routine. Any registers not listed may be assumed to remain unmodified and therefore to maintain their previous contents. Underlined registers are not actually modified by the RTL routine, but the compiler still assumes that they are.

Any modifications to compiler or library should be made carefully so as to maintain this interface properly.

	ROUTINE	CALL TYPE	BANK0	REGISTERS ASSUMED TO BE MODIFIED
1	ACOS	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
2	ACOSH	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
3	ASIN	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
4	ASINH	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
5	ATAN	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
6	ATANH	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
7	BIN	PROCEDURE	NO	F0, F1
8	BOUT	PROCEDURE	NO	F0, F1
9	BTOC	INTRINSIC	NO	R1, R2, R3, R4, R5, R6, R7
10	CAS	INTRINSIC	NO	R1, R2, R3, R4, R5
11	CASP	INTRINSIC	NO	R1, R2, R3, R4, R5, R6
12	CASPV	INTRINSIC	NO	R1, R2, R3, R4, R5, R6
13	CASR	PROCEDURE	NO	NONE
14	CASRP	PROCEDURE	NO	NONE
15	CASRPV	PROCEDURE	NO	NONE
16	CASRV	PROCEDURE	NO	NONE
17	CASV	INTRINSIC	NO	R1, R2, R3, R4, R5,
18	CAT	INTRINSIC	NO	R1, R2, R3, R4, R5, R6, R7, F0, F1
19	CATV	INTRINSIC	NO	R1, R2, R3, R4, R5, R6, R7, F0, F1
20	CEIL	INTRINSIC	YES	R4, R5, F0, F1
21	CIN	PROCEDURE	NO	NONE
22	CINDEX	PROCEDURE	NO	R5, F0, F1, F2, F3, F4, F5
23	CINP	PROCEDURE	NO	F0, F1
24	CLJSTV	PROCEDURE	NO	F0, F1
25	COLUMN	PROCEDURE	NO	F0, F1
26	COS	INTRINSIC	NO	R2, R3, R4, , F0, F1, F2, F3, F4, F5
27	COSH	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
28	COUT	PROCEDURE	NO	NONE
29	COUTP	PROCEDURE	NO	NONE
30	CPAS	PROCEDURE	NO	F0, F1
31	CPASP	PROCEDURE	NO	F0, F1
32	CPASR	PROCEDURE	NO	F0, F1
33	CPASRP	PROCEDURE	NO	F0, F1

	ROUTINE	CALL TYPE	BANK0	REGISTERS ASSUMED TO BE MODIFIED
34	CPR	INTRINSIC	NO	R2, R3, R4, R5, R6
35	CPRA	PROCEDURE	NO	NONE
36	CPRC	INTRINSIC	NO	R2, R3, R4, R5, R6
37	CPSLD	PROCEDURE	NO	R5, F0, F1
38	CPSLDP	PROCEDURE	NO	R5, F0, F1
39	CPSST	PROCEDURE	NO	R5, F0, F1
40	CPSSTP	PROCEDURE	NO	R5, F0, F1
41	CRJSTV	PROCEDURE	NO	F0, F1
42	CSHAPQ	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
43	CSLD	PROCEDURE	NO	R5, F0, F1
44	CSLDP	PROCEDURE	NO	R5, F0, F1
45	CSST	PROCEDURE	NO	R5, F0, F1
46	CSSTP	PROCEDURE	NO	R5, F0, F1
47	CSTR	PROCEDURE	NO	NONE
48	CSTRUC	INTRINSIC	NO	R2, R3, R4, R5, R6
49	CTOB	PROCEDURE	NO	R5, F0, F1
50	CTOD	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
51	CTOE	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
52	CTOH	PROCEDURE	NO	R5, F0, F1
53	CTOI	PROCEDURE	NO	R5, F0, F1
54	CTOK	PROCEDURE	NO	R5, F0, F1
55	CTOO	PROCEDURE	NO	R5, F0, F1
56	CTOX	PROCEDURE	NO	R5, F0, F1
57	CTRIMV	PROCEDURE	NO	F0, F1
58	DACOS	PROCEDURE	NO	F0, F1, F2, F3, F4, F5, F6, F7
59	DACOSH	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
60	DASIN	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
61	DASINH	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
62	DATAN	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
63	DATANH	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
64	DATAN2	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
65	DCEIL	INTRINSIC	YES	R4, R5, F0, F1
66	DCOS	PROCEDURE	NO	F0, F1, F2, F3, F4, F5, F6
67	DCOSH	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
68	DEXP	PROCEDURE	NO	F0, F1, F2, F3
69	DFLOOR	INTRINSIC	YES	R4, R5, F0, F1
70	DIN	PROCEDURE	NO	F0, F1
71	DLOG	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
72	DMAX	INTRINSIC	NO	R2, R4, R5, F0, F1
73	DMDVAL	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
74	DMIN	INTRINSIC	NO	R2, R4, R5, F0, F1
75	DMOD	INTRINSIC	NO	R4, F0, F1, F2, F3, F4, F5, F6, F7
76	DOUT	PROCEDURE	NO	F0, F1
77	DPROD	INTRINSIC	NO	R2, R4, R5, F0, F1
78	DPWRD	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
79	DPWRH	PROCEDURE	NO	F0, F1, F2, F3
80	DPWRI	PROCEDURE	NO	F0, F1, F2, F3
81	DROUND	INTRINSIC	YES	R4, R5, F0, F1
82	DSIN	PROCEDURE	NO	F0, F1, F2, F3, F4, F5, F5
83	DSINH	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
84	DSLD	PROCEDURE	NO	R5
85	DSNCS	PROCEDURE	NO	F0, F1, F2, F3, F4, F5, F6
86	DSQRT	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
87	DSST	PROCEDURE	NO	NONE
88	DSUM	INTRINSIC	NO	R2, R4, R5, F0, F1
89	DTAN	PROCEDURE	NO	F0, F1, F2, F3, F4, F5

	ROUTINE	CALL TYPE	BANK0	REGISTERS ASSUMED TO BE MODIFIED
90	DTANH	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
91	DTOC	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
92	DTOH	INTRINSIC	YES	R4, R5, F0, F1
93	DTOI	INTRINSIC	YES	R4, R5, F0, F1
94	DTRUNC	INTRINSIC	YES	R4, R5, F0, F1
95	EATAN2	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
96	EIN	PROCEDURE	NO	F0, F1
97	EMAX	INTRINSIC	NO	R2, R4, R5, F0, F1
98	EMIN	INTRINSIC	NO	R2, R4, R5, F0, F1
99	EMOD	INTRINSIC	NO	R4, F0, F1, F2, F3, F4, F5
100	EOUT	PROCEDURE	NO	F0,F1
101	EPROD	INTRINSIC	NO	R2, R4, R5, F0, F1
102	EPWR3	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
103	EPWRH	PROCEDURE	NO	F0, F1, F2, F3
104	EPWRI	PROCEDURE	NO	F0, F1, F2, F3
105	ESUM	INTRINSIC	NO	R2, R4, R5, F0, F1
106	ETOC	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
107	ETOH	INTRINSIC	YES	R4, R5, F0, F1
108	ETOI	INTRINSIC	YES	R4, R5, F0, F1
109	EXP	PROCEDURE	NO	F0, F1, F2, F3
110	FLOOR	INLINE	YES	N/A (RTL not used)
111	GTBYTE	INTRINSIC	NO	R2, R4, R5, F0, F1
112	HIN	PROCEDURE	NO	F0, F1
113	HMAX	INTRINSIC	NO	R2, R4, R5, R6
114	HMIN	INTRINSIC	NO	R2, R4, R5, R6
115	HMOD	INTRINSIC	NO	R2, R4, R5, R6, R7
116	HOUT	PROCEDURE	NO	F0, F1
117	HPROD	INTRINSIC	NO	R2, R4, R5, R6
118	HPWRH	PROCEDURE	NO	R5
119	HREM	INTRINSIC	NO	R2, R4, R5, R6, R7
120	HSUM	INTRINSIC	NO	R2, R4, R5, R6
121	HTOC	PROCEDURE	NO	NONE
122	IIN	PROCEDURE	NO	F0, F1
123	IMAX	INTRINSIC	NO	R2, R4, R5, R6
124	IMIN	INTRINSIC	NO	R2, R4, R5, R6
125	IMOD	INTRINSIC	NO	R2, R4, R5, R6, R7
126	IOINIT	PROCEDURE	NO	F0, F1
127	IOUT	PROCEDURE	NO	F0, F1
128	IPROD	INTRINSIC	NO	R2, R4, R5, R6, R7
129	IPWRH	PROCEDURE	NO	R5
130	IPWRI	PROCEDURE	NO	R5
131	IREM	INTRINSIC	NO	R2, R4, R5, R6, R7
132	ISUM	INTRINSIC	NO	R2, R4, R5, R6
133	ITOC	PROCEDURE	NO	NONE
134	ITOD	INTRINSIC	YES	R4, R5, F0, F1
135	ITOE	INTRINSIC	YES	R4, R5, F0, F1
136	KTOC	INTRINSIC	NO	R1, R2, R3, R4, R5, R6, R7, F0, F1
137	LINE	PROCEDURE	NO	F0, F1
138	LOG	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
139	MMRDNP	PROCEDURE	NO	NONE
140	MMRSNP	PROCEDURE	NO	NONE
141	MMWDNP	PROCEDURE	NO	F0, F1
142	MMWSNP	PROCEDURE	NO	F0, F1
143	MM0DNP	INTRINSIC	NO	R1, R3, R4, R5, R6, R7, F0, F1
144	MM0SNP	INTRINSIC	NO	R1, R3, R4, R5, R6, R7, F0, F1
145	MM1DNP	INTRINSIC	NO	R1, R2, R3, R4, R5, R6, R7, F0, F1, F2, F3

	ROUTINE	CALL TYPE	BANK0	REGISTERS ASSUMED TO BE MODIFIED
146	MM1SNP	INTRINSIC	NO	R1, R2, R3, R4, R5, R6, R7, F0, F1
147	MM1TNP	INTRINSIC	NO	R1, R2, R3, R4, R5, R6, R7, F0, F1, F2, F3
148	MM1WNP	INTRINSIC	NO	R1, R2, R3, R4, R5, R6, R7, F0, F1, F2, F3
149	MM11DN	INTRINSIC	NO	R1, R2, R3, R4, R5, R6, R7, F0, F1, F2, F3
150	MM11D3	INTRINSIC	NO	R1, R2, R4, R5, R7, F0, F1, F2, F3, F4, F5
151	MM11SN	INTRINSIC	NO	R1, R2, R3, R4, R5, R6, R7, F0, F1
152	MM11S3	INTRINSIC	NO	R1, R2, R4, R5, F0, F1, F2, F3
153	MM12DN	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
154	MM12D3	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
155	MM12SN	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
156	MM12S3	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
157	MM13DN	INTRINSIC	NO	R2, R4, R5, R6, F0, F1
158	MM13D3	INTRINSIC	NO	R2, R4, F0, F1
159	MM13SN	INTRINSIC	NO	R2, R4, R5, R6, F0, F1
160	MM13S3	INTRINSIC	NO	R2, R4, F0, F1
161	MM14DN	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
162	MM14D3	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
163	MM14SN	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
164	MM14S3	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
165	MM15DN	INTRINSIC	NO	R1, R4, R5, R6, R7, F0, F1, F2, F3
166	MM15SN	INTRINSIC	NO	R1, R4, R5, R6, R7, F0, F1, F2, F3
167	MM17DN	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
168	MM17D3	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
169	MM17SN	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
170	MM17S3	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
171	MM6DN	INTRINSIC	NO	R1, R2, R3, R4, R5, R6, R7, F0, F1, F2, F3, F4, F5
172	MM6D3	INTRINSIC	NO	R1, R2, R3, R4, R5, R6, R7, F0, F1, F2, F3, F4, F5
173	MM6SN	INTRINSIC	NO	R1, R2, R3, R4, R5, R6, R7, F0, F1, F2, F3, F4, F5
174	MM6S3	INTRINSIC	NO	R1, R2, R3, R4, R5, R6, R7, F0, F1, F2, F3, F4, F5
175	MR0DNP	PROCEDURE	NO	F0, F1
176	MR0SNP	PROCEDURE	NO	F0, F1
177	MR1DNP	PROCEDURE	NO	F0, F1
178	MR1SNP	PROCEDURE	NO	F0, F1
179	MR1TNP	PROCEDURE	NO	F0, F1
180	MR1WNP	PROCEDURE	NO	F0, F1
181	MSTR	PROCEDURE	NO	NONE
182		NOT USED		
183	MV6DN	INTRINSIC	NO	R1, R2, R3, R4, R5, R6, R7, F0, F1, F2, F3, F4, F5
184	MV6D3	INTRINSIC	NO	R1, R2, R3, R4, R6, F0, F1, F2, F3
185	MV6SN	INTRINSIC	NO	R1, R2, R3, R4, R5, R6, R7, F0, F1, F2, F3, F4, F5
186	MV6S3	INTRINSIC	NO	R1, R2, R3, R4, F0, F1, F2, F3
187	OTOC	INTRINSIC	NO	R1, R2, R3, R4, R5, R6, R7, F0, F1
188	OUTER1	PROCEDURE	NO	F0, F1
189	PAGE	PROCEDURE	NO	F0, F1
190	QSHAPQ	PROCEDURE	NO	F0, F1
191	RANDG	PROCEDURE	NO	F0, F1, F2, F3
192	RANDOM	PROCEDURE	NO	F0, F1, F2, F3
193	ROUND	INTRINSIC	YES	R4, R5, F0, F1
194	SIN	INTRINSIC	NO	R2, R3, R4, F0, F1, F2, F3, F4, F5
195	SINH	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
196	SKIP	PROCEDURE	NO	F0, F1
197	SNCS	INTRINSIC	NO	R2, R3, R4, F0, F1, F2, F3, F4, F5
198	SQRT	INTRINSIC	NO	R1, R4, R5, R6, R7, F0, F1, F2, F3
199	STBYTE	INTRINSIC	NO	R1, R4, R5, F0, F1
200	TAB	PROCEDURE	NO	F0, F1
201	TAN	PROCEDURE	NO	F0, F1, F2, F3, F4, F5

	ROUTINE	CALL TYPE	BANK0	REGISTERS ASSUMED TO BE MODIFIED
202	TANH	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
203	TRUNC	INTRINSIC	YES	R4, R5, F0, F1
204	VM6DN	INTRINSIC	NO	R1, R2, R3, R4, R5, R6, R7, F0, F1, F2, F3, F4, F5
205	VM6D3	INTRINSIC	NO	R1, R2, R3, R4, R5, F0, F1, F2, F3, F4, F5
206	VM6SN	INTRINSIC	NO	R1, R2, R3, R4, R5, R6, R7, F0, F1, F2, F3, F4, F5
207	VM6S3	INTRINSIC	NO	R1, R2, R3, R4, R5, F0, F1, F2, F3
208	VO6DN	INTRINSIC	NO	R1, R2, R3, R4, R5, R6, R7, F0, F1, F4, F5
209	VO6D3	INTRINSIC	NO	R1, R2, R3, R4, R5, R6, F0, F1
210	VO6SN	INTRINSIC	NO	R1, R2, R3, R4, R5, R6, R7, F0, F1, F4, F5
211	VO6S3	INTRINSIC	NO	R1, R2, R3, R4, R5, R6, F0, F1
212	VR0DN	PROCEDURE	NO	F0, F1
213	VR0DNP	PROCEDURE	NO	F0, F1
214	VR0SN	PROCEDURE	NO	F0, F1
215	VR0SNP	PROCEDURE	NO	F0, F1
216	VR1DN	PROCEDURE	NO	F0, F1
217	VR1DNP	PROCEDURE	NO	F0, F1
218	VR1SN	PROCEDURE	NO	F0, F1
219	VR1SNP	PROCEDURE	NO	F0, F1
220	VR1TN	PROCEDURE	NO	F0, F1
221	VR1TNP	PROCEDURE	NO	F0, F1
222	VR1WN	PROCEDURE	NO	F0, F1
223	VR1WNP	PROCEDURE	NO	F0, F1
224	VV0DN	INTRINSIC	NO	R1, R4, R5, F0, F1
225	VV0DNP	INTRINSIC	NO	R1, R4, R5, R7, F0, F1
226	VV0SN	INTRINSIC	NO	R1, R4, R5, F0
227	VV0SNP	INTRINSIC	NO	R1, R4, R5, R7, F0, F1
228	VV1DN	INTRINSIC	NO	R1, R2, R4, R5, F0, F1
229	VV1DNP	INTRINSIC	NO	R1, R2, R4, R5, R6, R7, F0, F1
230	VV1D3	INTRINSIC	NO	R1, R2, R4, F0, F1, F2, F3, F4, F5
231	VV1D3P	INTRINSIC	NO	R1, R2, R4, R5, R6, R7, F0, F1
232	VV1SN	INTRINSIC	NO	R1, R2, R4, R5, F0, F1
233	VV1SNP	INTRINSIC	NO	R1, R2, R4, R5, R6, R7, F0, F1
234	VV1S3	INTRINSIC	NO	R1, R2, R4, F0, F1, F2, F3, F4, F5
235	VV1S3P	INTRINSIC	NO	R1, R2, R4, R5, R6, R7, F0, F1
236	VV1TN	INTRINSIC	NO	R1, R2, R4, R5, F0, F1
237	VV1TNP	INTRINSIC	NO	R1, R2, R4, R5, R6, R7, F0, F1
238	VV1T3	INTRINSIC	NO	R1, R2, R4, F0, F1, F2, F3, F4, F5
239	VV1T3P	INTRINSIC	NO	R1, R2, R4, R5, R6, R7, F0, F1
240	VV1WN	INTRINSIC	NO	R1, R2, R4, R5, F0, F1
241	VV1WNP	INTRINSIC	NO	R1, R2, R4, R5, R6, R7, F0, F1
242	VV1W3	INTRINSIC	NO	R1, R2, R4, F0, F1
243	VV1W3P	INTRINSIC	NO	R1, R2, R4, R5, R6, R7, F0, F1
244	VV10DN	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
245	VV10D3	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
246	VV10SN	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
247	VV10S3	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
248	VV2DN	INTRINSIC	NO	R1, R2, R3, R4, R5, F0, F1
249	VV2D3	INTRINSIC	NO	R1, R2, R3, R4, F0, F1, F2, F3, F4, F5
250	VV2SN	INTRINSIC	NO	R1, R2, R3, R4, R5, F0, F1
251	VV2S3	INTRINSIC	NO	R1, R2, R3, R4, F0, F1, F2, F3, F4, F5
252	VV3DN	INTRINSIC	NO	R1, R2, R3, R4, R5, F0, F1
253	VV3D3	INTRINSIC	NO	R1, R2, R3, R4, F0, F1
254	VV3SN	INTRINSIC	NO	R1, R2, R3, R4, R5, F0, F1
255	VV3S3	INTRINSIC	NO	R1, R2, R3, R4, F0, F1, F2, F3, F4, F5
256	VV4DN	INTRINSIC	NO	R1, R2, R4, R5, F0, F1, F2, F3
257	VV4D3	INTRINSIC	NO	R1, R2, R4, F0, F1, F2, F3

	ROUTINE	CALL TYPE	BANK0	REGISTERS ASSUMED TO BE MODIFIED
258	VV4SN	INTRINSIC	NO	R1, R2, R4, R5, F0, F1, F2, F3
259	VV4S3	INTRINSIC	NO	R1, R2, R4, F0, F1 F2, F3
260	VV5DN	INTRINSIC	NO	R1, R2, R4, R5, F0, F1, F2, F3, F4, F5, F6,F7
261	VV5D3	INTRINSIC	NO	R1, R2, R4, F0, F1, F2, F3, F4, F5, F6,F7
262	VV5SN	INTRINSIC	NO	R1, R2, R4, R5, F0, F1, F2, F3
263	VV5S3	INTRINSIC	NO	R1, R2, R4, F0, F1, F2, F3
264	VV6DN	INTRINSIC	NO	R1, R2, R3, R4, R5, F0, F1, F2, F3
265	VV6D3	INTRINSIC	NO	R2, R3, R4, F0, F1, F2, F3
266	VV6SN	INTRINSIC	NO	R1, R2, R3, R4, R5, F0, F1, F2, F3
267	VV6S3	INTRINSIC	NO	R2, R3, R4, F0, F1, F2, F3
268	VV7DN	INTRINSIC	NO	R1, R2, R4, R5, F0, F1
269	VV7D3	INTRINSIC	NO	R1, R2, R4, F0, F1, F2, F3, F4, F5
270	VV7SN	INTRINSIC	NO	R1, R2, R4, R5, F0, F1
271	VV7S3	INTRINSIC	NO	R1, R2, R4, F0, F1, F2, F3, F4, F5
272	VV8DN	INTRINSIC	NO	R1, R2, R3, R4, R5, F0, F1
273	VV8D3	INTRINSIC	NO	R1, R2, R3, R4, R5, F0, F1
274	VV8SN	INTRINSIC	NO	R1, R2, R3, R4, R5, F0, F1
275	VV8S3	INTRINSIC	NO	R1, R2, R3, R4, R5, F0, F1
276	VV9DN	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
277	VV9D3	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
278	VV9SN	PROCEDURE	NO	F0, F1, F2, F3, F4, F5
279	VV9S3	PROCEDURE	NO	F0, F1, F2, F3
280	VX6D3	INTRINSIC	NO	R1, R2, R3, R4, F0, F1, F2, F3, F4, F4
281	VX6S3	INTRINSIC	NO	R1, R2, R3, R4, F0, F1, F2, F3
282	XTOC	INTRINSIC	NO	R1, R2, R3, R4, R5, R6, R7, F0, F1

6.2.5 Referencing Conventions

6.2.5.1 CSECT Names.

In order to comply with the CSECT naming standards described in the *HAL/SDL ICD*, all library code CSECTs begin with two alphabetic characters (A-Z)[1]. All library primary names and aliases are unique to 6 characters.

Whenever a data CSECT is needed for a particular library module, it is given the CSECT name #Lnnnnnn, where nnnnnn is the first 6 characters of the primary entry name.

6.2.5.2 ZCONs.

For each primary entry point and alternate entry point in the runtime library, a member exists in a separate ZCON library. The members in the ZCON library contain address constants which refer to the actual entry points. Thus, for the library routine named SIN which has an entry point named COS, there are two members in the ZCON library named #QSIN and #QCOS. These #Q modules contain references to the respective entry points. The individual ZCONs in the ZCON library are created by assembly code like the following:

1. Sector 0 routines are an exception: their CSECT names begin with #0. This is to conform to link editor conventions for routines which must reside in sector 0. Sector 0 routines are identified in the list in Section 6.2.4.3 and in the boxed area of the individual library routine description.

```
#QSIN     CSECT
          DC Z(SIN,,X'E')
          EXTRN SIN
          END
```

Some library routines make reference to other library routines via the ACALL macro (see Section 6.2.7). The ACALL macro references a library routine via a ZCON as is done when compiler-emitted code references a library routine.

6.2.6 Coding Structure

The following outline represents the standard coding structure of all library members.

1 TITLE
2 WORKAREA macro definition used only if additional stack storage was needed
3 AMAIN
4 * Comment card describing the function of the primary entry point
5 INPUT
6 OUTPUT
7 Body of executable code including use of WORK, AERROR, AEXIT macros where needed and alternate entry points defined using the AENTRY macro, function comment card, and INPUT and OUTPUT macros in the same manner as the primary entry point.
8 DC constant area addressed via PC relative mode
9 ADATA, followed by a DC constant area addressed via base and displacement mode. Used only if constants need to be indexed.
10 ACLOSE

6.2.7 The Macro Library

To standardize interface conventions, automate production of commonly used code sequences, and impose a structure to the runtime library, a series of macros are used. This section describes the function, use, and expansion of these macros. Lower case letters are used to indicate variable fields. Square brackets [ ] indicate optional fields, braces { } indicate a choice of required fields.

AMAIN

```
name AMAIN  [ INTSIC  = { YES      } ]
            [             INTERNAL   ]
            [                        ]
            [ ACALL   = YES          ]
            [ SECTOR  = 0            ]
```

Function:

Defines "name" as the primary entry point of a routine.

INTSIC=YES:

Defines the routine (and any entry points) as an intrinsic. If the INTSIC operand is omitted, the routine is defined as a procedure.

INTSIC=INTERNAL:

Defines an intrinsic which is called only by other routines in the library. At present, this is only GTBYTE and STBYTE.

ACALL=YES:

(Valid only for procedure routines) Allows use of the ACALL macro within the routine (See ACALL description).

SECTOR=0:

Defines the routine (intrinsic or procedure) as a Sector 0 routine.

Expansion:

The macro first defines the primary entry "name" (the AMAIN label) as the CSECT name, unless SECTOR=0 was specified. In the latter case, the CSECT name is generated by prefixing "name" with #0, and the primary entry "name" is defined using the DS and ENTRY statements. The options selected via the AMAIN operands are saved in global SETB variables for testing by the other macros. If either INTSIC option was selected, the macro ends. Otherwise, a procedure is being defined, so the STACK DSECT is generated.

The STACK DSECT consists of a standard 18 halfword area, including symbols for the saved copies of the fixed point register parameters (ARG2, ARG4, ARG5, ARG6, ARG7), followed by the WORKAREA macro. The WORKAREA macro is the means by which additional storage beyond the standard stack of 18 halfwords may be defined. If such storage is needed a local WORKAREA macro must have been defined earlier in the source which contains the appropriate DS assembler statements. These statements are thus incorporated as the remainder of the STACK DSECT. If additional storage is not needed, the local WORKAREA macro is not defined. As a result, the system WORKAREA macro is invoked, which does not define any storage, leaving the STACK DSECT at its standard length. The system WORKAREA macro also sets a global SETB variable, which is tested later by the AMAIN macro to determine if the stack is standard or augmented. The STACK DSECT is then terminated by resuming the original CSECT. The STACK DSECT is defined in this sequence so that the assembler will output the SYM records in the order expected by the link editor's stack size algorithm. A USING statement is generated to give addressability to the stack area. Finally, the executable code of the entry prologue is generated. For PASS, this consists of an NIST instruction to zero the 10th halfword of the new stack frame, establishing a null ON ERROR environment. The Backup Operating System (BOS) does not require NIST, therefore, BFS does not use NIST. In addition, if both ACALL=YES is specified and a local WORKAREA provided, the default stack size of 18 set up by the SCAL microcode will be insufficient, so an IAL to set up the new stack size is generated.

AENTRY

name AENTRY

Function:

Defines "name" as a secondary entry point.

Expansion:

"name" is externally defined using the DS and ENTRY statements. If the routine was defined as an intrinsic, the macro ends. Otherwise, the executable code of the entry prologue is generated in the same manner as the AMAIN macro.

AEXIT

```
                 ┌         ┌ KEEP  ┐ ┐
 name AEXIT      │ CC =   <  (rx)   >  │
                 │         │ EQ    │ │
                 │         └ NE    ┘ │
                 │ COND = code       │
                 └                   ┘
```

Function:

Cause return of control from a procedure or intrinsic routine.

CC:

Used to pass a condition code back to the caller. It can be used only if OUTPUT CC was specified (See OUTPUT macro).

Valid for Intrinsics Only:

CC=KEEP:

Passes back the condition code as is.

CC=(rx):

Passes back the condition code generated by a LR rx, rx.

Valid for Procedures Only:

CC=EQ:

Passes back an equal (B'00') condition code.

CC=NE:

Passes back a not equal (B'11') condition code.

Note: The CC= operand is used in the following 8 routines:

`CPR, CPRA, CSTR, CSTRUCT, VV8DN, VV8D3, VV8SN, and VV8S3.`

COND=code:

Used to do a conditional return, i.e. based on the current condition code. Valid for procedures only. "Code" is either a number used as the mask on a BC opcode, or a letter or letter pair representing the mask in the extended BC mnemonic op codes (E, Z, NE, NZ, H, O, L, M, HE, LE, NL, NM, NH, NO). This operand may be used to improve the efficiency of some routines. If used, be sure valid executable code follows it, so the fall through case is valid.

Expansion:

The code generated by the AEXIT macro depends primarily on whether the routine is an intrinsic or procedure, and secondarily on what operands were supplied, and, in the case of intrinsics, what fixed point registers were used. The expansions for intrinsics and procedures are described separately.

Intrinsics:

If register(s) R1 and/or R3 have been defined (see INPUT, OUTPUT, and WORK macros), it is assumed they have been modified and must be restored from the stack, since they are the addressing registers for compiled code. This is done via the appropriate LH instruction(s), or IHL and SLL instructions if CC=KEEP was specified, since LH would destroy the existing condition code. If CC=(rx) was specified, a LR rx,rx is generated to set the condition code. Finally, a BCRE or BCR is generated to cause a return to the caller. A BCR is generated if SECTOR=0 or INTSIC=INTERNAL was specified on the AMAIN macro.

Procedures:

If CC=EQ or CC=NE was specified, the condition code bits in the return PSW in the stack are zeroed or set via the ZB or SB instruction. Then, an SRET instruction is generated with a mask of 7 if the COND operand was omitted or the appropriate mask if it was supplied.

I2DEDR

name I2DEDR dpscalar1, dpscalar2, dpscalar3, dpscalar4

Function:

I2DEDR was substituted for DEDR in DMOD in order to avoid incorrect results caused by some inputs. See CR11164 and DR106660.

IBMCEDR

name IBMCEDR dpscalar1, dpscalar2

Function:

IBMCEDR was substituted for CEDR in DMDVAL and DMOD in order to avoid incorrect results caused by an incorrectly set condition code. See CR11163 and DR106644.

INPUT

$$\texttt{INPUT} \quad \left\{\begin{array}{l}\texttt{register spec type comments} \\ \texttt{NONE}\end{array}\right\}$$

Function:

Defines input interface of primary or alternate entry point and symbolic names for the register(s).

Register Spec:

One of R1, R2, R3, R4, R5, R6, R7, F0, F1, F2, F3, F4, F5, F6, or F7. If there is no input (RANDOM, RANDG only), code NONE. If there is more than one, use continuation lines for each subsequent one (see examples).

Type Comments:

type	precision	units
SCALAR	SINGLE/DOUBLE	RADIANS
MATRIX(3,3)		
MATRIX(N,N)		
VECTOR(3)		
VECTOR(N)		
INTEGER(N)		
CHARACTER		

Examples:

```
                      col. 16                                  col. 72
                         ↓                                        ↓
(1)     INPUT      F0    SCALAR      SINGLE     RADIANS     X
(2)     INPUT      R2,   VECTOR(N)   DOUBLE                 X
                   R3,   VECTOR(N)   DOUBLE                 X
                   R5    INTEGER(N)  SINGLE
```

Note: R1 and R3 are illegal inputs for procedure routines and R4 is illegal for intrinsic routines.

Expansion:

For each register spec supplied, the macro checks for a valid register symbolic, or for the special case of NONE. If the symbolic register name has not been previously defined, an EQU statement is generated to define it. The macro also tests for the illegal use of R1 or R3 for a procedure parameter and R4 for an intrinsic. A global arrayed SETB variable is set, which in conjunction with the AMAIN, AENTRY, and ACLOSE macros, will guarantee that an INPUT macro has been supplied for each entry point (see ACLOSE macro).

OUTPUT

```
             ⎧register spec type comments⎫
OUTPUT       ⎨NONE                       ⎬
             ⎩CC                         ⎭
```

Function:

Defines output interface of primary or alternate entry point.

Operand form is identical to that of INPUT macro, with the addition of CC as a possibility. This indicates that the condition code is the output of the routine. If CC is specified, the CC= option of the AEXIT macro must be used.

Expansion:

Same as for INPUT macro, except for special processing for the CC operand. If CC is supplied, a global SETB variable is set which is tested by the AEXIT macro for consistency with its CC operand.

WORK

WORK {register spec}

Function:

Defines work registers.

Expansion:

Similar to INPUT and OUTPUT, except that this macro is required only if additional register symbols need to be defined.

ABAL

ABAL name

Function:

Calls the intrinsic routine "name", valid only in a procedure routine.

Expansion:

When the runtime library routine that uses ABAL is compiled and the routine "name" is in sector 0 then ABAL generates a BAL 4, name. If the routine "name" is not in sector 0, then ABAL generates object code to call the intrinsic routine "name" indirectly. An EXTRN statement is also generated if "name" has not been previously defined.

ACALL

ACALL name

Function:

Calls the procedure routine "name", valid only in a procedure routine defined with ACALL=YES option.

Expansion:

ACALL generates object code to call the procedure routine "name" indirectly. An EXTRN statement is also generated if "name" has not been previously defined.

AERROR

AERROR number cause comment

Function:

Generates a send error SVC instruction to signal a run time error to the FCOS.

Number:

The error number.

Cause Comment:

Brief description of the cause of the error.

Expansion:

This macro accumulates, in GBLA variables, all errors sent within one assembly. It also checks to see that the error number indicates as an argument to AERROR is less than a maximum value. The actual code emitted is an SVC in which the operand is the label of an SVC parameter list to be emitted by the ADATA or ACLOSE macro via the ERRPARMS macro. If any error is sent more than once in an assembly, AERROR insures that only one SVC parameter list for that error is used.

ADATA

Function:

Defines the start of a separate data CSECT for indexable constant data.

Expansion:

A CSECT is created with the name #Lnnnnnn where nnnnnn is the first 6 characters of the primary CSECT name defined by the AMAIN macro. The ADATA macro ends leaving the data CSECT in effect so that any user-defined data following the macro call will be part of the data CSECT. The ERRPARMS macro is invoked so that any possible AERROR SVC parameter lists will appear before the indexed data. This is necessary so that the assembler will use the direct addressing mode instead of base and displacement.

ACLOSE

ACLOSE

Function:

Terminates the assembly.

Expansion:

The macro first invokes the ERRPARMS macro to create the AERROR SVC parameter lists. (See ERRPARMS macro). It then checks via arrayed global SETB variables if INPUT and OUTPUT macros were supplied for each entry point. Finally, it generates an END assembler statement, terminating the assembly.

ERRPARMS

ERRPARMS

Function:

Generates SVC parameter lists for the AERROR macro.

Expansion:

This macro is invoked by the ADATA and ACLOSE macro. It first tests a global SETB variable to see if it has already been invoked, in which case the macro does nothing. Otherwise, it generates a CSECT statement to define the data CSECT (FCOS parameter lists must reside in the data sector). The CSECT name is #Lname, where "name" is the primary entry name. The parameter lists are generated by looping through arrayed global SETA variables in which the AERROR macro saved the unique error numbers. ERRPARMS is invoked by the ADATA macro because the parameter lists must be before any indexed data following the optional ADATA macro. It is invoked by the ACLOSE macro in case the ADATA macro is not used.

WORKAREA

WORKAREA

Function:

An automatically invoked, user-created macro used to define extensions of the stack area for temporary reentrant storage. The WORKAREA macro is invoked by the AMAIN macro in procedure routines. A system supplied default is invoked in the absence of a user-created macro.

Expansion:

The system WORKAREA macro merely sets a global SETB variable which is tested by the AMAIN macro to determine whether the system or user macro is being expanded.

NOTE: Listings of the members of the MACRO library have been deleted from this document.

6.2.8 Precision Requirements

Single precision runtime library routines are required to return results that are accurate to 6 significant decimal digits. Double precision routines are required to return results that are accurate to 8 significant decimal digits.

Exceptions to this requirement are documented in the "Comments" section of the appropriate runtime library routine descriptions (Chapter 6.3).

6.2.9 Usage Restrictions

Several runtime library routines are not currently used by PASS or BFS FSW. Therefore, SSCR 11053 (Restrict Runtime Library Use) was written to require the compiler to prohibit access by the user to these routines. A mechanism for prohibiting access shall be implemented so that a new compiler release is not required should the set of supported routines change.

An asterisk (*) in the VERIFIED column indicates a routine that has been verified but its usage is still restricted by the compiler with an XS3 warning message. The routine will be unrestricted in a future compiler release.

Some of the routines are secondary entry points within another routine. These are identifiable in the table below by giving the primary entry point's name in the "Alias Of" column.

Since June 1989, the RTL routines identified as Unverified have not been audited for flight issues. Therefore, if these routines are ever used by the FSW, they should be audited to prevent possible FSW execution errors.

MEMBER NAME	ALIAS OF	VERIFIED
ACOS		YES
ACOSH		NO
ASIN	ACOS	YES
ASINH		NO
ATAN	EATAN2	YES
ATANH		NO
BIN	HIN	NO
BOUT	IOINIT	NO
BTOC		NO
CAS	CASV	YES
CASP	CASPV	NO
CASPV		YES
CASR	CASRV	YES
CASRP	CASRPV	NO
CASRPV		NO
CASRV		NO
CASV		YES
CAT	CATV	NO
CATV		YES
CEIL	ROUND	YES
CIN		NO
CINDEX		NO
CINP		NO
CLJSTV		NO
COLUMN	IOINIT	NO
COS	SNCS	YES
COSH	SINH	NO
COUT	COUTP	NO
COUTP		NO
CPAS		YES
CPASP		YES
CPASR		NO
CPASRP		NO
CPR		YES
CPRA		NO
CPRC	CPR	NO
CPSLD	CSLD	NO
CPSLDP	CSLD	NO
CPSST	CSLD	NO

MEMBER NAME	ALIAS OF	VERIFIED
CPSSTP	CSLD	NO
CRJSTV		NO
CSHAPQ		NO
CSLD		NO
CSLDP	CSLD	NO
CSST	CSLD	NO
CSSTP	CSLD	NO
CSTR		NO
CSTRUC		YES
CTOB		NO
CTOD	CTOE	NO
CTOE		NO
CTOH	CTOI	NO
CTOI		NO
CTOK	CTOI	NO
CTOO	CTOX	NO
CTOX		NO
CTRIMV		NO
DACOS		YES
DACOSH		NO
DASIN	DACOS	YES
DASINH		NO
DATAN	DATAN2	YES
DATAN2		YES
DATANH		NO
DCEIL	ROUND	YES
DCOS	DSNCS	YES
DCOSH	DSINH	NO
DEXP		YES
DFLOOR	ROUND	YES
DIN	HIN	NO
DLOG		YES
DMAX		NO
DMDVAL		YES
DMIN		NO
DMOD		YES
DOUT	IOINIT	NO
DPROD		NO
DPWRD		YES

MEMBER NAME	ALIAS OF	VERIFIED
DPWRH	DPWRI	YES
DPWRI		NO
DROUND	ROUND	YES
DSIN	DSNCS	YES
DSINH		NO
DSLD		NO
DSNCS		YES
DSQRT		YES
DSST		NO
DSUM		NO
DTAN		YES
DTANH		NO
DTOC	ETOC	NO
DTOH	ETOH	YES
DTOI	ROUND	YES
DTRUNC	ROUND	YES
EATAN2		YES
EIN	HIN	NO
EMAX		YES
EMIN		YES
EMOD		YES
EOUT	IOINIT	NO
EPROD		NO
EPWRE		YES
EPWRH	EPWRI	YES
EPWRI		NO
ESUM		NO
ETOC		NO
ETOH		YES
ETOI	ROUND	YES
EXP		YES
FLOOR	ROUND	NO
GTBYTE		YES
HIN		NO
HMAX		YES
HMIN		YES
HMOD	IMOD	YES
HOUT	IOINIT	NO
HPROD		NO

MEMBER NAME	ALIAS OF	VERIFIED
HPWRH	IPWRI	NO
HREM	IREM	YES
HSUM		YES
HTOC	ITOC	YES
IIN	HIN	NO
IMAX		NO
IMIN		NO
IMOD		YES
INTRAP	IOINIT	NO
IOINIT		NO
IOUT	IOINIT	NO
IPROD		NO
IPWRH	IPWRI	NO
IPWRI		NO
IREM		YES
ISUM		NO
ITOC		NO
ITOD		YES
ITOE		YES
KTOC		NO
LINE	IOINIT	NO
LOG		YES
MM0DNP		YES
MM0SNP		NO
MM11D3		YES
MM11DN		YES
MM11S3		YES
MM11SN		NO
MM12D3		YES
MM12DN		NO
MM12S3		NO
MM12SN		NO
MM13D3		NO
MM13DN		NO
MM13S3		YES
MM13SN		NO
MM14D3		YES
MM14DN		NO
MM14S3		NO

MEMBER NAME	ALIAS OF	VERIFIED
MM14SN		NO
MM15DN		YES
MM15SN		NO
MM17D3		NO
MM17DN	MM17D3	NO
MM17S3		NO
MM17SN	MM17S3	NO
MM1DNP		YES
MM1SNP		NO
MM1TNP		NO
MM1WNP		NO
MM6D3		YES
MM6DN		YES
MM6S3		YES
MM6SN		NO
MMRDNP		NO
MMRSNP		NO
MMWDNP		NO
MMWSNP		NO
MR0DNP		NO
MR0SNP		NO
MR1DNP		NO
MR1SNP		NO
MR1TNP		NO
MR1WNP		NO
MSTR		YES
MV6D3		YES
MV6DN		YES
MV6S3		YES
MV6SN		NO
OTOC	XTOC	NO
OUTER1	IOINIT	NO
PAGE	IOINIT	NO
QSHAPQ		NO
RANDG	RANDOM	NO
RANDOM		NO
ROUND		YES
SIN	SNCS	YES
SINH		NO

MEMBER NAME	ALIAS OF	VERIFIED
SKIP	IOINIT	NO
SNCS		YES
SQRT		YES
STBYTE		YES
TAB	IOINIT	NO
TAN		YES
TANH		NO
TRUNC	ROUND	YES
VM6D3		YES
VM6DN		YES
VM6S3		YES
VM6SN		NO
VO6D3		YES
VO6DN		YES
VO6S3		YES
VO6SN		NO
VR0DN		NO
VR0DNP		NO
VR0SN		NO
VR0SNP		NO
VR1DN		NO
VR1DNP		NO
VR1SN		YES
VR1SNP		NO
VR1TN		NO
VR1TNP		NO
VR1WN		NO
VR1WNP		NO
VV0DN		YES
VV0DNP		YES
VV0SN		YES
VV0SNP		NO
VV10D3		YES
VV10DN	VV10D3	NO
VV10S3		YES
VV10SN	VV10S3	NO
VV1D3		YES
VV1D3P		YES
VV1DN		YES

MEMBER NAME	ALIAS OF	VERIFIED
VV1DNP	VV1D3P	YES
VV1S3		YES
VV1S3P		YES
VV1SN		YES
VV1SNP	VV1S3P	NO
VV1T3		YES
VV1T3P		YES
VV1TN		YES
VV1TNP	VV1T3P	NO
VV1W3		YES
VV1W3P		NO
VV1WN		YES
VV1WNP	VV1W3P	NO
VV2D3		YES
VV2DN		NO
VV2S3		YES
VV2SN		NO
VV3D3		YES
VV3DN		YES
VV3S3		YES
VV3SN		NO
VV4D3		YES
VV4DN		YES
VV4S3		YES
VV4SN		NO
VV5D3		YES
VV5DN		NO
VV5S3		YES
VV5SN		NO
VV6D3		YES
VV6DN		YES
VV6S3		YES
VV6SN		YES
VV7D3		YES
VV7DN		NO
VV7S3		YES
VV7SN		NO
VV8D3		NO
VV8DN	VV8D3	NO

MEMBER NAME	ALIAS OF	VERIFIED
VV8S3		YES
VV8SN	VV8S3	NO
VV9D3	VV10D3	YES
VV9DN	VV10D3	NO
VV9S3		YES
VV9SN	VV10S3	YES
VX6D3		YES
VX6S3		YES
XTOC		NO

6.3 Library Routine Descriptions

This section contains descriptive material for all routines in the HAL/S-FC runtime library. The routines have been grouped into seven categories. The routines within each category are described in one sub section as follows:

6.3.1 Arithmetic
6.3.2 Algebraic
6.3.3 Vector/Matrix
6.3.4 Character
6.3.5 Array Functions
6.3.6 Miscellaneous
6.3.7 Remote Operations

This documentation is based upon the "load module" as a basic unit. A load module is the entity created by a single invocation of the AP-101 linkage editor. It has a primary member name and may have up to 16 alias names. The primary and alias names indicate entry points to the module.

For each load module in the runtime library, an LRD form will be found in the succeeding sections. The basic LRD form is shown in Figure 6-1. The circled numbers in the figure are explained below.

- 0 - The boxed area of the form (1 - 7 below) contains information relating to qualities and attributes of the load module apart from any of its entry points.
- 1 - In the upper right portion of every routine or entry point description, the name of the primary entry point will be seen. This serves as a quick reference aid in locating the documentation for a load module.
- 2 - Source Member Name - The name of the member in the assembler language source PDS of the library. This name is always the same as the primary entry point name.

③ - Size of Code Area - Each library module contains one code CSECT, regardless of the number of entry points. This number is the count of halfwords of code that would be used if the module were loaded. A module will be loaded if any one of its entry points is referenced.

④ - Stack requirement - If a module is not an intrinsic (see ±), it will have a requirement for runtime stack space. The minimum required will be one standard stack frame (18 Hw). The number listed on the form indicates the module's total stack requirement. If the module is an intrinsic, zero will be indicated. Individual entry points in one module cannot have different stack requirements. Therefore, the stack requirement is an attribute of the module.

⑤ - Data CSECT size - If the module contains a #L CSECT, its size is indicated. Otherwise, a zero is indicated. This number shows the number of halfwords of data area that will be used if the module is loaded.

⑥ - Intrinsic/Procedure - The appropriate box is marked. Entry points in a module are either all intrinsic or all procedure, hence this is a quality of the module. Sector 0 routines are noted here.

⑦ - Other modules referenced - A list of other load modules referenced in EXTRN statements by this load module. If this module is loaded, the indicated modules will also be loaded.

⑧ - Entry point descriptions - Following the aggregate attributes of the module in 0-7 above, the descriptions of specific entry points follow.

⑨ - Primary Entry Name - The name of the code CSECT in the module and the primary entry for the module in the library load module PDS.

⑩ - Function - A brief prose description of what this entry point does.

⑪ - Invoked By - Entry points may be referenced directly from compiler-emitted code, from other library modules, or both. The appropriate boxes are marked. If the upper box is marked, an example of a HAL/S construct which results in reference to the entry point is shown. If the lower box is marked, the names of other modules which refer to this entry point are listed. If any of the other modules listed here are loaded, this module will also be brought in.

⑫ - Execution Time - The time, in microseconds, needed to perform this entry point's function. These times are obtained from examinations of trace listings of simulations of the execution of the particular library routine or entry point on Version 11.3 of the GPC simulator in detailed timing mode. Times include times for referenced routines unless specifically stated.

- ⑬ - Input Arguments - The data that the entry point receives as input is listed. "Type" indicates the nature of the data (integer, scalar, etc.). "Precision", where applicable, is generally SP for single precision and DP for double precision. "How Passed" indicates the method of communication of the data. In the case of DP scalar arguments, this field may indicate the first floating point register of an even/odd pair. "Units", when applicable, specifies the units presumed for an argument.
- ⑭ - Output Results - The data that is considered the "answer" from the entry point. The fields are used in the same way as in ⑬.
- ⑮ - Errors Detected - If invocation of this entry point can result in a Send Error SVC being executed, the error #, cause, and standard fixup for all such errors are indicated.
- ⑯ - Comments - Any special behavior of this entry point or notes to users are entered here.
- ⑰ - Algorithm - The steps taken by the entry point to produce its results are shown. When appropriate, references are made to other entry point descriptions for further documentation.

In addition to the basic LRD form of Figure 6-1, which documents module attributes and the primary entry point, an extension LRD form is used to document additional alias entry points within a module. The extension LRD is shown in Figure 6-2. The circled numbers are explained below:

- ⑱ - The primary entry name of the module is displayed. This is the same name as is displayed in the basic LRD form ① to which this extension form is appended.
- ⑲ - Secondary Entry Name - The name of the secondary entry point being documented.
- ⑳ - The remainder of the extension form is identical to the primary entry point description entries ⑩ through ⑰, and describe the function and interface to this entry.

⓪

①

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

② Source Member Name: ________ ③ Size of Code Area: _____ Hw

④ Stack Requirement:_____ Hw ⑤ Data CSECT Size: _____ Hw

☐ Intrinsic ⑥ ☐ Procedure

⑦ Other Library Modules Referenced:

⑧ ENTRY POINT DESCRIPTIONS

⑨ Primary Entry Name:

⑩ Function:

⑪ Invoked By:

☐ Compiler emitted code for HAL/S construct of the form:

☐ Other Library Modules:

⑫ Execution time (microseconds):

⑬ Input Arguments:

Type	Precision	How Passed	Units

⑭ Output Results:

Type	Precision	How Passed	Units

⑮ Errors Detected:

Error #	Cause	Fixup

⑯ Comments:

⑰ Algorithm:

Figure 6-1 Basic LRD Form

18

19 Secondary Entry Name

20 Function:

Invoked By:

☐ Compiler emitted code for HAL/S construct of the form:

☐ Other Library Modules

Input Arguments:

Type	Precision	How Passed	Units

Output Results:

Type	Precision	How Passed	Units

Errors Detected:

Error #	Cause	Fixup

Comments:

Algorithm:

Figure 6-2 Extension LRD Form

The following table shows the routines which are assigned to each group. The table contains a list of primary and secondary entry points with each secondary indented under its primary entry. With each primary entry point, basic descriptive information is shown along with the sizes of the CSECTs in the module and the module's stack requirement. A final entry shows the timing information for the entry point. Secondary entry points have only the descriptive information and the timing for the entry. In cases where the timing information is too involved to be listed in the space available, the notice "See LRD" indicates that the detailed write-up of the module (on an LRD form in the proper subsection) should be referenced. In all cases, information in the table is taken from the LRDs and further details on the routines' performance can be found in those detailed descriptions.

ARITHMETIC ROUTINES (Section 6.3.1)

ENTRY	FUNCTION	PREC.	CODE	DATA	STACK	TIME	Page
DMDVAL	MIDVAL(D,D,D)	D	20	0	18	41.4	6-36
DMOD	MOD(D,D)	D	152	4	0	74.6	6-38
EMOD	MOD(S,S)	S	52	4	0	46.6	6-40
IMOD	MOD(I,I)	I	20	2	0	29.4	6-41
HMOD	MOD(H,H)	H				29.4	6-42
IREM	REMAINDER(I,I)	I	14	2	0	27.0	6-43
HREM	REMAINDER(H,H)	H				27.0	6-44
ROUND	ROUND(S)	I	84	2	0	39.0	6-45
CEIL	CEILING(S)	I				See LRD	6-46
DCEIL	CEILING(D)	I				See LRD	6-47
DFLOOR	FLOOR(D)	I				See LRD	6-48
DROUND	ROUND(D)	I				33.8	6-49
DTOI	D $\rightarrow$I	I				33.8	6-50
DTRUNC	TRUNCATE(D)	I				28.6	6-51
ETOI	S $\rightarrow$I	I				39.0	6-52
FLOOR	FLOOR(S)	I				See LRD	6-52
TRUNC	TRUNCATE(S)	I				31.4	6-53

ALGEBRAIC ROUTINES (Section 6.3.2)

ENTRY	FUNCTION	PREC.	CODE	DATA	STACK	TIME	Page
ACOS	ARCCOS(S)	S	102	2	24	See LRD	6-54
ASIN	ARCSIN(S)	S				See LRD	6-55
ACOSH	ARCCOSH(S)	S	36	2	20	See LRD	6-57
ASINH	ARCSINH(S)	S	64	0	20	See LRD	6-58
ATANH	ARCTANH(S)	S	58	2	18	See LRD	6-59
DACOS	ARCCOS(D)	D	230	2	26	See LRD	6-60
DASIN	ARCSIN(D)	D				See LRD	6-61
DACOSH	ARCCOSH(D)	D	50	2	22	See LRD	6-63
DASINH	ARCSINH(D)	D	94	0	22	See LRD	6-64
DATANH	ARCTANH(D)	D	132	2	26	See LRD	6-65
DATAN2	ARCTAN2(D,D)	D	194	26	18	248.4	6-66
DATAN	ARCTAN(D)	D				237.3	6-68
DEXP	EXP(D)	D	154	66	18	290.5	6-69
DLOG	LOG(D)	D	184	2	30	282.2	6-71
DPWRD	D**D	D	40	4	22	See LRD	6-73
DPWRI	D**I	D	40	2	18	See LRD	6-74
DPWRH	D**H	D				See LRD	6-76
DSINH	SINH(D)	D	130	2	22	See LRD	6-77
DCOSH	COSH(D)	D				422.6	6-79
DSNCS	S**I	D	54	2	26	See LRD	6-80
DCOS	COS(D)	D				261.8-264.2	6-83
DSIN	SIN(D)	D	102	62	20	267.0	6-84
DSQRT	SQRT(D)	D	70	2	26	345.2	6-85
DTAN	TAN(D)	D	196	4	38	302.2	6-88
DTANH	TANH(D)	D	94	0	22	See LRD	6-91
EATAN2	ARCTAN2(S,S)	S	132	10	18	120.0	6-93
ATAN	ARCTAN(S)	S				116.5	6-96
EPWRE	S**S	S	32	4	22	See LRD	6-97
EPWRI	S**I	S	38	2	18	See LRD	6-98
EPWRH	S**H	S				See LRD	6-100
EXP	EXP(S)	S	108	2	18	141.8	6-101
IPWRI	I**I	I	46	2	18	See LRD	6-103
HPWRH	H**H	H				See LRD	6-106
IPWRH	I**H	I				See LRD	6-105
LOG	LOG(S)	S	90	2	18	140.5	6-107
SINH	SINH(S)	S	80	2	18	See LRD	6-109
COSH	COSH(S)	S				228.9	6-111
SNCS	SIN(S),COS(S)	S	122	28	0	See LRD	6-112
COS	COS(S)	S				122.1-123.1	6-115
SIN	SIN(S)	S	70	30	0	123.6-124.5	6-116
SQRT	SQRT(S)	S	48	14	0	88.3	6-117
TAN	TAN(S)	S	112	4	20	164.0	6-119
TANH	TANH(S)	S	56	0	18	See LRD	6-121

VECTOR/MATRIX ROUTINES (Section 6.3.3)

ENTRY	FUNCTION	SIZE	PREC	CODE	DATA	STACK	TIME	PAGE
MM0DNP	Scalar to Partitioned Matrix Move	n,m	D	12	0	0	6.8+n(4.0+8.0m)	6-122
MM0SNP	"	n,m	S	10	0	0	6.4+n(4.4+6.4m)	6-124
MM1DNP	Partitioned Matrix Move	n,m	D	18	0	0	10.8+n(5.4+12.2m)	6-125
MM1SNP	"	n,m	S	16	0	0	10.8+n(5.4+9.4m)	6-127
MM1TNP	"	n,m	D-S	16	0	0	10.4+n(5.8+10.6)	6-129
MM1WNP	"	n,m	S-D	18	0	0	13.6+n(5.0+11.0m)	6-131
MM6DN	Matrix Multiply	(m,n),(n,λ)	D	42	0	0	22.2+m(10.8+λ(21.2+27.n))	6-133
MM6D3	"	(3,3),(3,3)	D	32	0	0	671.6	6-134
MM6SN	"	(m,n),(n,λ)	S	40	0	0	22.2+m(10.8+λ(20.2+18.0n))	6-135
MM6S3	"	(3,3),(3,3)	S	24	0	0	409.6	6-136
MM11DN	Matrix Transpose	n,m	D	16	0	0	8.0+m(5.8+12.2n)	6-137
MM11D3	"	3,3	D	22	0	0	93.6	6-138
MM11SN	"	m,n	S	16	0	0	8.4+m(5.8+9.4n)	6-139
MM11S3	"	3,3	S	18	0	0	71.8	6-140
MM12DN	Matrix Determinant	n,n	D	150	0	22	See LRD	6-141
MM12D3	"	3,3	D	44	0	18	229.6	6-143
MM12SN	"	n,n	S	138	0	20	See LRD	6-144
MM12S3	"	3,3	S	26	0	18	116.0	6-146
MM13DN	Matrix Trace	n,n	D	10	0	0	12.0+10.2n	6-147
MM13D3	"	3,3	D	8	0	0	19.8	6-148
MM13SN	"	n,n	S	8	0	0	8.8+6.2n	6-149
MM13S3	"	3,3	S	4	0	0	9.8	6-150
MM14DN	Matrix Inverse	n,n	D	258	2	20	$63.0+129.5n+43.0n^2+65.4n^3$	6-151
MM14D3	"	3,3	D	128	2	18	795.4	6-152
MM14SN	"	n,n	S	242	2	20	$52.0+39.2n+10.5n^2+54.6n^3$	6-153
MM14S3	"	3,3	S	80	2	18	458.8	6-155
MM15DN	Identity Matrix	n,n	D	18	0	0	$15.6+5.0n+11.2n^2$	6-156
MM15SN	"	n,n	S	14	0	0	$10.0+5.2n+9.6n^2$	6-157
MM17D3	Matrix to a Power	3,3	D	86	0	20	See LRD	6-158
MM17DN	"	n,n	D				See LRD	6-159
MM17S3	"	3,3	S	78	0	20	See LRD	6-160
MM17SN	"	n,n	S				See LRD	6-161
MV6DN	Matrix times Vector	(m,n),n	D	24	0	0	12.0+m(19.3+26.0n)	6-162
MV6D3	"	(3,3),3	D	22	0	0	304.4	6-163
MV6SN	"	(m,n),n	S	18	0	0	11.2+m(11.0+18.4n)	6-164
MV6S3	"	(3,3),3	S	20	0	0	137.6	6-165
VM6DN	Vector times Matrix	n,(n,m)	D	26	0	0	23.2+m(23.2+27.6n)	6-166
VM6D3	"	3,(3,3)	D	24	0	0	227.8	6-167
VM6SN	"	n,(n,m)	S	22	0	0	12.4+m(19.2+18.2n)	6-168
VM6S3	"	3,(3,3)	S	16	0	0	141.2	6-169
VO6DN	Vector Outer Product	n,m	D	20	0	0	12.8+n(5.8+24.4m)	6-170
VO6D3	"	3,3	D	22	0	0	251.0	6-171
VO6SN	"	n,m	S	20	0	0	14.2+n(5.8+14.4m)	6-172
VO6S3	"	3,3	S	20	0	0	160.6	6-173
VV0DN	Scalar to Vector Move	n	D	6	0	0	7.0+5.1n	6-174
VV0DNP	Scalar to Column Vector Move	n	D	6	0	0	7.0+7.2n	6-175
VV0SN	Scalar to Vector Move	n	S	6	0	0	7.0+5.6n	6-176
VV0SNP	Scalar to Column Vector Move	n	S	6	0	0	7.0+6.0n	6-177
VV1DN	Vector Move	n	D	8	0	0	4.2+10.2n	6-178
VV1D3	"	3	D	14	0	0	25.2	6-179
VV1D3P	Column Vector Move	3	D	18	0	0	See LRD	6-180
VV1DNP	"	n	D				See LRD	6-181

VECTOR/MATRIX ROUTINES (Section 6.3.3)

ENTRY	FUNCTION	SIZE	PREC	CODE	DATA	STACK	TIME	PAGE
VV1SN	Vector Move	n	S	8	0	0	4.2+7.8n	6-182
VV1S3	"	3	S	8	0	0	16.8	6-183
VV1S3P	Column Vector Move	3	S	14	0	0	See LRD	6-184
VV1SNP	"	n	S				See LRD	6-185
VV1TN	Vector Move	n	D-S	8	0	0	4.2+9.0n	6-186
VV1T3	Vector Move	3	D-S	12	0	0	21.2	6-187
VV1T3P	Column Vector Move	3	D-S	14	0	0	See LRD	6-188
VV1TNP	"	n	D-S				See LRD	6-189
VV1WN	Vector Move	n	S-D	10	0	0	8.4+9.0n	6-190
VV1W3	"	3	S-D	12	0	0	23.8	6-191
VV1W3P	Column Vector Move	3	S-D	18	0	0	See LRD	6-192
VV1WNP	"	n	S-D				See LRD	6-193
VV2DN	Vector Add/Matrix Add	n	D	14	0	0	8.8+20.6n	6-194
VV2D3	Vector Add	3	D	22	22	0	51.4	6-195
VV2SN	Vector Add/Matrix Add	n	S	10	0	0	8.2+13.6n	6-196
VV2S3	Vector Add	3	S	12	0	0	29.6	6-197
VV3DN	Vector Subtract/Matrix Subtract	n	D	16	0	0	6.0+22.7n	6-198
VV3D3	Vector Subtract	3	D	24	0	0	55.4	6-199
VV3SN	Vector Subtract/Matrix Subtract	n	S	10	0	0	8.4+13.6n	6-200
VV3S3	Vector Subtract	3	S	12	0	0	29.6	6-201
VV4DN	Vector or Matrix Times Scalar	n	D	8	0	0	7.0+23.4n	6-202
VV4D3	Vector Times Scalar	3	D	18	0	0	68.4	6-203
VV4SN	Vector or Matrix Times Scalar	n	S	8	0	0	7.0+14.0n	6-204
VV4S3	Vector Times Scalar	3	S	12	0	0	38.4	6-205
VV5DN	Vector or Matrix Divided by Scalar	n	D	16	2	0	37.0+24.2n	6-206
VV5D3	Vector Divided by Scalar	3	D	26	2	0	98.4	6-207
VV5SN	Vector or Matrix Divided by Scalar	n	S	14	2	0	7.2+18.0n	6-208
VV5S3	Vector Divided by Scalar	3	S	18	2	0	50.6	6-209
VV6DN	Vector Dot Product	n	D	12	0	0	16.4+25.4n	6-210
VV6D3	"	3	D	16	0	0	71.8	6-211
VV6SN	"	n	S	12	0	0	15.2+16.8n	6-212
VV6S3	"	3	S	10	0	0	41.8	6-213
VV7DN	Vector or Matrix Negate	n	D	8	0	0	7.0+11.4n	6-214
VV7D3	Vector Negate	3	D	18	0	0	32.4	6-215
VV7SN	Vector or Matrix Negate	n	S	8	0	0	7.0+9.0n	6-216
VV7S3	Vector Negate	3	S	12	0	0	23.4	6-217
VV8D3	Vector Compare	3	D				See LRD	6-218
VV8DN	Vector or Matrix Compare	n	D				See LRD	6-219
VV8S3	Vector Compare	3	S	12	0	0	See LRD	6-220
VV8SN	Vector or Matrix Compare	n	S				See LRD	6-221
VV9S3	Vector Magnitude	3	D				168.3	6-222
VV10D3	Unit Vector	3	D	56	2	20	402.7	6-223
VV9DN	Vector Magnitude	n	D				226.6+24.4n	6-225
VV9D3	"	3	D				300.2	6-224
VV10DN	Unit Vector	n	D				259.7+47.8n	6-226
VV10S3	Unit Vector	3	S	50	2	24	236.4	6-227
VV9SN	Vector Magnitude	n	S				118.9+14.0n	6-228
VV10SN	Unit Vector	n	S				130.6+32.8n	6-229
VX6D3	Vector Cross Product	3	D	36	0	0	137.6	6-230
VX6S3	"	3	S	22	0	0	78.0	6-231

CHARACTER ROUTINES (Section 6.3.4)

ENTRY	FUNCTION	CODE	DATA	STACK	TIME	PAGE
CASPV	Partitioned Assign to VAC	64	2	0	See LRD	6-232
CASP	Partitioned Assign				See LRD	6-234
CASV	Assign to VAC	28	0	0	29.2(n=0) See LRD	6-235
CAS	Assign				32.0(n=0) See LRD	6-237
CATV	Catenate into VAC	76	0	0	See LRD	6-238
CAT	Catenate into Data				See LRD	6-240
CINDEX	INDEX Function	52	0	18	See LRD	6-241
CLJSTV	LJUST	40	2	18	See LRD	6-243
CPAS	Assign to Partition	80	2	20	See LRD	6-245
CPASP	Partition Assign to Partition	16	0	146	See LRD	6-247
CPR	Compare (= or ¬=)	46	0	0	See LRD	6-248
CPRC	Compare (all relations except = and ¬=)					6-250
CPRA	Arrayed Compare	20	0	22	See LRD	6-251
CRJSTV	RJUST	46	2	18	See LRD	6-253
CTRIMV	TRIM	94	0	18	See LRD	6-255

ARRAY ROUTINES (Section 6.3.5)

ENTRY	FUNCTION	PREC	CODE	DATA	STACK	TIME	PAGE
DMAX	MAX(DA)	D	10	0	0	See LRD	6-257
DMIN	MIN(DA)	D	10	0	0	See LRD	6-258
DPROD	PROD(DA)	D	14	0	0	See LRD	6-259
DSUM	SUM(DA)	D	6	0	0	7.2+11.6n	6-260
EMAX	MAX(SA)	S	8	0	0	See LRD	6-261
EMIN	MIN(SA)	S	8	0	0	See LRD	6-262
EPROD	PROD(SA)	S	10	0	0	See LRD	6-263
ESUM	SUM(SA)	S	6	0	0	5.2+6.6n6-264	6-264
HMAX	MAX(HA)	H	8	0	0	See LRD	6-265
HMIN	MIN(HA)	H	8	0	0	See LRD	6-266
HPROD	PROD(HA)	H	12	0	0	See LRD	6-267
HSUM	SUM(HA)	H	6	0	0	4.4+5.4n	6-268
IMAX	MAX(IA)	I	8	0	0	See LRD	6-269
IMIN	MIN(IA)	I	8	0	0	See LRD	6-270
IPROD	PROD(IA)	I	22	0	0	See LRD	6-271
ISUM	SUM(IA)	I	6	0	0	4.4+5.4n	6-272

MISCELLANEOUS ROUTINES (Section 6.3.6)

ENTRY	FUNCTION	CODE	DATA	STACK	TIME	PAGE
BTOC	Bit to Character Conversion	28	0	0	161.0(16 bits)	6-273
CSHAPQ	Shaping Function	40	4	18	See LRD	6-274
CSLD	SUBBIT Load of Character	246	4	22	See LRD	6-276
CPSLD	Partitioned SUBBIT Load of Character				71.8	6-277
CPSST	Partitioned SUBBIT Store to Character				114.4	6-279
CPSLDP	Partitioned SUBBIT Load of Partitioned Character				See LRD	6-278
CPSSTP	Partitioned SUBBIT Store to Partitioned Character				See LRD	6-281
CSLDP	SUBBIT Load of Partitioned Character				See LRD	6-282
CSST	SUBBIT Store to Character				See LRD	6-283
CSSTP	SUBBIT Store to Partitioned Character				See LRD	6-284
CSTRUC	Structure Compare	12	0	0	5.4+10.4n	6-285
CTOB	Character to Bit Conversion	32	2	18	See LRD	6-286
CTOE	Character to SP Scalar Conversion	287	2	30	See LRD	6-288
CTOD	Character to DP Scalar Conversion				See LRD	6-291
CTOI	Character to DP Integer Conversion	104	2	20	See LRD	6-292
CTOH	Character to SP Integer Conversion				See LRD	6-294
CTOK	Character to Bit Conversion, DEC Radix				See LRD	6-295
CTOX	Character to Bit Conversion, HEX Radix	58	4	18	See LRD	6-296
CTOO	Character to Bit Conversion, OCT Radix				See LRD	6-298
DSLD	SUBBIT Load of DP Scalar	22	2	18	36.5	6-299
DSST	SUBBIT Store into DP Scalar	54	2	18	64.6	6-300
ETOC	SP Scalar to Character Conversion	278	64	20	336.9	6-301
DTOC	DP Scalar to Character Conversion				602.5	6-303
ETOH	SP Scalar to SP Integer Conversion	14	0	0	15.4	6-304
DTOH	DP Scalar to SP Integer Conversion				17.4	6-306
GTBYTE	Character Fetch	14	0	0	See LRD	6-307
ITOC	DP Integer to Character Conversion	104	0	28	254.6	6-308
HTOC	SP Integer to Character Conversion				189.6	6-309
ITOD	DP Integer to DP Scalar Conversion	20	0	0	15.6	6-310
ITOE	DP Integer to SP Scalar Conversion	6	0	0	12.0	6-311
KTOC	Bit to Character Conversion, DEC Radix	70	0	0	262.5(16 bits)	6-312
MSTRUC	Structure Move	8	0	0	4.2+9.4n	-
QSHAPQ	Shaping Functions	74	0	18	42.6+31.8n	6-314
RANDOM	Random Number Generator, Uniform Dist.	46	2	18	54.4	6-316
RANDG	Random Number Generator, Gaussian Dist.				575.8	6-317
STBYTE	Character Store	22	0	0	See LRD	6-318
XTOC	Bit to Character Conversion, HEX Radix	68	0	0	See LRD	6-319
OTOC	Bit to Character Conversion, OCT Radix				See LRD	6-321

REMOTE ROUTINES (Section 6.3.7)

A. CHARACTER ROUTINES

ENTRY	FUNCTION	CODE	DATA	STACK	TIME	PAGE
CASRPV	Partitioned Assign to VAC	86	2	22	See LRD	6-323
CASRP	Partition Assign				See LRD	6-325
CASRV	Assign to VAC	36	0	18	See LRD	6-326
CASR	Assign				See LRD	6-327
CPASR	Assign to Partition	132	2	24	See LRD	6-328
CPASRP	Partition Assign to Partition	16	0	146	See LRD	6-330

B. STRUCTURE ROUTINES

ENTRY	FUNCTION	CODE	DATA	STACK	TIME	PAGE
CSTR	Structure Compare	18	0	18	See LRD	6-332
MSTR	Structure Move	10	0	18	See LRD	6-343

C. VECTOR AND MATRIX ROUTINES

ENTRY	FUNCTION			CODE	DATA	STACK	TIME	PAGE
MR0DNP	Scalar to Partitioned Matrix Move	n,m	D	16	0	20	22.8+n(5.6+9.8m)	6-333
MR0SNP	"	n,m	S	16	0	20	22.8+n(5.6+8.6m)	6-334
MR1DNP	Partitioned Matrix Move	n,m	D	22	0	20	28.4+n(8.2+15.0m)	6-336
MR1SNP	"	n,m	S	22	0	22	28.4+n(8.2+12.6m)	6-338
MR1TNP	"	n,m	D-S	24	0	22	31.2+n(7.6+13.8m)	6-340
MR1WNP	"	n,m	S-D	24	0	22	32.8+n(8.2+15.8m)	6-342
VR0DN	Scalar to Vector Move	n	D	6	0	18	16.4+9.2n	6-344
VR0DNP	Scalar to Column Vector Move	n	D	10	0	18	21.2+10.0n	6-345
VR0SN	Scalar to Vector Move	n	S	6	0	18	16.4+8.0n	6-346
VR0SNP	Scalar to Column Vector Move	n	S	10	0	18	21.2+8.8n	6-347
VR1DN	Vector Move	n	D	8	0	18	16.4+15.0n	6-348
VR1DNP	Column Vector Move	n	D	20	0	18	See LRD	6-349
VR1SN	Vector Move	n	S	8	0	18	16.4+12.6n	6-350
VR1SNP	Column Vector Move	n	S	20	0	18	See LRD	6-351
VR1TN	Vector Move	n	D-S	8	0	18	16.4+13.8n	6-352
VR1TNP	Column Vector Move	n	D-S	20	0	18	See LRD	6-353
VR1WN	Vector Move	n	S-D	10	0	18	20.6+13.8n	6-354
VR1WNP	Column Vector Move	n	S-D	22	0	18	See LRD	6-355

6.3.1 Arithmetic Routine Descriptions

This subsection presents the detailed descriptions of a class of routines generally denoted as "Arithmetic". Appendix C of the HAL/S Language Specification contains a list of HAL/S functions which are implemented by the routines described here.

DMDVAL

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DMDVAL Size of Code Area 84 Hw

Stack Requirement: 18 Hw Data CSECT Size: 0 Hw

[ ] Intrinsic [x] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DMDVAL

Function: Finds mid value of three double precision scalar arguments.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`MIDVAL(A,B,C), where A, B, C are double precision scalars.`

[ ] Other Library Modules:

Execution Time (microseconds):

Input Arguments:

Type	Precision	How Passed	Units
Scalar(A)	DP	F0	-
Scalar(B)	DP	F2	-
Scalar(C)	DP	F4	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

```
IF A<=B THEN DO;
      IF A =B THEN RETURN A;
      IF B <=C THEN RETURN B;
      ELSE IF A <=C THEN RETURN C;
           ELSE RETURN A;
 END;
 ELSE DO;
       IF C <=B THEN RETURN B;
       ELSE IF C < A THEN RETURN C;
           ELSE RETURN A;
  END;
```

DMOD

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DMOD Size of Code Area 152 Hw

Stack Requirement: 0 Hw Data CSECT Size: 4 Hw

[x] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DMOD

Function: Calclates HAL/S MOD function in double precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`MOD(A,B), where at least one of A or B is a double precision scalar.`

[ ] Other Library Modules:

Execution Time (microseconds):

Input Arguments:

Type	Precision	How Passed	Units
Scalar(A)	DP	F0/F1	-
Scalar(B)	DP	F2/F3	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0/F1	-

Errors Detected:

Error #	Cause	Fixup
19	MOD domain error when B=0, A < 0	Return 0
33	$\|A/B\|$ approx. > 16^{14}	Return 0

Comments:

The precision of the result of the MOD(A,B) function is limited by the precision of the inputs A and B. Specifically, the EMOD output will be precise to six significant digits of the input of greatest magnitude. Similarly, the DMOD output will be precise to eight significant digits of the input of greatest magnitude. A fixup value of zero is returned by MOD(A,B) if:

1. It is a positive operand case and the result computed for MOD(A,B) is greater than |B|, or
2. It is a negative operand case and the result computed for MOD(A,B) is less than zero.

Warning:

The DMOD routine uses the formula MOD(A,B)=A-|B|*FLOOR(A/|B|).

The operation A-|B| * FLOOR(A/|B|) may cause an underflow when H'8D10000000000000'<u><</u>A<u><</u>H0D'10000000000000' (-2.4308653429145086E-63<u><</u>A<u><</u>2.4308653429145086E-63).

An exponent overflow can also be generated during the divide operation (I2DEDR macro for DMOD, DER for EMOD) when A and B differ in order of magnitude as described in the AP101 POO for the operation A/|B|. For overflow, A would be greater than B by approximately half of the floating point exponential range.

Registers Unsafe Across Call: R4,F0,F1,F2,F3,F4,F5,F6,F7.

Algorithm:

First check for mod domain error (B=0 and A < 0) and signal an error 19 and return a fixup value of zero. If B< >0, then take |B|. For positive A values, MOD(A,B) is computed as A-(|B|*FLOOR(A/|B|)). For positive A values, a pre-divide check is performed, and if A < |B| then return A as the answer. For negative A values, MOD(A,B) is computed as A+(|B|*FLOOR(|A|/|B|)). For negative A values, a pre-divide check is performed, and if |A|<|B| then return A+|B| as the answer. For both positive and negative A's, the FLOOR function is accomplished by adding and then subtracting a value BIGNUM (X'4E80000000000000' for DMOD and X'4680000000000000' for EMOD), which causes all the fractional part of the quotient to be lost, leaving only the integer portion. The positive and negative A parts of the algorithm then converge for pre-exit validation of the result. If the answer is negative, then add in one more |B|. If the answer is still negative, then log a GPC error and return a fixup value of zero. If the answer is positive or zero, check that it is less than |B|. If not, subtract one |B| from the answer and check it again. If the answer is still greater than or equal to |B| then log a GPC error and return a fixup value of zero.

EMOD

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: EMOD Size of Code Area 52 Hw

Stack Requirement: 0 Hw Data CSECT Size: 4 Hw

[x] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: EMOD

Function: Calculates HAL/S MOD function in single precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
MOD(A,B), where A and B are single precision
          scalars.
```

[ ] Other Library Modules:

Execution Time (microseconds):

Input Arguments:

Type	Precision	How Passed	Units
Scalar(A)	SP	F0	-
Scalar(B)	SP	F2	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
19	MOD domain error when B=0,A<0	Return 0
33	$\lvert A/B \rvert$ approx.$>16^6$	Return 0

Comments:

See DMOD.

Registers Unsafe Across Call: R4,F0,F2,F4,F5.

Algorithm:

See DMOD.

IMOD

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: IMOD Size of Code Area 152 Hw

Stack Requirement: 0 Hw Data CSECT Size: 4 Hw

[x] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: IMOD

Function: Calculates HAL/S MOD(A,B) in double precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
MOD(A,B), where A and B are both integers and at least
          A or B is a fullword integer value.
```

[ ] Other Library Modules:

Execution Time (microseconds): 29.4

Input Arguments:

Type	Precision	How Passed	Units
Integer(A)	DP	R5	-
Integer(B)	DP	R6	-

Output Results:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup
19	MOD not defined for A<0 and B=0	Return A

Comments:

MOD(A,B) is defined to be the smallest non-negative value that can be reached by starting from A and adding or subtracting |B| as often as is necessary.

Registers Unsafe Across Call: R2,R4,R5,R6,R7.

Algorithm:

If B = 0

  If A≥0 Return A

  If A<0 Error #19 generated

If B ≠ 0

  MOD(A,B) = A - [(|B|(A/|B|)]

  If this result < 0, then add |B| to this to make it positive.

For all values of A and B, the result is always non-negative.

For A≥0, MOD = REMAINDER(A,B). These equations are used because AP-101 division (scalar or integer) does not yield a remainder.

IMOD

Secondary Entry Name: HMOD

Function: Performs HAL/S MOD(A,B) where both A and B are single precision integers.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
MOD(A,B), where A and B are both integers and at least
          A or B is a fullword integer value.
```

[ ] Other Library Modules:

Execution Time (microseconds): 29.4

Input Arguments:

Type	Precision	How Passed	Units
Integer(A)	SP	R5	-
Integer(B)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Integer	SP	R5	-

Errors Detected:

Error #	Cause	Fixup
Same as IMOD		

Comments:

MOD(A,B) is defined to be the smallest non-negative value that can be reached by starting from A and adding or subtracting |B| as often as is necessary.

Registers Unsafe Across Call: R2,R4,R5,R6,R7.

Algorithm:

Same as IMOD

IREM

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: IREM Size of Code Area 16 Hw

Stack Requirement: 0 Hw Data CSECT Size: 2 Hw

[x] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: IREM

Function: Calculates integer remainder of (A,B).

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`MOD(A,B), where A and B are both single precision integers.`

[ ] Other Library Modules:

Execution Time (microseconds): 27.0

Input Arguments:

Type	Precision	How Passed	Units
Integer(A)	DP } one can be SP	R5	-
Integer(B)	DP }	R6	-

Output Results:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup
16	Zero denominator (B)	Return A

Comments:

Registers Unsafe Across Call: R2,R4,R5,R6,R7.

Algorithm:

If B=0, then error. For B ≠ 0, the remainder is found using REMAINDER(A,B) = [A - B*(A/B)]. The result can be negative.

IREM

Secondary Entry Name: HREM

Function: Calculates integer remainder of A/B.

Invoked By:

- [X] Compiler emitted code for HAL/S construct of the form: `REMAINDER(A,B)`, where A and B are both single precision integers.
- [ ] Other Library Modules:

Execution Time (microseconds): 27.0

Input Arguments:

Type	Precision	How Passed	Units
Integer(A)	SP	R5	-
Integer(B)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Integer	SP	R5	-

Errors Detected:

Error #	Cause	Fixup
16	Zero denominator (B)	Return A

Comments:

Registers Unsafe Across Call: R2,R4,R5,R6,R7.

Algorithm:

Same as IREM.

ROUND

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: ROUND Size of Code Area 80 Hw

Stack Requirement: 0 Hw Data CSECT Size: 2 Hw

[x] Intrinsic - Sector - 0 [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: ROUND

Function: Converts single precision scalar to fullword integer.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`ROUND(X), where X is a single precision scalar.`

[X] Other Library Modules:

QSHAPQ

Execution Time (microseconds): 39.0

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup
15	Scalar too large for integer conversion	Return either: Posmax = X'7FFFFFFF' or Negmax = X'80000000'

Comments:

See DROUND.

Registers Unsafe Across Call: R4,R5,F0,F1.

Algorithm:

Second register of a floating point register pair is cleared then routine merges into the double precision float-to- fix routine, DROUND.

ROUND

Secondary Entry Name: CEIL

Function: Performs HAL/S CEILING function: Returns smallest integer $\geq$ the argument.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`CEILING(X), where X is a single precision scalar.`

[ ] Other Library Modules:

Execution Time (microseconds): 31.4 if X > 0
40.8 if X < 0

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup
15	Scalar too large for integer conversion	Return either: Posmax = X'7FFFFFFF' or Negmax = X'80000000'

Comments:

See DCEIL.

Registers Unsafe Across Call: R4,R5,F0,F1.

An invalid result of 0 is returned for arguments between 0 < N < 1.0 X 16 E-14.

Algorithm:

Second register of floating point register pair is cleared, then routine merges with DCEIL.

ROUND

Secondary Entry Name: DCEIL

Function: Performs HAL/S CEILING function: Finds the smallest integer ≥ the argument.

Invoked By:

- [X] Compiler emitted code for HAL/S construct of the form:
  `CEILING(X), where X is a double precision scalar.`
- [ ] Other Library Modules:

Execution Time (microseconds): 26.6 if X ≥ 0
36.0 if X < 0

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0, F1	-

Output Results:

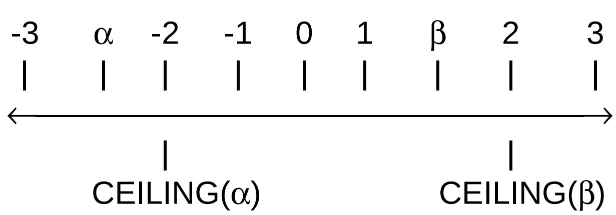
Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup
15	Scalar too large for integer conversion	Return either: Posmax = X'7FFFFFFF' or Negmax = X'80000000'

Comments:

Negative args become less negative after CEILING, positive args more positive.



Registers Unsafe Across Call: R4,R5,F0,F1.

An invalid result of 0 is returned for arguments between 0 < N < 1.0 X 16 E-14.

Algorithm:

Same as DROUND, except positive arguments are rounded up by almost 1. Negative arguments are not rounded.

ROUND

Secondary Entry Name: DFLOOR

Function: Performs HAL/S FLOOR function: Finds the largest integer $\leq$ the argument.

Invoked By:

- [X] Compiler emitted code for HAL/S construct of the form:
  `FLOOR(X), where X is a double precision scalar.`
- [ ] Other Library Modules:

Execution Time (microseconds): 27.0 if $X \geq 0$
36.4 if $X < 0$

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0, F1	-

Output Results:

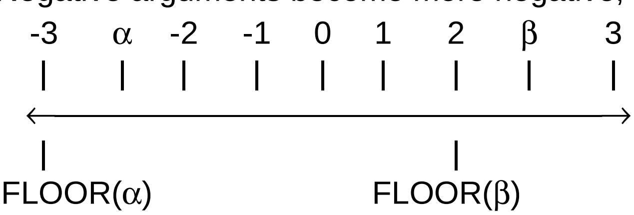
Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup
15	Scalar too large for integer conversion	Return either: Posmax = X'7FFFFFFF' or Negmax = X'80000000'

Comments:

Negative arguments become more negative, positive arguments less positive.



Registers Unsafe Across Call: R4,R5,F0,F1.

An invalid result of 0 is returned from the DFLOOR function for arguments between -1.0 X 16 E-14 < N < 0.

Algorithm:

Same as DROUND, except argument is rounded down by almost 1 (X'40FFFFFFFFFFFFFF') if negative. Positive arguments are not rounded.

ROUND

Secondary Entry Name: DROUND

Function: Converts double precision scalar to fullword integer.

Invoked By:

- [X] Compiler emitted code for HAL/S construct of the form:

  `ROUND(X), where X is a double precision scalar.`
- [ ] Other Library Modules:

Execution Time (microseconds): 33.8

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0, F1	-

Output Results:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup
15	Scalar too large for integer conversion	Return either: Posmax = X'7FFFFFFF' or Negmax = X'80000000'

Comments:

Negative arguments are converted to the next more negative integer value; positive args to the next greater positive integer value, unless the original argument is an integer (argument rounded up or down by not quite 1 before truncating decimal places). Values such as xx.5 are rounded down to xx.0 and values such as -xx.5 are rounded up to -xx.0.

Registers Unsafe Across Call: R4,R5,F0,F1.

Algorithm:

The argument is checked for negative/not negative. If the argument is negative, the value is rounded down by subtracting just under 1/2. The resulting value is then checked against MAXNEG(X'C880000000FFFFFF'). If within the legal range, the integer part of the scalar is shifted to the second register of the floating point register pair. This remaining integer value is then put in a fixed point register and complemented to leave it in the correct two's complement fixed point form. If the argument is not negative, the value is rounded up by adding almost 1/2, and the resulting value is compared to MAXPOS(X'487FFFFFFFFFFFFF'). Then, as with negative values, it is shifted to leave the integer part in floating point format and loaded into a fixed point register.

ROUND

Secondary Entry Name: DTOI
Function: Converts double precision scalar to fullword integer.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
I = D; where I is a double precision integer, and D is a
       double precision scalar.
```

[ ] Other Library Modules:

Execution Time (microseconds): 33.8

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Output Results:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup
15	Scalar too large for integer conversion	Return either: Posmax = X'7FFFFFFF' or Negmax = X'80000000'

Comments:

DTOI is identical entry point to DROUND.

Registers Unsafe Across Call: R4,R5,F0,F1.

Algorithm:

Algorithm: Same as DROUND.

ROUND

Secondary Entry Name: DTRUNC

Function: Performs HAL/S TRUNCATE function: Finds the signed value that is the largest integer ≤ absolute value of the argument.

Invoked By:

- [X] Compiler emitted code for HAL/S construct of the form:
  `TRUNCATE(X), where X is a double precision scalar.`
- [ ] Other Library Modules:

Execution Time (microseconds): 28.6

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0, F1	-

Output Results:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup
15	Scalar too large for integer conversion	Return either: Posmax: X'7FFFFFFF' or Negmax: X'80000000'

Comments:

After truncation, negative and positive arguments are closer to 0; no rounding done before truncation.

```
-3   α   -2   -1   0   1   2   β   3
 |   |    |    |   |   |   |   |   |
<------------------------------------>
          |                |
     TRUNCATE(α)      TRUNCATE(β)
```

Registers Unsafe Across Call: R4,R5,F0,F1.

Algorithm:

Same as DROUND, except argument is not rounded up or down.

ROUND

Secondary Entry Name: ETOI

Function: Converts single precision scalar to fullword integer.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
I=S where I is a double precision integer, S is a
   single precision scalar.
```

[ ] Other Library Modules:

Execution Time (microseconds): 39.0

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup
15	Scalar too large for integer conversion	Return either: Posmax =X'7FFFFFFF' or Negmax = X'80000000'

Comments:

ETOI is identical entry point to ROUND; also see DTOI.

Registers Unsafe Across Call: R4,R5,F0,F1.

Algorithm:

Same as ROUND.

Secondary Entry Name: FLOOR

Function: Performs HAL/S FLOOR function: Returns largest integer ≤ the argument.

Note: The compiler now uses inline code for the FLOOR function instead of calling this routine.

The compiler uses the convert to fixed point instruction which is valid only in the range:

.7FFFFF X 16E04 (16) ≥ N ≥ -.800000 X 16E04 (16)

≃ 32767.99 (10) ≥ N ≥ -32768(10)

A convert overflow will occur for FLOOR arguments outside this range.

An invalid result is returned from the FLOOR function for arguments between -16 and 0 which have a fractional portion whose absolute value is smaller than 1/16**4 (I.E. - (X + 1/16**4) < N <-X where X is an integer between 0 and 15 inclusive).

ROUND

Secondary Entry Name: TRUNC

Function: Performs HAL/S TRUNCATE function: Returns signed value that is the largest integer ≤ absolute value of the argument.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`TRUNCATE(X), where X is a single precision scalar.`

[ ] Other Library Modules:

Execution Time (microseconds): 31.4

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup
15	Scalar too large for integer conversion	Return either: Posmax =X'7FFFFFFF' or Negmax =X'80000000'

Comments:

See DTRUNC

Registers Unsafe Across Call: R4,R5,F0,F1.

Algorithm:

Second register of floating point register pair is cleared, then routine merges with DTRUNC.

6.3.2 Algebraic Routine Descriptions

This subsection presents the detailed descriptions of "Algebraic" routines as defined in Appendix C of the *HAL/S Language Specification*.

ACOS

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: ACOS Size of Code Area 116 Hw

Stack Requirement: 24 Hw Data CSECT Size: 2 Hw

[ ] Intrinsic [x] Procedure

Other Library Modules Referenced: SQRT

ENTRY POINT DESCRIPTIONS

Primary Entry Name: ACOS

Function: Computes arc-cosine(x) of scalar argument.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
ARCCOS(X), where X is a single precision scalar.
```

[ ] Other Library Modules:

Execution Time (microseconds): $0.5 < |X| \leq 1$: 225.5

$2.441406252 \times 10^{-4} < |X| \leq 0.5$: 132.7

$|X| \leq 2.441406252 \times 10^{-4}$: 71.5

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	Radians

Errors Detected:

Error #	Cause	Fixup
10	Argument outside range $-1 \leq x \leq 1$	Return π for x<-1 Return 0 for x>1

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4.

Algorithm:

ACOS(X) is computed as $\pi/2$ - ARCSIN(X).

ACOS

Secondary Entry Name: ASIN

Function: Computes arc-sine of scalar argument.

Invoked By:

- [X] Compiler emitted code for HAL/S construct of the form:
  `ARCSIN(X), where X is a single precision scalar.`
- [ ] Other Library Modules:

Execution Time (microseconds):

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	Radians

Errors Detected:

Error #	Cause	Fixup
10	Argument outside range $-1 \le x \le 1$	Return $-\pi/2$ if $x<-1$ Return $\pi/2$ if $x>1$

Comments:

For a very small number of input arguments, the ASIN routine will return results that are accurate to 5 significant decimal digits (instead of the 6 significant decimal digits that are generally required for single precision routines). For most of the range of the ASIN routine, results are returned that are accurate to 6 significant decimal digits.

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

The value of X is restricted to $0 \le X \le 1$ by using the identity arcsin(-X) = -arcsin(X), and further to $0 \le |X| \le 0.5$

by the identity $\arcsin(X) = \frac{\pi}{2} - 2 * \arcsin\left(\sqrt{\frac{1-x}{2}}\right)$

For $0 \le |X| \le 0.5$, $Z = |X|$, and for $0.5 < |X| < 1.0$, $Z = \left(\sqrt{\frac{1-|x|}{2}}\right)$

Arcsin(Z) is then computed as a truncated continued fraction in Z^2, multiplied by W (where W = Z for 0.5 < |X| < 1.0 and W = 2Z for $0 \le |X| \le 0.5$). The form of the approximation is:

$$\arcsin(X) \approx W + \cfrac{W * d_1 * Z^2}{c_1 + Z^2 + \cfrac{d_2}{c_2 + Z^2}}$$

where the values of the constants are:

c_1 = X'C13B446A' = -3.7042025
c_2 = X'C11DB034' = -1.8555182
d_1 = X'C08143C7' = -0.5049404
d_2 = X'C11406BF' = -1.2516474

For arguments $|X|<2.4414063 * 10^{-4}(16^{-3})$, arcsin(X) is computed as arcsin(X) = X.

ACOSH

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: ACOSH Size of Code Area 36 Hw

Stack Requirement: 20 Hw Data CSECT Size: 2 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: LOG,SQRT

ENTRY POINT DESCRIPTIONS

Primary Entry Name: ACOSH

Function: Computes hyperbolic arc-cosine in single precision.

Invoked By:

☒ Compiler emitted code for HAL/S construct of the form:

`ARCCOSH(x), where x is a single precision scalar.`

☐ Other Library Modules:

Execution Time (microseconds):

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
59	ARG < 1	Return 0

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

Using the external SQRT and LOG functions:

$$\text{arccosh}(x)=1n\left(x+\sqrt{x^2+1}\right)$$

ASINH

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: ASINH Size of Code Area 64 Hw

Stack Requirement: 20 Hw Data CSECT Size: 0 Hw

[ ] Intrinsic [x] Procedure

Other Library Modules Referenced: LOG,SQRT

ENTRY POINT DESCRIPTIONS

Primary Entry Name: ASINH

Function: Computes hyperbolic arc-sine in single precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`ARCSINH(X), where X is a single precision scalar.`

[ ] Other Library Modules:

Execution Time (microseconds): |X|<8.876589E-4: 31.5

8.8726589E-4≤|X|<2.1632850E-1: 85.4
2.1632850E-1<|X|<1.6777216E+7: 314.1
|X|≥1.6777216E+7: 141.2

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

Using the external SQRT and LOG routines,
For |X|<8.8726589E-4, arcsinh(X) = X

For 8.8726589E-4≤|X|≤2.1632850E-1, arcsinh(X)=X- $\frac{1}{6}X^3 + \frac{3}{40}X^5$

For 2.1632850E-1<|X|<1.6777216E+7, arcsinh(X)=1n $\left(X + \sqrt{X^2 + 1}\right)$

For |X|≥1.6777216E+7, arcsinh(X) = 1n(X) + 1n(2)

ATANH

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: ATANH Size of Code Area 58 Hw
Stack Requirement: 18 Hw Data CSECT Size: 2 Hw
[ ] Intrinsic [x] Procedure
Other Library Modules Referenced: LOG

ENTRY POINT DESCRIPTIONS

Primary Entry Name: ATANH

Function: Computes hyperbolic arc-tangent in single precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
ATANH(X), where X is a single precision scalar
```

[ ] Other Library Modules:

Execution Time (microseconds): $|X|<4.113892E-5$: 33.9
$4.113892E-5 \leq |X| \leq 1.875E-1$: 85.7
$1.875E-1<|X|$: 228.2

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
60	Argument outside range:-1<X<1	Return 0

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

Using the external LOG function,

For $|X|<4.1138977E-5$, $\text{arctanh}(X) = X$

For $4.1138977E-5<|X|<1.875E-1$, $\text{arctanh}(X) = X + \frac{1}{3}X^3 + \frac{1}{5}X^5$

For $1.875E-1<|X|<1$, $\text{arctanh}(X) = \frac{1}{2}\ln\left(\frac{1+X}{1-X}\right)$

DACOS

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DACOS Size of Code Area 230 Hw

Stack Requirement: 26 Hw Data CSECT Size: 2 Hw

[ ] Intrinsic [x] Procedure

Other Library Modules Referenced: DSQRT

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DACOS

Function: Computes ARCCOS(X) in double precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
ARCCOS(X), where X is a double precision
scalar.
```

[ ] Other Library Modules:

Execution Time (microseconds): $|X| \leq 3.7252903E{-}9$ (16^{-7}): 89.1
3.7252903E-9<|X|≤0.5: 263.1
0.5<|X|<1: 460.5
|X|= 1: 79.7

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	Radians

Errors Detected:

Error #	Cause	Fixup
10	Argument outside range(-1 - epsilon)≤ X≤ (1+epsilon) where epsilon=hex'3AFFFFFFF0000000' (approximately 5.9604644E-08)	Return π if x<-1 Return 0 if x>1

Comments:

The fixup value will be returned, but an error will not be issued until the argument is |X|>1 + epsilon.

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

Computed as $\pi/2$ - ARCSIN(X)

DACOS

Secondary Entry Name: DASIN

Function: Computes ARCSIN(X) in double precision.

Invoked By:

- [X] Compiler emitted code for HAL/S construct of the form:
  `ARCSIN(X), where X is a double precision scalar.`
- [ ] Other Library Modules:

Execution Time (microseconds):

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	Radians

Errors Detected:

Error #	Cause	Fixup
10	Argument outside range(-1 - epsilon)≤X≤ (1 + epsilon) where epsilon=hex'3AFFFFFFF0000000' (approximately 5.96046445E-08)	Return-π/2 if x<-1 Return π/2 if x >1

Comments:

The fixup value will be returned, but an error will not be issued until the argument is |X| > 1 + epsilon.

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

The value of X is restricted to $0 \leq X \leq 1$ by using the identity arcsin(-X) = -arcsin(X), and further to $0 \leq X \leq 1/2$

by the identity $\arcsin(X) = \frac{\pi}{2} - 2 * \arcsin\left(\sqrt{\frac{1-x}{2}}\right)$

For $0 \leq |X| \leq 0.5$, $Z = |X|$, and for $0.5 < |X| < 1.0$, $Z = \left(\sqrt{\frac{1-|x|}{2}}\right)$

Arcsin(Z) is then computed as a truncated continued fraction in Z^2, multiplied by W.

The form of the approximation is:

$$\arcsin(Z)=W + W{*}Z^2\left(C_1 + \cfrac{d_1}{Z^2 + C_2 + \cfrac{d_2}{Z^2 + C_3 + \cfrac{d_3}{Z^2 + C_4 + \cfrac{d_4}{Z^2 + C_5}}}}\right)$$

(where W=Z for 0.5<|X|<1.0, and W=2Z for 0≤|X|≤0.5)
where the values of the constants are:

C_1 = X'3F180CD96B42A610' = .00587162904063511
d_1 = X'C07FE6DD798CBF27' =-.49961647241138661
C_2 = X'C1470EC5E7C7075C' =-4.44110670602864049
d_2 = X'C1489A752C6A6B54' =-4.53770940160639666
C_3 = X'C13A5496A02A788D' =-3.64565146031194167
d_3 = X'C06B411D9ED01722' =-.41896233680025977
C_4 = X'C11BFB2E6EB617AA' =-1.74882357832528117
d_4 = X'BF99119272C87E78' =-.03737027365107758
C_5 = X'C11323D9C96F1661' =-1.19625261960154476

For arguments |X|<3.7252903 E-9(16^{-7}), arcsin(X) is computed as arcsin(X)=X.

DACOSH

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DACOSH Size of Code Area 50 Hw

Stack Requirement: 22 Hw Data CSECT Size: 2 Hw

[ ] Intrinsic [x] Procedure

Other Library Modules Referenced: DLOG, DSQRT

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DACOSH

Function: Computes hyperbolic arc-cosine in double precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`ARCCOSH(X),  where X is a double precision scalar.`

[ ] Other Library Modules:

Execution Time (microseconds): 1≤X<6.7108864E+7: 403.4
6.7108864E+7<X: 332.4

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
59	Argument<1- EPSILON	Return 0

Note: The fixup value will be returned but an error will not be issued until x<0.9999999403953555 where EPSILON=hex'3AFFFFFFF0000000' (approximately 5.96046445E-08).

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

Using the external DSQRT and DLOG functions,

For $1 \le x < 6.7108864 \times 10^7$, $\mathrm{arccosh}(x) = 1n\,(x + \sqrt{x^2 - 1})$

For $x \ge 6.7108864 \times 10^7$, $\mathrm{arccosh}(x) = 1n(x) + 1n(2)$

DASINH

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DASINH Size of Code Area 96 Hw
Stack Requirement: 22 Hw Data CSECT Size: 0 Hw
[ ] Intrinsic [x] Procedure
Other Library Modules Referenced: DLOG, DSQRT

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DASINH
Function: Computes hyperbolic arc-sine in double precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
ARCSINH(X),  where X is a double precision scalar.
```

[ ] Other Library Modules:

Execution Time (microseconds): |X|≤1.353860E-8: 33.6
1.353860E-8<|X|≤6.25E-2: 185.4
6.25E-02<|X|<6.7108864E+7: 570.8
6.7108864E+7≤|X|: 348.2

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

Using the external DSQRT and DLOG functions,
For |X|<1.353860E-8, arcsinh(X) = X
For 1.353860E-8<|X|<6.25E-2, arcsinh(X) =

$$X - \frac{1}{6}X^3 + \frac{3}{40}X^5 - \frac{15}{336}X^7 + \frac{105}{3456}X^9 - \frac{945}{42240}X^{11}$$

For 6.25E-2<|X|<6.7108864E+7, arcsinh(X) =1n(X+ $\sqrt{x^2 - 1}$)

For 6.7108864E+7≤|X|, arcsinh(X) = 1n(X) + 1n(2)

DATANH

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DATANH Size of Code Area 132 Hw

Stack Requirement: 26 Hw Data CSECT Size: 2 Hw

[ ] Intrinsic [x] Procedure

Other Library Modules Referenced: DLOG

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DATANH

Function: Computes hyperbolic arc-tangent in double precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
ARCTANH(X),  where X is a double precision scalar.
```

[ ] Other Library Modules:

Execution Time (microseconds): |X|<1.07455946E-8: 42.6
1.07455946E-8≤|X|<6.25E-2: 186.6
6.25E-2≤|X|: 399.0

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
60	Argument outside range: (-1 - EPSILON)<X<(1 + EPSILON) where EPSILON = hex'3AFFFFFFF0000000' (approximately 5.96046445E-08)	Return 0

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

Save sign of argument X
X=|X| (force positive, arctanh(-X)= -arctanh(X))
If X>1: Return 0, indicate error
If X<1.07455946E-08: Return X

If 1.07455946E-08≤X<6.250E-02: $\text{arctanh}(X) = X + \frac{X^3}{3} + \frac{X^5}{5} + \frac{X^7}{7} + \frac{X^9}{9}$

If 6.250E - 02≤X: arctanh (X)= 1/2 1n ((1+X)/(1-X))
(uses the external DLOG library function)
Note: For non-zero results, set the sign of the result to the original sign of X.

DATANH2

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DATANH 2 Size of Code Area 342 Hw

Stack Requirement: 26 Hw Data CSECT Size: 26 Hw

[ ] Intrinsic [x] Procedure

Other Library Modules Referenced:

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DATAN2

Function: Computes arctan by fraction approximation in the range $(-\pi,\pi)$ in double precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`ARCTAN2(X,Y), where X and Y are double precision scalars corresponding to sine and cosine respectively of the intended arc tangent argument.`

[ ] Other Library Modules:

Execution Time (microseconds): 248.4

Input Arguments:

Type	Precision	How Passed	Units
Scalar (sin)	DP	F0	-
Scalar (cos)	DP	F2	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	Radians

Errors Detected:

Error #	Cause	Fixup
62	arg 1 = arg 2 = 0	Return 0

Comments:Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

Same algorithm as EATAN2, but values of constants and the fractional approximation formula is different for the double precision, as follows.

Again, $Z=\left|\frac{\sin x}{\cos x}\right|$. Special cases:

(1) If cos x<0 and $Z<16^{-14}$, return $\pm\pi$

(2) sin x=cos x=0, signal error and return 0.

(3) sin x≠0, cos x=0, return $\pm\pi/2$.

(4) sin x≠0, cos x≠0, but $Z>16^{14}$, return $\pm$ π/2.

(5) If (Z or 1/Z)<16^{-7}, return Z.

(6) If exponent of sin(x)- exponent of cos(x)>15,
return $\pm$ π/2.

(7) If exponent of sin(x)- exponent of cos(x) <-51,
return arctan(0).

The fractional approximation after reduction of Z to ≤ tan 15° is:

$$\mathrm{Tan}^{-1}(Z) = Z + Z * Z^2 * F, \text{ where}$$

$$F = C1 + C2/(Z^2 + C3 + C4/(Z^2 + C5 + (C6/(Z^2 + C7)))).$$

C1 = X'BF1E31FF1784B965' (-0.7371899082768562E-2)
C2 = X'C0ACDB34C0D1B35D' (-0.6752198191404210)
C3 = X'412B7CE45AF5C165' (0.2717991214096480E+1)
C4 = X'C11A8F923B178C78' (-0.1660051565960002E+1)
C5 = X'412AB4FD5D433FF6' (0.2669186939532663E+1)
C6 = X'C02298BB68CFD869' (-0.1351430064094942)
C7 = X'41154CEE8B70CA99' (0.1331282181443987E+1)

As in EATAN2, the intermediate result is adjusted to the proper section in the first quadrant, as follows:

(original) Z ≤ tan 15°		+0
tan 15°<Z ≤ 1	→	+π/6
1/Z ≤ tan 15°	→	(-π/2 + 1) then-1 (to preserve significant bits)
tan 15°< 1/Z ≤1	→	(-π/3 + 1) then-1 (to preserve significant bits)

The resulting angle is adjusted to the proper quadrant as in EATAN2 (according to sign of sin x and cos x).

DATAN2

Secondary Entry Name: DATAN
Function: Computes arc tangent by fractional approximation in the range $(-\pi/2, +\pi/2)$ in double precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:
`ARCTAN(X), where X is a double precision scalar.`

[ ] Other Library Modules:

Execution Time (microseconds): 237.3

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	Radians

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

Same as ARCTAN, but see DATAN2 for changes in values of DP constants and TAN^{-1} formula.

DEXP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DEXP Size of Code Area 158 Hw

Stack Requirement: 18 Hw Data CSECT Size: 66 Hw

[ ] Intrinsic [x] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DEXP

Function: Computes e^X in double precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`EXP(X), where X is a double precision scalar.`

[X] Other Library Modules:

DPWRD, DSINH, DTANH

Execution Time (microseconds): 290.5

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
6	X>174.673085	Return maximum positive floating point number

Comments:

Gives exponent underflow if argument less than -180.21825 -- no RTL error number; GPC error group 3 code 9. The error will not be detected if the program status word masks out the underflow errors.

Registers Unsafe Across Call: F0,F1,F2,F3.

Algorithm:

First, decompose X as P'log2 + R', where P' is the integer part and first hexadecimal place of the result of dividing the high-order part of X by LOG2H, which is a single precision approximation to log2, rounded up. This is done in 80-bit precision in order to yield a true 56-bit value for R', by expressing log2 = LOG2H + LOG2L, where LOG2L is a double precision scalar. R' has the same sign as X, and |R'| might be slightly > $\frac{\log 2}{16}$.

Now, if R'>0, subtract $\frac{\log 2}{16}$ from it until it becomes $\leq$ 0, each time adding $\frac{1}{16}$ to P'.

If R' $\leq$- $\frac{\log 2}{16}$, add $\frac{\log 2}{16}$ to it until it becomes >- $\frac{\log 2}{16}$, each time subtracting $\frac{1}{16}$ from P'.

At the end of this, we have

$$X = P * \log 2 + R, \text{ P an integral multiple of } \frac{1}{16}, \text{ and } -\frac{\log 2}{16} < R \leq 0.$$

Represent P as 4A - B - C/16 , where A, B, and C are integers, 0 < B$\leq$3, 0$\leq$C$\leq$15. Then:

$$e^x = 16^A * 2^{-B} * 2^{-C/16} * e^R$$

To calculate this, we compute e^R with a polynomial approximation of the form:

$$e^r = 1 + c_1 r + c_2 r^2 + c_3 r^3 + c_4 r^4 + c_5 r^5 + c_6 r^6$$

where the values of the constants are:

c_1 = X'40FFFFFFFFFFFCFC'	= .9999999999999892
c_2 = X'407FFFFFFFFAB64A'	= .4999999999951906
c_3 = X'402AAAAAA794AA99'	= .1666666659481656
c_4 = X'3FAAAA9D6AC1D734'	= .0416666173078875
c_5 = X'3F2220559A15E158'	= .00833161772003906
c_6 = X'3E591893'	= .001359497

Then, $2^{-C/16}$ is computed by table lookup, 2^{-B} by shifting, and 16^A by adding A to the exponent of the answer.

DLOG

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DLOG Size of Code Area 184 Hw

Stack Requirement: 30 Hw Data CSECT Size: 2 Hw

[ ] Intrinsic [x] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DLOG

Function: Computes 1n(X) in double precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
LOG(X), where X is a double precision scalar.
```

[X] Other Library Modules:

DPWRD, DASINH, DATANH, DACOSH

Execution Time (microseconds): 282.2

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
7	Argument outside range X > 0	If X<0 return 1n(|X|); if X=0, return maximum negative floating point number

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

We write $X = 16^{P} * 2^{-Q} * M$, where $½ \le M < 1$, P, Q are integers, $0 \le Q \le 3$. P, Q, and M are found by fixed-point calculations. Define A=1, B=0, if $M > \sqrt{2}/2$, and A=1/2, B=1 otherwise. Let Z=(M-A)/(M+A).

Then 1n(X) = (4P-Q-B)1n(2)+1n((1+Z)/(1-Z)) is computed by an approximation of the form:

$$W+C_1 W^3\left(W^2+C_2+\cfrac{C_3}{W^2+C_4+\cfrac{C_5}{W^2+C_6}}\right)$$

where W=2Z, and the values of the constants are:

c_1	= X'3DDABB6C9F18C6DD'	= 0.2085992109128247E-3
c_2	= X'422FC604E13C20FE'	= 0.4777351196020117E+2
c_3	= X'C38E5A1C55CEB1C4'	= -0.2277631917769813E+4
c_4	= X'C16F2A64DDFCC1FD'	= -6.947850100648906
c_5	= X'C12A017578F548D1'	= -2.625356171124214
c_6	= X'C158FA4E0E40C0A5'	= -5.561109595943017

DPWRD

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DPWRD Size of Code Area 40 Hw
Stack Requirement: 22 Hw Data CSECT Size: 2 Hw
[ ] Intrinsic [x] Procedure
Other Library Modules Referenced: DEXP,DLOG,DSQRT

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DPWRD

Function: Performs exponentiation of double precision scalar to double precision power.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
X**Y,  where X and Y are scalars and at least X or Y
       is double precision.
```

[ ] Other Library Modules:

Execution Time (microseconds):

Input Arguments:

Type	Precision	How Passed	Units
Scalar(X)	DP	F0	-
Scalar(Y)	DP	F2	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
4	X=0; Y$\leq$0	Return 0
24	X<0	Return |X|

Comments:

Other than Errors 4 and 24, no additional range or overflow checking is performed.
Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

If exponent = 0.5, compute $X^{0.5}$ as $\sqrt{X}$ (by using the external DSQRT function), otherwise $X^Y = e^{Y \text{ Log } X}$, using the external DEXP and DLOG functions. The call to DEXP could result in error #6 if Y Log X is sufficiently large.

DPWRI

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DPWRI Size of Code Area 54 Hw

Stack Requirement: 26 Hw Data CSECT Size: 2 Hw

[ ] Intrinsic [x] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DPWRI

Function: Exponentiation of a double precision scalar to a fullword integer power.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
X**I  where X and Y are scalars and at least X or Y
      is double precision.
```

[ ] Other Library Modules:

Execution Time (microseconds):

Input Arguments:

Type	Precision	How Passed	Units
Scalar(base)	DP	F0	-
Integer(exponent)	DP	R6	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
4	Zero raised to power≤0	Return 0

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3.

Other than detection of Error 4, no additional range or exponent overflow checking is done. According to the *Space Shuttle Model AP101-S Principles of Operation*, an exponent overflow occurs when the result exponent exceeds 16**63 and an exponent underflow occurs when the result exponent is less than 16**-64. For base ** |exp|, if |base|>1 then an exponent overflow is possible. If exp<0 and 0<|base|<1 then an exponent underflow is possible. If the exponent underflow mask bit is zero (inhibiting the interrupt) then the result is a true zero. This will cause a divide by zero GPC error when the reciprocal of the result is taken. If the exponent underflow mask bit is one (enabling the interrupt) then the operands are unchanged. This could cause an exponent overflow when the reciprocal of the result is taken.

Algorithm:

If I is the fullword exponent, D the base, write

$$I = \sum_{i=0}^{32} e_i 2^i, \text{ where } e_i = 0 \text{ or } 1.$$

Then:

$$D^I = D^{\Sigma_i e_i 2^i} = \prod_{i=0}^{32} D^{e_i 2^i} = \Pi_{e_i=1} D^{2^i}, \text{ if any } e_i = 1, \text{ and } = 1 \text{ otherwise.}$$

To compute $\Pi_{e_i=1} D^{2^i}$, it is only necessary to compute successively $D^{2^i} = D, D^2, D^4, D^8, \ldots$, and multiply the result by D^{2^i} whenever the i-th bit of the exponent is 1. This is determined by shifting bits one by one out of the exponent, and testing each one for a value of one. The loop terminates when the remaining part of the exponent is zero. Operations are done on absolute value of exponent. If exponent was negative, the reciprocal of the result is taken as the final result.

DPWRI

Secondary Entry Name: DPWRH

Function: Exponentiation of a double precision scalar to a halfword integer power.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
X**I  where X is a double precision scalar; I is a
      single precision integer.
```

[ ] Other Library Modules:

Execution Time (microseconds):

Input Arguments:

Type	Precision	How Passed	Units
Scalar(X)	DP	F0	-
Integer(I)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
4	Zero raised to power ≤ 0	Return 0

Comments:

Other than detection of Error 4, no additional range or exponent overflow checking is done. According to the *Space Shuttle Model AP101-S Principles of Operation,* an exponent overflow occurs when the result exponent exceeds 16**63 and an exponent underflow occurs when the result exponent is less than 16**-64. For base ** |exp|, if |base|>1 then an exponent overflow is possible. If exp<0 and 0<|base|<1 then an exponent underflow is possible. If the exponent underflow mask bit is zero (inhibiting the interrupt) then the result is a true zero. This will cause a divide by zero GPC error when the reciprocal of the result is taken. If the exponent underflow mask bit is one (enabling the interrupt) then the operands are unchanged. This could cause an exponent overflow when the reciprocal of the result is taken.

Registers Unsafe Across Call: F0,F1,F2,F3.

Algorithm:

The halfword exponent is shifted right to convert it to a fullword, then the DPWRI algorithm is used.

DSINH

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DSINHI Size of Code Area 130 Hw
Stack Requirement: 22 Hw Data CSECT Size: 2 Hw
[ ] Intrinsic [x] Procedure
Other Library Modules Referenced: DEXP

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DSINH

Function: Computes hyperbolic sine in double precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`SINH(X), where X is a double precision scalar.`

[ ] Other Library Modules:

Execution Time (microseconds): 8.81374E-1≤|X|≤1.75366E+2: 434.1
2.063017E-10≤|X|<8.81374E-01: 196.7
|X|<2.063017E-10: 45.8

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
9	Argument outside range: |X|≤175.366	Return maximum positive floating point number

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

If |X|<1.626459E-10, then sinh(X) = X.

If 1.626459E-10≤|X|<.881375, then sinh(X) is computed via a polynomial approximation.

The form of the polynomial approximation is:

$$\sinh(X) = X + C_1X^3 + C_2X^5 + C_3X^7 + C_4X^9 + C_5X^{11} + C_6X^{13}$$

where the values of the constants are:

C_1 = X'402AAAAAAAAAAA4D' = 0.1666666666666653
C_2 = X'3F2222222222BACE' = 0.8333333333367232E-2
C_3 = X'3DD00D00CB06A6F5' = 1.984126981270711E-4
C_4 = X'3C2E3BC881345D91' = 2.755733025610683E-6
C_5 = X'3A6B96B8975A1636' = 2.504995887597646E-8
C_6 = X'38B2D4C184418A97' = 1.626459177981471E-10

Otherwise, sinh(|X|) or cosh(|X|) is calculated using EXP. The number V, equal to 0.4995050, is introduced to control rounding errors and the formula is as follows:

$$\sinh(X) = \frac{1}{2v}\left(e^{(X+\log v)} - \frac{v^2}{e^{(X+\log v)}}\right)$$

$$\cosh(X) = \frac{1}{2v}\left(e^{(X+\log v)} - \frac{v^2}{e^{(X+\log v)}}\right)$$

The identities sinh(-X) = -sinh(X) and cosh(-X) = cosh(X) are used to recover sinh(X) and cosh(X) from sinh(|X|) and cosh(|X|).

DSINH

Secondary Entry Name: DCOSH

Function: Computes hyperbolic cosine in double precision.

Invoked By:

- [X] Compiler emitted code for HAL/S construct of the form:
  `COSH(X), where X is a double precision scalar.`
- [ ] Other Library Modules:

Execution Time (microseconds): $|X| \leq 1.75366E+2$: 422.6

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
9	Argument outside range $\|X\| \leq 175.366$	Return maximum positive floating point number

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

See DSINH Algorithm.

DSNCS

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DSNCS Size of Code Area 140 Hw

Stack Requirement: 28 Hw Data CSECT Size: 64 Hw

[ ] Intrinsic [x] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DSNCS

Function: Computes sine(X) and cosine(X) in double precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
SIN(X)and COS(X), where X is a double precision scalar
                  that is recognized as the same for
                  both invocations.
```

[ ] Other Library Modules:

Execution Time (microseconds):

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	Radians

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-
Scalar(COS Result)	DP	F2	-

Errors Detected:

Error #	Cause	Fixup
8	Argument outside range: |X|<823296.0625 (approximately $\pi * 2^{18}$)	Return $\frac{\sqrt{2}}{2}$

Comments:

The actual values of 32/π (X'41A2F9836E4E4414') and π/32 (X'401921FB54442D17') are not used. Instead, an approximate value is used for 32/π(X'41A2F9836E4E45C9') because the lower halfword of this value (45C9) is used as the maximum limit of the DCOS, DSIN, and DSNCS routines. Similarly, an approximate value is used for π/32 (X'401921FB544420B9') because the lower halfword of this value (20B9) is used as the underflow limit of the DCOS, DSIN, and DSNCS routines.

The precision of the DSNCS outputs is limited to 8 significant digits relative to 1.0 (or +/-1E-8)

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

Register 5 is first set to indicate the sign of the argument (x) and which entry point was used. For |x| < X'25100000' (floating point), x gets set to zero to prevent underflow. The argument is then multiplied by $32/\pi$; the integer portion of this result is converted to fixed point and the fractional portion (remainder) is retained in floating point (Δ). The fixed point number is an integer multiple of the number of $\pi/32$ sections that comprise the number, divided into the following logical sections:

- the number of 2πs in the original number,
- the octant in which the argument lies (within the range 0 to 2π),
- the number of the $\pi/32$ interval in which the argument lies within a range of 0 to $\pi/4$ (used as an index value into a table of sine and cosine values).

If the octant is odd, the argument has to be complemented. This is done by complementing each of the index bits and taking $\Delta = 1 - \Delta$. The table of index values are good for only odd $\pi/32$ intervals, so for the even index values $\Delta = \Delta - 1$.

However, this is not done for an index value of zero. An index value of zero indicates that the argument is in the range of $n\pi/2 - \pi/32 \leq x \leq n\pi/2 + \pi/32$, n=0, $\pm$ 1, $\pm$ 2,..., in which case Δ is left unchanged and the index value made negative for later use. At this time the octant is isolated and saved for later use and the number of 2πs is no longer needed and dropped.

STEP A): The floating point remainder (Δ) represents the normalized increment of the argument relative to one of the table arguments. At this point, the sine and cosine of this increment can be evaluated with the following polynomials:

$$\sin \Delta\frac{\pi}{32} = \Delta\left(\frac{\pi}{32} + \Delta^2(\mathrm{AS}\Delta^2 + \mathrm{BS})\right)$$

$$\cos\Delta\frac{\pi}{32} = 1.0 + 2\Delta^2(\mathrm{AC}\Delta^2 + \mathrm{BC})$$

Where:

AS = X'3B14634A'

BS = X'BDA55DE5'

AC = X'3C2075A1'

BC = X'BE9DE937'

STEP B): If the sign of the index value (previously computed) is not negative then the following trigonometric identities are used to compute the sine and/or cosine:

$$\sin x = \sin(\mathrm{TS}_i + \Delta\frac{\pi}{32}) = (\sin \mathrm{TS}_i)(\cos \Delta\frac{\pi}{32}) + (\cos \mathrm{TS}_i)(\sin\ )\ \Delta\frac{\pi}{32}$$

$$\cos x = \cos(\mathrm{TC}_i + \Delta\frac{\pi}{32}) = (\cos \mathrm{TC}_i)(\cos \Delta\frac{\pi}{32}) + (\sin \mathrm{TC}_i)(\sin \Delta\frac{\pi}{32})$$

The appropriate equation and values for TS_i and TC_i are determined by the octant value.

Octant	Equation	I
0	Sin	0
1	Cos	0
2	Cos	1
3	Sin	1
4	Sin	1
5	Cos	2
6	Cos	2
7	Sin	3

and TS and TC are given by the table:

TS_0 = D'.09801714032875'

TS_1 = D'.290284677254'

TS_2 = D'.4713967368259'

TS_3 = D'.6343932841633'

TC_0 = D'.9951847266737'

TC_1 = D'.9569403357347'

TC_2 = D'.8819212643506'

TC_3 = D'.7730104533640'

If the index value is negative the sine or cosine computed from STEP A is used without the table or the above equations. Finally, the sign of the result is complemented if the octant number is greater than 3. If the entry point was DSNCS then the octant is incremented by 2 (MOD 8) and the logic STEP B is executed again.

Output is passed back in the register pair F0/F1 if DSIN or DCOS was called. If the entry point was DSNCS then the output is in F0/F1 for Sin and F2/F3 for Cos.

DSNCS

Secondary Entry Name: DCOS

Function: Computes cosine(x) in double precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`COS(X), where X is a double precision scalar.`

[ ] Other Library Modules:

Execution Time (microseconds): $-\pi \leq X \leq \pi$: 261.8

$X > \pi$ or $X < -\pi$: 264.2

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	Radians

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
8	Argument outside range $\|X\|<823296.0625$ (approximately $\pi * 2^{18}$)	Return $\frac{\sqrt{2}}{2}$

Comments:

For most of the range of input arguments to the DCOS routine, results are returned that are accurate to 5 significant decimal digits (instead of the 8 significant decimal digits that are generally required for double precision routines).

The value used in the routine for $\pi(2^{18})$ is hex'45C90000'=823296. Because a halfword instruction is used in the limit test, the first error occurs when $|X|=823296.0625$.

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

See DSNCS Algorithm.

DSNCS

Secondary Entry Name: DSIN

Function: Computes sine(x) in double precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`SIN(X), where X is a double precision scalar.`

[ ] Other Library Modules:

Execution Time (microseconds): 267

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	Radians

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
8	Argument outside range |X|<823296.0625 (approximately $\pi*2^{18}$)	Return $\frac{\sqrt{2}}{2}$

Comments:

For most of the range of input arguments to the DSIN routine, results are returned that are accurate to 5 significant decimal digits (instead of the 8 significant decimal digits that are generally required for double precision routines). The value used in the routine for $\pi*(2^{18})$ is hex'45C90000' = 823296. Because a halfword instruction is used in the limit test, the first error occurs when |X| = 823296.0625.

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

See DSNCS Algorithm.

DSQRT

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DSQRT Size of Code Area 184 Hw

Stack Requirement: 18 Hw Data CSECT Size: 8 Hw

[ ] Intrinsic [x] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DSQRT

Function: Computes square root in double precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
SQRT(X), where X is a double precision scalar.
```

[X] Other Library Modules:

DACOS, DASINH, DPWRD, VV10D3, DACOSH

Execution Time (microseconds): 345.2

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
5	X<0	Return sqrt (|X|)

Comments:

Prior to Release 11.0 of the HAL/S-FC compiler, this routine computed square roots in full (8/56) precision. PCR 4791 was incorporated in Release 11.0 and reduced precision to 8/31. For further information refer to PCR 4791.

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

Since the input value X_0 is in floating point format, it can be viewed as an exponent value 16^{2P+Q} and a mantissa value M. The input value X can therefore be represented as:

$$X_0 = 16^{2P+Q} \cdot M_0$$

where Q = 0 if the characteristic is even and Q = 1 if the characteristic is odd (i.e. Q represents the LSB of the characteristic). If Q is subtracted from the characteristic then the remaining portion of the characteristic can be represented as a multiple of 2 (i.e. 2P). The input value is assumed to be normalized so the value of M can be defined as

$$\frac{1}{16} \leq M_0 < 1$$

A first approximation of $\sqrt{X}_0$ is made by processing each part of the input argument (characteristic & mantissa) separately. The characteristic of $\sqrt{X}_0$ is computed by adding 1 to the characteristic of X_0 and dividing by 2. Therefore:

$$C_{x1} = \frac{(C_{x0} + 1)}{2}$$

where C_{x1} is the characteristic of the first approximation and C_{x0} is the characteristic of the input value.

The square root of the mantissa, M_1, is computed using a quadratic equation of the form:

$$M_1 = AM_0^2 + BM_0 + C$$

where A, B, and C are precomputed halfword constants and M_0 is the mantissa of the input value X in halfword fixed point format. Two values are possible for each of the constants A, B, and C. The assumed values of these constants are dependent on the value of the fixed point mantissa:

M_0	<0.25	$M_0 \geq 0.25$
A	HEX'AF76'	HEX'F5EF'
B	HEX'433E'	HEX'219F'
C	HEX'0427'	HEX'084D'

The characteristic C_{X1} and mantissa M_1 are then recombined by shifting and 'ORing' producing a floating point format first approximation X_1. This value is then used in the first of two Newton-Raphson iterations. The form of the first is:

$$X_2 = \frac{1}{2}\left(\frac{X_0}{X_1} + X_1\right)$$

The value X2 is then used in a second iteration of Newton-Raphson of the form:

$$X_3 = X_2 + \frac{X_0 - X_2^2}{2X_2}$$

The final result X_3 is returned, in double precision floating point format, to the calling routine via F0.

Warning:

The DSQRT will return correct results if the argument is less than 7.2368577E+75. For arguments $\geq$ 7.2368577E+75, floating point exponent overflow occurs and incorrect results are returned.

DTAN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DTAN Size of Code Area 196 Hw

Stack Requirement: 38 Hw Data CSECT Size: 4 Hw

[ ] Intrinsic [x] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DTAN

Function: Computes tangent in double precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`TAN(X), where X is a double precision scalar.`

[ ] Other Library Modules:

Execution Time (microseconds):

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	Radians

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
11	Argument outside range: $\|X\|<\pi_*2^{50}$	Return 1
12	Argument too near a singularity of the tangent function	Return maximum positive floating point number

Comments:

Error gets very large near a singularity, before error #12 is sent.

The value used in the routine for π_*2^{50} is hex'4DC90FDA' ≈ 3.53711870600810E+15.

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

Multiply X by $\frac{4}{\pi}$, and give the characteristic of this to X'0000000000000008' for use as a comparand to determine nearness to a singularity.

The integer part of $|X * \frac{4}{\pi}|$ is the octant.

If the octant is even, let w = fraction part of $|X| * \frac{4}{\pi}$.

If the octant is odd, let w = -(1 - fraction) part of $|X| * \frac{4}{\pi}$.

Next, compute two polynomials P(w) and Q(w).

If $w \geq 2^{-46}$, then the forms of the polynomials are:

$$P(w) = w(a_0 + a_1w^2 + a_2w^4 + w^6)$$

$$Q(w) = b_0 + b_1w^2 + b_2w^4 + b_3w^6$$

If $w < 2^{-46}$, then with u = w if $|X| * \frac{4}{\pi} < 1$, and u = -w otherwise.

$$P(w) = w(a_0 + u)$$

$$Q(w) = b_0 + b_3u$$

where the values of the constants are:

a_0 = X'C58AFDD0A41992D4'	=-569309.04006345
a_1 = X'44AFFA6393159226'	= 45050.3889630777
a_2 = X'C325FD4A87357CAF'	=-607.8306953515
b_0 = X'C5B0F82C871A3B68'	=-724866.7829840012
b_1 = X'4532644B1E45A133'	= 206404.6948906228
b_2 = X'C41926DBBB1F469B'	=-6438.8583240077
b_3 = X'422376F171F72282'	= 35.4646216610

If w ≤ the comparand derived earlier and the octant = 1 or 2 (mod 4), then error 12 is sent. Otherwise, Q(w)/P(w) is returned with its sign adjusted. In octants = 0 or 3 (mod 4), P(w)/Q(w) is returned, with the sign adjusted according to tan(-x) = -tan(x). The justification for this computation is that

$$\frac{P(w)}{Q(w)} = \tan\left(W * \frac{\pi}{4}\right) \text{ and } \frac{Q(w)}{P(w)} = \cot\left(W * \frac{\pi}{4}\right)$$

and simple trigonometric identities give, for R=fraction part of $X * \frac{4}{\pi}$:

Octant (mod 4)	Formula for tan
0	$\tan(\lvert x \rvert) = \tan\left(R * \frac{\pi}{4}\right)$
1	$\tan(\lvert x \rvert) = \cot\left((1-R) * \frac{\pi}{4}\right)$
2	$\tan(\lvert x \rvert) = -\cot\left(R * \frac{\pi}{4}\right)$
3	$\tan(\lvert x \rvert) = -\tan\left((1-R) * \frac{\pi}{4}\right)$

which is the result of the computation as performed.

Notes:

DTAN does not always meet the 8 significant decimal digit precision requirement for double precision RTL routines because of a limitation in the algorithm. The algorithm multiplies the input value (X) by $4/\pi$ to determine the octant. The integer part of the product $|X*4/\pi|$ is the octant. Depending on whether the octant is even or odd, the fraction part or the sign inverse of one minus the fraction part of $|X*4/\pi|$ is input into a set of polynomial equations to determine the tangent value. The precision limitation is caused by using this fractional number.

Double precision floating point numbers on the AP-101 system are represented by 64 bits where the first bit is the sign, the next 7 bits give a hexadecimal exponent biased by 64, and the final 56 bits are a fractional mantissa.

$$(-1)^{\text{sign}} * 16^{(\text{exponent -bias})} * \text{fraction}$$

The fraction is composed as follows:

$$f = \text{bit9}*2^{-1} + \text{bit10}*2^{-2} + \text{bit11}*2^{-3} + \ldots + \text{bit64}*2^{-56}$$

This means that if the number being represented is completely fractional (zero on the left of the radix point) it can be as accurate as $1 \text{x} 2^{-56}$ or approximately $1.39 \text{x} 10^{-17}$. As the input value gets larger, more of the bits in the mantissa are used to represent the integer portion of the number to the left of the radix point which is used to determine the octant. This leaves fewer bits to represent the fractional part of the number which determines the tangent value.

For exponents equivalent to 2^{50} this means that the bits used for the fraction can only be as accurate as

$$2^{50} * 2^{-56} = 2^{-6} \text{ or } 1.5625*10^{-2}$$

In order to retain eight digits of accuracy in the fractional portion of the mantissa, and therefore the tangent value, the exponent of the input must be no larger than the equivalent of 2^{29}.

DTANH

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DTANH Size of Code Area 154 Hw
Stack Requirement: 30 Hw Data CSECT Size: 0 Hw
[ ] Intrinsic [x] Procedure
Other Library Modules Referenced: DEXP

ENTRY POINT DESCRIPTIONS

Primary Entry Name: DTANH

Function: Computes hyperbolic tangent in double precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:
`TANH(X), where X is a double precision scalar.`

[ ] Other Library Modules:

Execution Time (microseconds): X|≤3.72529E-9: 47.8
3.72529E-9<|X| ≤ 5.4931E-1: 177.9
5.4931E-1<|X| < 2.0101E+1: 420.6
2.0101E+1≤|X|: 54.6

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

If |X|>20.101, return +1 or -1, according to the sign of X.
If 0.54931≤|X|≤20.101, then (using DEXP),

$$\tanh(\,|X|\,) = 1 - \frac{2}{1 + e^{2|X|}}$$

Restore sign with tanh(-X) = -tanh(X). For |X|≤16^{-7}, tanh(X) = X.

For other values of X, $16^{-7}<|X|<0.54931$, use a continued fraction approximation:

$$\tanh(X) = X + X^3 \left(\cfrac{C_0}{X^2 + C_1 + \cfrac{C_2}{X^2 + C_3 + \cfrac{C_4}{X^2 + C_5}}} \right)$$

where the values of the constants are:

C_0 = X'C0F6E12F40F5590A' =-.9643735440816707
C_1 = X'419DA5D6FD3DBC84' = 9.8529882328255392
C_2 = X'C31C504FEF537AF6' =-453.01951534852503
C_3 = X'424D2FA31CAD8D00' = 77.186082641955181
C_4 = X'C3136E2A5891D8E9' =-310.8853383729134
C_5 = X'4219B3ACA4C6E790' = 25.701853083191565

EATAN2

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: EATAN2 Size of Code Area 148 Hw
Stack Requirement: 18 Hw Data CSECT Size: 10 Hw
[ ] Intrinsic [x] Procedure
Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: EATAN2
Function: Computes arctangent by fractional approximation in the range (-π,π) in single precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
ARCTAN2(X,Y), where X and Y are single precision scalars
              corresponding to sine and cosine
              respectively of the intended arctangent
              argument.
```

[ ] Other Library Modules:

Execution Time (microseconds): 120.0

Input Arguments:

Type	Precision	How Passed	Units
Scalar(sin)	SP	F0	-
Scalar(cos)	SP	F2	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	Radians

Errors Detected:

Error #	Cause	Fixup
62	arg 1 = arg 2 = 0	Return 0

Comments:

For a very small number of input arguments, the EATAN2 routine will return results that are accurate to 5 significant decimal digits (instead of the 6 significant decimal digits that are generally required for single precision routines). For most of the range of the EATAN2 routine, results are returned that are accurate to 6 significant decimal digits.

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

The pointer to the data area that contains quadrant section constants is set and the sign of sin(x) is saved. The value Z = |sin(x)|/|cos(x)| is checked for several special cases.

(1) If cos(x)<0 and $Z<16^{-6}$, then return $\pi *$ SIGNUM(sin(x)).

(2) If sin(x) = cos(x) = 0, then signal error and return 0.

(3) If sin(x) $\neq$ 0, cos(x) = 0, then return $\pm\pi/2 *$ SIGNUM(sin(x)).

(4) If sin(x) $\neq$ 0, cos(x) $\neq$ 0, but $Z>16^{6}$, again return $\pm\pi/2$ = ($\pi/2*$SIGNUM(sin(x))).

(5) If (Z or 1/Z)$<16^{-3}$, return Z.

(6) If exponent of sin(x) - exponent of cos(x)$\geq$7, return $\pm\pi/2$.

Now, all of the special cases have been checked for. If the routine gets this far, it is time to reduce Z = tan x so that Z < tan $\pi/12$(tan 15°)

There are four cases to examine for Z in the 1st quadrant.

A) Z>1. Use 1/Z.

tan 15°<1/Z$\leq$1

1/Z$\leq$tan 15°

B) Z $\leq$1

tan 15°<Z$\leq$1

Z$\leq$tan 15°

For Z or 1/Z>tan 15°, the reduction

$\tan^{-1}(Z) = \pi/6 + \tan(Y)$, where $Y = (Z\sqrt{3} -1)/(Z+ \sqrt{3})$ is used.

To protect significant bits, Y is computed as

$Y = (Z (\sqrt{3} - 1) - 1 + Z)/(Z + \sqrt{3})$

Once Z or 1/Z$\leq$tan 15°, the formula for arctan Z can be applied.

$$\frac{\tan^{-1}(Z)}{Z} = D + CZ^2 + (B/(Z^2 + A)),$$

where the constants have the following values (hex values are used in the routine):

A	=	X'41168A5E'	=	1.4087812
B	=	X'408F239C'	=	0.55913711
C	=	X'BFD35F49'	=	-0.051604543
D	=	X'409A6524'	=	0.60310579

To adjust the angle to the proper section, the appropriate section constant is added to or subtracted from the intermediate result, as follows:

Z$\leq$tan	15°	$\rightarrow$	+ 0 (E'0')
tan	15°<Z$\leq$1	$\rightarrow$	+ $\pi/6$ (X'40860A92')
1/Z$\leq$tan	15°	$\rightarrow$	- $\pi/2$ (X'C11921FB')
tan	15°$\leq$1/Z$\leq$1	$\rightarrow$	- $\pi/3$ (X'C110C152)

We now have the correct angle for the first quadrant. All that remains is to fix the quadrant. If the $\cos(x)<0$, then $\tan^{-1}(X) = \pi - \tan^{-1}(Z)$. That fixes the angle to agree with the sign of $\cos(x)$. Now make the sign of the answer agree with the sign of $\sin(x)$,i.e. $-\tan^{-1}(Z)$ for $-\sin(x)$ and $+\tan^{-1}(Z)$ for $+\sin(x)$. The result, in radians, is in the correct quadrant in the range $(-\pi, +\pi)$.

EATAN2

Secondary Entry Name: ATAN

Function: Computes arctangent by fractional approximation in the range ($-\pi/2$, + $\pi/2$) in single precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`ARCTAN(X), where X is a single precision scalar.`

[ ] Other Library Modules:

Execution Time (microseconds): 116.5

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	Radians

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

For a very small number of input arguments, the ATAN routine will return results that are accurate to 5 significant decimal digits (instead of the 6 significant decimal digits that are generally required for single precision routines). For most of the range of the ATAN routine, results are returned that are accurate to 6 significant decimal digits.

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

Very similar to EATAN2, but the only special case that can be checked is $Z = |\tan(x)| < (16)^{-3}$. If Z is this small, then return Z to avoid an underflow exception later on. The algorithm for reduction and computation of $\tan^{-1}(Z)$ is the same as EATAN2 again until quadrant fixing time. Since ARCTAN has only one arg, the result can only be adjusted in the range ($-\pi/2$, $+\pi/2$). The $\tan^{-1}(Z)$ is computed for the first quadrant.

If the argument, tan(x), is negative, the result is made negative.

EPWRE

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: EPWRE Size of Code Area 32 Hw
Stack Requirement: 22 Hw Data CSECT Size: 4 Hw
[ ] Intrinsic [x] Procedure
Other Library Modules Referenced: EXP,LOG,SQRT

ENTRY POINT DESCRIPTIONS

Primary Entry Name: EPWRE

Function: Exponentiation of a single precision scalar to a single precision scalar power.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
X**Y, where X and Y are single precision scalars.
```

[ ] Other Library Modules:

Execution Time (microseconds): If Y = .5: 124.7
If Y≠ .5: 337.1

Input Arguments:

Type	Precision	How Passed	Units
Scalar (base)	SP	F0	-
Scalar (exponent)	SP	F2	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
4	Zero raised to power ≤0	Return 0
24	Base<0	Use |base|

Comments:

For most of the range, the EPWRE routine will return results that are accurate to 4 significant decimal digits (instead of the 6 significant decimal digits that are generally required for single precision routines).

Except for the detection of errors 4 and 24, no range or overflow checking is performed.

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

If exponent = 0.5 compute $X^{0.5}$ as $\sqrt{X}$ (by using the external SQRT function). Otherwise, $X^Y = e^{Y \ln X}$, using the external EXP and LOG functions.

EPWRI

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: EPWRI Size of Code Area 38 Hw

Stack Requirement: 18 Hw Data CSECT Size: 2 Hw

[ ] Intrinsic [x] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: EPWRI

Function: Exponentiation of a single precision scalar to a double precision integer power.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
X**I, where X is a single precision scalar, and I is a
     double precision integer.
```

[ ] Other Library Modules:

Execution Time (microseconds):

38.2 + (n-1) * 16.2 + 6.0m + 14.2 (if exponent negative), where

n = number of significant digits in binary representation of |exponent|.

m = number of significant 1's in binary representation of |exponent|.

Input Arguments:

Type	Precision	How Passed	Units
Scalar (base)	SP	F0	-
Integer (exponent)	DP	R6	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
4	Zero raised to power $\leq$ 0	Return 0

Comments:

Other than detection of Error 4, no additional range or exponent overflow checking is done. According to the *Space Shuttle Model AP101-S Principles of Operation,* an exponent overflow occurs when the result exponent exceeds 16**63 and an exponent underflow occurs when the result exponent is less than 16**-64. For base ** |exp|, if |base|>1 then an exponent overflow is possible. If exp<0 and 0<|base|<1 then an exponent underflow is possible. If the exponent underflow mask bit is zero (inhibiting the interrupt) then the result is a true zero. This will cause a divide by zero GPC error when the reciprocal of the result is taken. If the exponent underflow mask bit is one (enabling the interrupt) then the operands are unchanged. This could cause an exponent overflow when the reciprocal of the result is taken.

Registers Unsafe Across Call: F0,F1,F2,F3.

Algorithm:

Let I = |exponent|, E = base. Write

$I = \sum_{i=0}^{32} e_i 2^i$, where $e_i = 0$ or 1 for all i.

Then

$E^I = E^{\Sigma_i e_i 2^i} = \pi_i E^{e_i 2^i} = \pi_{e_i=1} E^{2^i}$ if some $e_i = 1$, and = 1 otherwise.

The product $\pi_{e_i=1} E^{2^i}$ is computed in a loop. Each time around the loop, E^{2k} is multiplied by itself to give $E^{2^{k+1}}$. The k+1-st bit is shifted out of the exponent. If it is 1 $E^{2^{k+1}}$,is multiplied into the result. If not, the result is left alone. When the remaining exponent is zero, the loop is finished and the result is E^I. If the exponent was positive, return E^I. Otherwise, return the reciprocal of E^I.

EPWRI

Secondary Entry Name: EPWRH

Function: Exponentiation of a single precision scalar to a single precision integer power.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
X**I, where X is a single precision scalar, and I is a
      single precision integer.
```

[ ] Other Library Modules:

Execution Time (microseconds): same as EPWRI, except constant is 38.8.

Input Arguments:

Type	Precision	How Passed	Units
Scalar (base)	SP	F0	-
Integer (exponent)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
4	Zero raised to power $\leq$ 0	Return 0

Comments:

For most of the range of the EPRWH routine, results are returned that are accurate to 4 significant decimal digits (instead of the 6 significant decimal digits that are generally required for single precision routines).

Other than detection of Error 4, no additional range or exponent overflow checking is done. According to the *Space Shuttle Model AP101-S Principles of Operation*, an exponent overflow occurs when the result exponent exceeds 16**63 and an exponent underflow occurs when the result exponent is less than 16**-64. For base ** |exp|, if |base|>1 then an exponent overflow is possible. If exp<0 and 0<|base|<1 then an exponent underflow is possible. If the exponent underflow mask bit is zero (inhibiting the interrupt) then the result is a true zero. This will cause a divide by zero GPC error when the reciprocal of the result is taken. If the exponent underflow mask bit is one (enabling the interrupt) then the operands are unchanged. This could cause an exponent overflow when the reciprocal of the result is taken.

Registers Unsafe Across Call: F0,F1,F2,F3.

Algorithm:

Halfword exponent is shifted right to convert to a fullword. Then, EPWRI routine is used.

EXP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: EXP Size of Code Area 108 Hw
Stack Requirement: 18 Hw Data CSECT Size: 2 Hw
[ ] Intrinsic [x] Procedure
Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: EXP

Function: Computes e^X in single precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
EXP(X), where X is a single precision scalar
```

[X] Other Library Modules:

TANH, EPWRE, SINH

Execution Time (microseconds): 141.8

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
6	X>174.67308	Return maximum positive floating point number

Comments:

Gives exponent underflow if argument is less than -180.21826 --No RTL error number; GPC error group 3 code 9. The error will not be detected if the program status word masks out the underflow errors.

Registers Unsafe Across Call: F0,F1,F2.

Warning: For values in the range $-180.21826 \le X \le -180.21806$ the EXP routine returns values accurate to only 5 significant digits, instead of the expected 6.

Algorithm:

Let X ($\log_2 e$) = 4R-S-T, where R and S are integers, $0 \le S \le 3$, and $0 \le T < 1$.

Then $\exp(X) = 16^R * 2^{-S} * 2^{-T}$

2^{-T} is computed by a fractional approximation of the form:

$$2^{-T}=1+\frac{2T}{CT^2-T+D+\frac{B}{A+T^2}}$$

The computation is carried out in fixed-point, and the values and scaling of the constants are:

A = X'576AE119'	= 87.417497 at bit 7
B = X'269F8E6B'	= 617.97227 at bit 11
C = X'B9059003'	= -0.03465736 at bit (-4)
D = X'B05CFCE3'	= -9.95459578 at bit 4

The multiplication by 2^{-S} is carried out by shifting right S places, and the multiplication of 16^R is done by adding R to the floating exponent.

IPWRI

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: IPWRI Size of Code Area 46 Hw

Stack Requirement: 18 Hw Data CSECT Size: 2 Hw

[ ] Intrinsic [x] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: IPWRI

Function: Computes double precision integer to positive double precision integer power.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
arg 1 ** arg 2, where arg 1 is a double precision integer
                variable, and arg 2 is a positive double
                precision integer literal.   A literal
                which has a magnitude greater than 32,767
                is considered to be a double precision
```

[ ] Other Library Modules:

Execution Time (microseconds): k+16.4(n-1)+7.0m+0.4(n-2) if n$\geq$2, where k=44.6,
n = # of significant digits in binary representation of arg2,
m = # of significant ones in binary representation of arg2.

Input Arguments:

Type	Precision	How Passed	Units
Integer (base)	DP	R5	-
Integer (exponent)	DP	R6	-

Output Results:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup
4	Zero raised to power $\leq$ 0	Return 0

Comments:

Except for Error 4, no range or overflow checking is performed.

Registers Unsafe Across Call: R5

Algorithm:

Shift all halfwords to convert to fullwords. Let B = base, I = exponent.

Write I= $\sum_{i=0}^{32} e_i 2^i$ where e_i = 1 for each i. Then:

$$B^I = \Pi_i B^{e_i 2^i} = \Pi_{e_i=1} B^{2^i} \text{ if some } e_i = 1\text{, and} = 1 \text{ otherwise.}$$

The product $\Pi_{e_i=1} B^{2^i}$ is computed in a loop. Each time around the loop, B^{2k} is multiplied by itself to give B^{2k+1}. The k+1-st bit is shifted out of the exponent and tested. If it is 1, the partial result is multiplied by B^{2k+1}. If not, the partial result is left as is. When the remaining exponent is 0, the result is E^I and the exit is taken from the loop. The answer is stored in ARG5 to be available after registers are restored.

IPWRI

Secondary Entry Name: IPWRH

Function: Computes double precision integer to positive single precision integer power.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
arg 1 ** arg 2, where arg 1 is a double precision integer
                variable, and arg 2 is a positive single
                precision integer literal.  A literal
                which has a magnitude between 0 and
                32,767 is considered to be a single
                precision
```

[ ] Other Library Modules:

Execution Time (microseconds): Same as for IPWRI, except k = 46.6.

Input Arguments:

Type	Precision	How Passed	Units
Integer (base)	DP	R5	-
Integer (exponent)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup
4	Zero raised to power ≤ 0	Return 0

Comments:

Except for the detection of error 4 no range or overflow checking is performed.

Registers Unsafe Across Call: R5

Algorithm:

See IPWRI.

IPWRI

Secondary Entry Name: HPWRH

Function: Computes single precision integer to positive single precision integer power.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
arg 1 ** arg 2, where arg 1 is a double precision integer
                variable, and arg 2 is a positive single
                precision integer literal.   A literal
                which has a magnitude between 0 and 32,767
                is considered to be a single precision.
```

[ ] Other Library Modules:

Execution Time (microseconds): Same as for IPWRI, except k = 49.4.

Input Arguments:

Type	Precision	How Passed	Units
Integer (base)	SP	R5	-
Integer (exponent)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Integer	SP	R5	-

Errors Detected:

Error #	Cause	Fixup
4	Zero raised to power $\leq$ 0	Return 0

Comments:

Registers Unsafe Across Call: R5

Algorithm:

See IPWRI.

LOG

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: LOG Size of Code Area 80 Hw
Stack Requirement: 18 Hw Data CSECT Size: 2 Hw
[ ] Intrinsic [x] Procedure
Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: LOG

Function: Computes the natural log(X) in single precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
LOG(X), where X is a single precision scalar.
```

[X] Other Library Modules:

ASINH, ATANH, EPWRE, ACOSH

Execution Time (microseconds): 140.5

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
7	Argument outside range X>0	For X<0, return LOG(|X|) For X = 0, return maximum negative floating point number

Comments:

Registers Unsafe Across Call: F0,Fl,F2,F3,F4,F5.

Algorithm:

Write $X = 16^P * 2^{-Q} * M$, where P and Q are integers, $0 \leq Q \leq 3$, and $1/2 \leq M \leq 1$. P, Q, and M are found by fixed-point calculations. Let A = 1, B = 0, if $M > \frac{\sqrt{2}}{2}$, and A = 1/2, B = 1 otherwise.

Let Z=(M-A)/(M+A). Then log(X) = (4P-Q-B)log 2+ log((1+Z)/(1-Z)).

Log((1+Z)/(1-Z)) is computed by an approximation of the form:

$$W+W\left(\frac{RW^2}{S-W^2}\right)$$

where W = 2Z, and the values of the constants are:

R = X'408D8BC7' = 0.55291413
S = X'416A298C' = 6.6351437

SINH

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: SINH   Size of Code Area 82 Hw

Stack Requirement: 18 Hw   Data CSECT Size: 2 Hw

[ ] Intrinsic   [x] Procedure

Other Library Modules Referenced: EXP

ENTRY POINT DESCRIPTIONS

Primary Entry Name: SINH

Function: Computes hyperbolic sine in single precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
SINH(X) where X is a single precision scalar.
```

[ ] Other Library Modules:

Execution Time (microseconds): 1≤|X|≤1.75366E+2: 235.6
2.0394E-4 ≤|X|<1: 80.7
|X|<2.0394E-4: 40.0

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
9	Argument outside range: X|≤175.366	Return maximum positive floating point number

Comments:

Registers Unsafe Across Call: F0,Fl,F2,F3,F4,F5.

Algorithm:

If X<2.04E-4, then sinh(X) = X. If 2.04E-4≤|X|<1, then sinh(X) is computed via a polynomial approximation.

The form of the polynomial is:

$$\sinh(X) = X + C_1X^3 + C_2X^5 + C_3X^7$$

where the values of the constants are:

C_1 = X'402AAAB8' = 0.16666734

C_2 = X'3F221E8C' = 0.008329912

C_3 = X'3DD5D8B3' = .2039399E-3

Otherwise, sinh(|X|) or cosh(|X|) is calculated using EXP. The number V, equal to 0.4995050, is introduced to control rounding errors and the formula is as follows:

$$\sinh(X) = \frac{1}{2v}\left(e^{(X+\log v)} - \frac{v^2}{e^{(X+\log v)}}\right)$$

$$\cosh(X) = \frac{1}{2v}\left(e^{(X+\log v)} - \frac{v^2}{e^{(X+\log v)}}\right)$$

the identities sinh(-X) = -sinh(X) and cosh(-X) = cosh(X) are used to recover sinh(X) and cosh(X) from sinh(|X|) and cosh(|X|).

SINH

Secondary Entry Name: COSH

Function: Computes hyperbolic cosine in single precision.

Invoked By:

- [X] Compiler emitted code for HAL/S construct of the form:
  `COSH(X) where X is a single precision scalar.`
- [ ] Other Library Modules:

Execution Time (microseconds): 228.9

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
9	Argument outside range: $\lvert X\rvert \leq 175.366$	Return maximum positive floating point number

Comments:

Registers Unsafe Across Call: F0,Fl,F2,F3,F4,F5.

Algorithm:

See SINH Algorithm.

SNCS

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: SNCS Size of Code Area 122 Hw

Stack Requirement: 0 Hw Data CSECT Size: 28 Hw

[ ] Intrinsic [x] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: SNCS

Function: Computes sine(X) and cosine(X) in single precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
SIN(X) and COS(X) where X is a single precision scalar
                  that is recognized as the same for
                  both invocations.
```

[ ] Other Library Modules:

Execution Time (microseconds): (TBD)

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	Radians

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-
Scalar (COS result)	SP	F2	-

Errors Detected:

Error #	Cause	Fixup
8	Argument outside range: $\|X\|<\pi_* 2^{18}$	Return $\frac{\sqrt{2}}{2}$

Comments:

For a very small number of input arguments, the SNCS routine will return results that are accurate to 5 significant decimal digits (instead of the 6 significant digits that are generally required for single precision routines). For most of the range of the SNCS routine, results are returned that are accurate to the 6 significant decimal digits.

Called as a library by compiler: uses only fixed-point registers R1 and R3 which are restored at exit from an intrinsic.

The value used in the routine for $\pi_* 2^{18}$ is hex'45C93E10'≈824289.

Registers Unsafe Across Call: R2,R3,F0,Fl,F2,F3,F4.

Algorithm:

Let $|X| * \frac{4}{\pi} = Q + R$, Q an integer, $0 \le R < 1$. Add 4 to Q if the sine is desired and X<0, and add 2 to Q if the cosine is desired. Since sin (-x) = sin (x+π), and cos (x) = $\sin\left(\frac{\pi}{2} + x\right)$ This reduces the problem of computing sin(x) for $x \ge 0$.

Since Q has been adjusted, it is only necessary to compute $\sin\left(\frac{\pi}{4} * (Q + R)\right)$

If Q_0 = Q mod 8, then this is equal to $\sin\left(\frac{\pi}{4} * (Q_0 + R)\right)$.

The eight cases of this yield, through simple trigonometric identities:

Q_0 =	0:	sin	$\left(R * \frac{\pi}{4}\right)$
	1:	cos	$\left((1-R) * \frac{\pi}{4}\right)$
	2:	cos	$\left(R * \frac{\pi}{4}\right)$
	3:	sin	$\left((1-R) * \frac{\pi}{4}\right)$
	4:	-sin	$\left(R * \frac{\pi}{4}\right)$
	5:	-cos	$\left((1-R) * \frac{\pi}{4}\right)$
	6:	-cos	$\left(R * \frac{\pi}{4}\right)$
	7:	-sin	$\left((1-R) * \frac{\pi}{4}\right)$

Let R_0 = R in octants 0, 2, 4, 6 and R_0 = 1-R in octants 1, 3, 5, 7.

We compute sin $(R_0 * \frac{\pi}{4})$ in octants 0, 3, 4, 7 and cos $(R_0 * \frac{\pi}{4})$ in octants 1, 2, 5, 6, and negate the result in octants 4, 5, 6, 7.

Sin ($R_0 * \frac{\pi}{4}$) and cos($R_0 * \frac{\pi}{4}$) are computed by polynomial approximations.

The form of the approximation for sine is:

$$\sin(R_0 * \frac{\pi}{4}) = R_0(a_0 + a_1 R_0^2 + a_2 R_0^4 + a_3 R_0^6)$$

where the values of the constants are:

a_0 = X'40C90FDB' = .78539819
a_1 = X'C014ABBC' =-.080745459
a_2 = X'3EA32F62' = .0024900069
a_3 = X'BD25B368' =-.000035943

The form of the approximation for cosine is:

$$\cos(R_0 * \frac{\pi}{4}) = 1 + a_1 R_0^2 + a_2 R_0^4 + a_3 R_0^6$$

where the values of the constants are:

a_1 = X'C04EF4EE' =-.30842483
a_2 = X'3F40ED0F' = .0158510767
a_3 = X'BE14F17D' =-.000319570

SNCS

Secondary Entry Name: COS

Function: Computes cosine(X) in single precision.

Invoked By:

- [X] Compiler emitted code for HAL/S construct of the form:
  `COS(X), where X is a single precision scalar.`
- [ ] Other Library Modules:

Execution Time (microseconds): $-\pi \leq X \leq \pi$: 124.5
$X > \pi$ or $X < -\pi$: 123.6

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	Radians

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
8	Argument outside range $\|X\|<\pi*2^{18}$	Return $\frac{\sqrt{2}}{2}$

Comments:

For a very small number of input arguments, the COS routine will return results that are accurate to 5 significant decimal digits (instead of the 6 significant decimal digits that are generally required for single precision routines). For most of the range of the COS routine, results are returned that are accurate to 6 significant decimal digits.

The value used in the routine for $\pi * 2^{18}$ is hex'45C93E10'≈824289.

Registers Unsafe Across Call: R2,R3, F0,Fl, F2,F3,F4.

Algorithm:

See SNCS Algorithm.

SNCS

Secondary Entry Name: SIN

Function: Computes sine(X) in single precision.

Invoked By:

- [X] Compiler emitted code for HAL/S construct of the form:
  `SIN(X), where X is a single precision scalar.`
- [ ] Other Library Modules:

Execution Time (microseconds): $-\pi \le X \le \pi$: 124.5
$X > \pi$ or $X < -\pi$: 123.6

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	Radians

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
8	Argument outside range $\|X\|<\pi_* 2^{18}$	Return $\frac{\sqrt{2}}{2}$

Comments:

For a very small number of input arguments, the SIN routine will return results that are accurate to 5 significant digits (instead of the 6 significant decimal digits that are generally required for single precision routines). For most of the range of the SIN routine, results are returned that are accurate to 6 significant decimal digits.

See SNCS comments.

The value used in the routine for $\pi_* 2^{18}$ is hex'45C93E10'≈824289.

Registers Unsafe Across Call: R2,R3, F0,Fl, F2,F3,F4.

Algorithm:

See SNCS Algorithm.

SQRT

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: SQRT Size of Code Area 48 Hw
Stack Requirement: 0 Hw Data CSECT Size: 14 Hw
[x] Intrinsic [ ] Procedure
Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: SQRT

Function: Computes square root in single precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`SQRT(X), where X is a single precision scalar.`

[X] Other Library Modules:

ACOS, ASINH, EPWRE, VV10S3, ACOSH

Execution Time (microseconds): 88.3

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
5	Argument outside range: X$\geq$0	Return SQRT (|X|)

Comments:

Registers Unsafe Across Call: Rl,R4,R5,R6,R7,F0,Fl,F2,F3.

Algorithm:

Write $X = 16^{2P-Q} * M$, where $\frac{1}{16} \leq M < 1$. Then, $\sqrt{X} = 16^{P} * 4^{-Q} * \sqrt{M}$. This fact is used to obtain a good first approximation to $\sqrt{X}$ by approximating $\sqrt{M}$ by a hyperbolic approximation. The form of the approximation is, for Q=0

$\sqrt{M} = a - \dfrac{b}{\left(\frac{C}{2}\right)+\left(\frac{M}{2}\right)}$ [$\frac{C}{2}+\frac{M}{2}$ is to avoid fixed-point overflow for large M]

where the calculations are done in fixed-point.

The values of the constants are:

a	= X'0lAE7D00'	= 1.6815948 at bit 7
b	= X'FF5B02Fl'	= -1.2889728 at bit 7
$\frac{C}{2}$	= X'35CFC610'	= 0.42040325 at bit 0

For Q = 1, $\frac{a}{4}$ and $\frac{b}{4}$ are used instead of a and b.

$\frac{a}{4}$	= X'006B9F40'	= 0.4203987 at bit 7
$\frac{b}{4}$	= X'FFD6C0BD'	= -0.3222432 at bit 7

The first approximation is improved with two passes of the Newton-Raphson iteration. The form of the first is:

$$Y_1 = \frac{1}{2}\left(\frac{X}{Y_0} + Y_0\right)$$

The form of the second, to minimize truncation errors, is:

$$Y_2 = \frac{1}{2}\left(Y_1 + Y_0 - \frac{X}{Y_1}\right) + \frac{X}{Y_1}$$

Y_2 is returned as the answer.

TAN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: TAN Size of Code Area 116 Hw

Stack Requirement: 20 Hw Data CSECT Size: 4 Hw

[ ] Intrinsic [x] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: TAN

Function: Computes tangent in single precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`TAN(X), where X is a single precision scalar.`

[ ] Other Library Modules:

Execution Time (microseconds): 164.0

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	Radians

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
11	Argument outside range: $\|X\|<\pi_*2^{18}$	Return 1
12	Argument too close to singularity of tangent function	Return maximum positive floating point number

Comments:

For a very small number of input arguments, the TAN routine will return results that are accurate to 5 significant decimal digits (instead of the 6 significant decimal digits that are generally required for single precision routines). For most of the range of the TAN routine, results are returned that are accurate to 6 significant decimal digits.

The value used in the routine for π_*2^{18} is hex'45C93E10'≈824289.

Registers Unsafe Across Call: F0,Fl,F2,F3,F4,F5.

Algorithm:

Let $|X| * \frac{4}{\pi} = Q+R$. Q an integer, $0 \le R < 1$. Give the characteristic of $|X|_* \frac{4}{\pi}$ to X'00000008' for later use as a comparand, to determine nearness of X to a singularity.

We have the following table:

Q mod 4	tan(X)
0	$\tan(\|X\|) = \tan\left(R * \frac{\pi}{4}\right)$
1	$\tan(\|X\|) = \cot\left((1-R) * \frac{\pi}{4}\right)$
2	$\tan(\|X\|) = -\cot\left(R * \frac{\pi}{4}\right)$
3	$\tan(\|X\|) = -\tan\left((1-R) * \frac{\pi}{4}\right)$

For Q mod 4 even, let w = R, and for Q mod 4 odd, let w = 1-R. If $|W|<16^{-3}$, then $Q(W)=b_0$. Compute two polynomials in w, as polynomials in $u = \frac{w^2}{2}$:

$$P(w) = w\,(a_0 + u)$$

$$Q(w) = b_0 + b_1 u + b_2 u^2$$

then $\tan(w) = \frac{P(w)}{Q(w)}$, $\cot(w) = \frac{Q(w)}{P(w)}$ and the above table describes how tan(x) is computed. Finally, tan(x) is computed using the identity tan(-x)=-tan(x).

The values of the constants are:

a_0 = X'C1875FDC' =-8.4609032

b_0 = X'ClAC5D33' =-10.7727537

b_1 = X'415B40FD' = 5.7033663

b_2 = X'C028C93' =-.15932077

NOTE: When |x| is close to zero ($|x|<10^{-10}$), zero is returned, and not the value x.

TANH

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: TANH Size of Code Area 58 Hw
Stack Requirement: 18 Hw Data CSECT Size: 0 Hw
[ ] Intrinsic [x] Procedure
Other Library Modules Referenced: EXP

ENTRY POINT DESCRIPTIONS

Primary Entry Name: TANH
Function: Computes hyperbolic tangent in single precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
TANH(X), where X is a single precision scalar.
```

[ ] Other Library Modules:

Execution Time (microseconds): $|X| \leq 2.4414E-4$: 38.2
$2.4414E-4 < |X| \leq 7.0E-1$: 78.7
$7.0E-1 < |X| < 9.011$: 224.4
$9.011 \leq |X|$: 42.4

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,Fl,F2,F3,F4,F5.

Algorithm:

If $|X|>9.011$, return +1 or -1, according to the sign of X.

If $0.7 \leq |X| \leq 9.011$, then (using EXP), $\tanh(|X|)=1-2/(1+e^{2|X|})$.

Restore sign with $\tanh(-X) = -\tanh(X)$. For $|X| \leq 16^{-3}$, $\tanh(X) = X$. For other values of X, $16^{-3} < |X| < 0.7$, use a rational approximation where the values of the constants are:

a = X'EF7EA70' =-.003782895
b = X'C0D08756' =-.81456511
c = X'41278C49' = 2.4717498

6.3.3 Vector/Matrix Routine Descriptions

This subsection presents a class of routines which deal with HAL/S vector/matrix operations. Some of the routines implement HAL/S language built-in functions while others implement HAL/S operators.

MM0DNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM0DNP Size of Code Area 12 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[x] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM0DNP

Function: Moves a double precision scalar to all positions in an n x m partition of a double precision matrix.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$M_{A\ TO\ B,C\ TO\ D}$=S; where M is a double precision matrix, and either or both of the 'TO' subscripts may be replaced by the 'AT' subscript under rules given by matrix types.

[ ] Other Library Modules:

Execution Time (microseconds): 6.8 + n(4.0 + 8.0m)

Input Arguments:

Type	Precision	How Passed	Units
Scalar(s)	DP	F0	-
Integer(n)	SP	R5	-
Integer(m)	SP	R6	-
Integer(outdel)	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
Matrix(n,m)	DP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R3, R4,R5,R6,F0,F1.

Algorithm:

Uses two nested loops:

Outer loop selects row;

Inner loop selects column and moves scalar to current row/column position.

Upon exiting inner loop, 'outdel' is added to pointer to output matrix location.

MM0SNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM0SNP Size of Code Area 10 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[x] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM0SNP

Function: Fill an n x m partition of a single precision matrix with a single precision scalar.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$M_{A\ TO\ B,C\ TO\ D}$ =S; where M is a double precision matrix, and either or both of the 'TO' subscripts may be replaced by the 'AT' subscript under rules given by matrix types.

[ ] Other Library Modules:

Execution Time (microseconds): 6.4 + n(4.4 + 6.4m)

Input Arguments:

Type	Precision	How Passed	Units
Scalar(s)	SP	F0	-
Integer(n)	SP	R5	-
Integer(m)	SP	R6	-
Integer(outdel)	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
Matrix(n,m)	SP	R1→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R3, R4,R5,R6,R7,F0,F1.

Algorithm:

Uses two nested loops, one on n; one on m;

Inner loop selects row and column of result matrix and moves input scalar into location. At exit of inner loop, pointer to matrix element is incremented by outdel, new row is selected, and inner loop is executed again.

MM1DNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM1DNP Size of Code Area 18 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[x] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM1DNP

Function: Moves a partition of a double precision matrix.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$M2=M1_{A\ TO\ B,C\ TO\ D}$ where M1 and M2 are double precision matrices, and either or both of the

$M2_{A\ TO\ B,C\ TO\ D}=M1;$ 'TO' subscripts may be replaced by the 'AT' subscript under rules given by matrix types.

[ ] Other Library Modules:

Execution Time (microseconds): 10.8 +n(5.4 + 12.2m) for n x m matrix moved.

Input Arguments:

Type	Precision	How Passed	Units
Matrix(n,m)	DP	R2→ 0^{th} element	-
Integer(n)	SP	R5	-
Integer(m)	SP	R6	-
Integer(indel, outdel)	DP	R7(indel in highest Hw, outdel in Low Hw)	-

Output Results:

Type	Precision	How Passed	Units
Matrix(n,m)	DP	R1→ 0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3, R4,R5,R6,R7,F0,F1,F2.

Algorithm:

```
Loops on # rows;
   Loops on # columns;
     Load and store current element pointed to by input/output pointers;
         Increment pointers to next row element;
  End column loop;
  Increment input pointer by indel;
  Increment output pointer by outdel;
End row loop;
```

MM1SNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM1SNP Size of Code Area 16 Hw
Stack Requirement: 0 Hw Data CSECT Size: 0 Hw
[x] Intrinsic [ ] Procedure
Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM1SNP

Function: Moves a partition of a single precision matrix.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$M2=M1_{A\ TO\ B,G\ TO\ D}$

$M2_{A\ TO\ B,C\ TO\ D}=M1;$

where M1 and M2 are single precision matrices, and either or both of the 'TO' subscripts may be replaced by the 'AT' subscript under rules given for matrix types.

[ ] Other Library Modules:

Execution Time (microseconds): 10.8 +n(5.4 + 9.4m) for n x m matrix.

Input Arguments:

Type	Precision	How Passed	Units
Matrix(n,m)	SP	R2→0th element	-
Integer(n)	SP	R5	-
Integer(m)	SP	R6	-
Integer(indel, outdel)	DP	R7 (high Hw=indel, Low Hw=outdel)	-

Output Results:

Type	Precision	How Passed	Units
Matrix(n,m)	SP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3, R4,R5,R6,R7,F0,F1.

Algorithm:

```
Loop on # rows;
   Loop on # columns;
      Load and store current element pointed to by input/output pointer;
        Increment pointers to next row;
   End column loop;
   Increment input pointer by indel;
   Increment output pointer by outdel;
End row loop;
```

MM1TNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM1TNP Size of Code Area 16 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[x] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM1TNP

Function: Moves a partition of a double precision matrix and stores it as a single precision matrix.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$M2=M1_{A\ TO\ B,C\ TO\ D}$

$M2_{A\ TO\ B,C\ TO\ D}=M1;$

where M2 is a single precision matrix, and M1 is a double precision matrix. Both or either of the 'TO' subscripts may be replaced by the 'AT' subscript under rules given for matrix types.

[ ] Other Library Modules:

Execution Time (microseconds): 10.4 + n(5.8 + 10.6m) for n x m move.

Input Arguments:

Type	Precision	How Passed	Units
Matrix(n,m)	DP	R2→0th element	-
Integer(n)	SP	R5	-
Integer(m)	SP	R6	-
Integer(indel,outdel)	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
Matrix(n,m)	SP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0,F1,F2.

Algorithm:

```
Loops on # rows;
   Loops on # columns;
      Load long input element pointed to by input pointer;
      Store short into output element pointed to by output pointer;
      Increment pointer to next element;
   End column loops;
   Increment input pointer by indel;
   Increment output pointer by outdel;
End row loop;
```

MM1WNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM1WNP Size of Code Area 18 Hw
Stack Requirement: 0 Hw Data CSECT Size: 0 Hw
[x] Intrinsic [ ] Procedure
Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM1WNP

Function: Moves a partition of a single precision matrix and stores it as a double precision matrix.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$M2=M1_{A\ TO\ B,C\ TO\ D}$

$M2_{A\ TO\ B,C\ TO\ D}=M1;$

where M1 is a single precision matrix. M2 is a double precision matrix. Both or either of the 'TO' subscripts may be replaced by the 'AT' subscripts under rules given for matrix types.

[ ] Other Library Modules:

Execution Time (microseconds): 13.6 + n(5.0 + 11.0m) for n x m move.

Input Arguments:

Type	Precision	How Passed	Units
Matrix(n,m)	SP	R2→0th element	-
Integer(n)	SP	R5	-
Integer(m)	SP	R6	-
Integer(indel,outdel)	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
Matrix(n,m)	DP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3, R4,R5,R6,R7,F0,F1,F2.

Algorithm:

Clears lower half of floating point register pair.
Loops on # rows;
  Loops on # columns;
    Load short input element pointed to by input pointer;
    Store long (with zeroed second word) into output element pointer;
    Increment pointers to next row element;
  End column loop;
  Increment input pointer by indel;
  Increment output pointer by outdel;
End row loop;

MM6DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM6DN Size of Code Area 42 Hw
Stack Requirement: 0 Hw Data CSECT Size: 0 Hw
[x] Intrinsic [ ] Procedure
Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM6DN

Function: Multiplies two double precision matrices.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
M1 M2,where M1 is a m x n double precision matrix⎤
            M2 is a n x ℓ double precision matrix⎦ m,n ℓ ≠ 3;
```

[ ] Other Library Modules:

Execution Time (microseconds): $22.2 + m(10.8 + \ell(21.2 + 27.2n))$

Input Arguments:

Type	Precision	How Passed	Units
Matrix(m,n)	DP	R2→0th element	-
Matrix(n,ℓ)	DP	R3→0th element	-
Integer(m)	SP	R5	-
Integer(n)	SP	R6	-
Integer(ℓ)	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
Matrix(m,ℓ)	DP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3, R4,R5,R6,R7,F0,F1,F2,F3 F4,F5.

Algorithm:

Uses 3 loops:

Innermost (on n) multiplies a row of M1 by a column of M2;
The second loop (on ℓ) resets the column pointer;
The outer loop (on m) resets the row pointer.

MM6D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM6D3 Size of Code Area 32 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[x] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM6D3

Function: Multiplies two 3 x 3 double precision matrices.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
M1 M2, where M1 and M2 are double precision 3 x 3
       matrices.
```

[ ] Other Library Modules:

Execution Time (microseconds): 671.6

Input Arguments:

Type	Precision	How Passed	Units
Matrix(M1)	DP	R2→0th element	-
Matrix(M2)	DP	R3→0th element	-

Output Results:

Type	Precision	How Passed	Units
Matrix(3,3)	DP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3, R4,R5,R6,R7, F0,F1,F2,F3,F4,F5.

Algorithm:

Explicitly multiplies row by column, element by element. Uses BCTB to advance to each new column, and BCTB to advance to each new row.

MM6SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM6SN Size of Code Area 40 Hw
Stack Requirement: 0 Hw Data CSECT Size: 0 Hw
[x] Intrinsic [ ] Procedure
Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM6SN
Function: Multiplies two single precision matrices.

Invoked By

[X] Compiler emitted code for HAL/S construct of the form:

```
M1 M2,where M1 is a m x n double precision matrix⎤
            M2 is a n x ℓ double precision matrix⎦ m,n ℓ ≠ 3;
```

[ ] Other Library Modules:

Execution Time (microseconds): 22.2 + m(10.8 + ℓ (20.2 + 18.0n))

Input Arguments:

Type	Precision	How Passed	Units
Matrix(m,n)	SP	R2→0th element	-
Matrix(n,ℓ)	SP	R3→0th element	-
Integer(m)	SP	R5	-
Integer(n)	SP	R6	-
Integer(ℓ)	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
Matrix(m,ℓ)	SP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0,F1,F2,F3,F4,F5.

Algorithm:

Same as MM6DN.

MM6S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM6S3 Size of Code Area 24 Hw
Stack Requirement: 0 Hw Data CSECT Size: 0 Hw
[x] Intrinsic [ ] Procedure
Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM6S3

Function: Multiplies two 3 x 3 single precision matrices.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
M1 M2, where M1 and M2 are 3 x 3 single precision
matrices.
```

[ ] Other Library Modules:

Execution Time (microseconds): 409.6

Input Arguments:

Type	Precision	How Passed	Units
Matrix(M1)	SP	R2→0^{th} element	-
Matrix(M2)	SP	R3→0^{th} element	-

Output Results:

Type	Precision	How Passed	Units
Matrix(3,3)	SP	R1→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3, R4,R5,R6,R7,F0,F1,F2,F3,F4,F5.

Algorithm:

Same as MM6D3, except the matrices are single precision.

MM11DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM11DN Size of Code Area 16 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[x] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM11DN

Function: Transposes an n x m double precision matrix.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`TRANSPOSE(M)` or M^T where M is an n x m double precision matrix and m and/or n ≠ 3.

[ ] Other Library Modules:

Execution Time (microseconds): 8.0 + m(5.8 + 12.2n)

Input Arguments:

Type	Precision	How Passed	Units
Matrix(n,m)	DP	R2→0th element	-
Integer	SP	R5	-
Integer	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Matrix(m,n)	DP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3, R4,R5,R6,R7,F0,F1,F2.

Algorithm:

Uses two nested loops:

Outer loop selects column of input matrix;

Inner loop moves elements of selected column to corresponding row of result matrix.

MM11D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM11D3 Size of Code Area 22 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[x] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM11D3

Function: Performs transpose of 3 x 3 double precision matrix.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`TRANSPOSE(M)or` M^T `where M is a 3 x 3 double precision matrix.`

[ ] Other Library Modules:

Execution Time (microseconds): 93.6

Input Arguments:

Type	Precision	How Passed	Units
Matrix(3,3)	DP	R2→0^{th} element	-

Output Results:

Type	Precision	How Passed	Units
Matrix(3,3)	DP	R1→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R4,R5,F0,F1,F2,F3,F4,F5.

Algorithm:

Uses loop to load elements of one column into registers, then store into row elements of resultant for each pass through the loop.

MM11SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM11SN Size of Code Area 16 Hw
Stack Requirement: 0 Hw Data CSECT Size: 0 Hw
[x] Intrinsic [ ] Procedure
Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM11SN

Function: Transpose an n x m single precision matrix.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
TRANSPOSE(M)or MT  where M is an n x m single precision
                   matrix and m and/or n ≠ 3.
```

[ ] Other Library Modules:

Execution Time (microseconds): 8.4 + m(5.8 + 9.4n)

Input Arguments:

Type	Precision	How Passed	Units
Matrix(n,m)	SP	R2→0th element	-
Integer(#rows=n)	SP	R5	-
Integer(#columns=m)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Matrix(m,n)	SP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3, R4,R5,R6,R7,F0,F1.

Algorithm:

Uses two nested loops:
Outer loop selects which column of input matrix to use;
Inner loop loads and stores column elements as row elements of result.

MM11S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM11S3 Size of Code Area 18 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[x] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM11S3

Function: Performs transpose of 3 x 3 single precision matrix.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`TRANSPOSE(M)` or M^T where M is a 3 x 3 single precision matrix.

[ ] Other Library Modules:

Execution Time (microseconds): 71.8

Input Arguments:

Type	Precision	How Passed	Units
Matrix(3,3)	SP	R2→0^{th} element	-

Output Results:

Type	Precision	How Passed	Units
Matrix(3,3)	SP	R1→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R4,R5,F0,F1,F2.

Algorithm:

Uses loop to load F0, F1, F2 with columns of input matrix and store them as rows of output matrix for columns 1 → 3, rows 1 → 3.

MM12DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM12DN Size of Code Area 164 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[ ] Intrinsic [x] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM12DN

Function: Find the determinant of an n x n double precision matrix.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

DET(M), where M is an n x n, double precision matrix, and n ≠ 3.

[ ] Other Library Modules:

Execution Time (microseconds): for n=2: 63.2

for $n \geq 4$:

$$\text{minimum time} = 59.4 + 10.2n^2 + \sum_{k=1}^{n-1} (54.8k^2 + 81.2k + 115.6)$$

$$\text{maximum time} = 59.4 + 10.2n^2 + \sum_{k=1}^{n-1} (60.2k^2 + 134.8k + 169.0 + 3.6n)$$

See MM12SN LRD for a description of maximum time vs. minimum time.

Input Arguments:

Type	Precision	How Passed	Units
Matrix(n,n)	DP	R2→0^{th} element	-
Matrix(n,n) workarea	DP	R4	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

Same as MM12SN.

MM12D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM12D3 Size of Code Area 44 Hw
Stack Requirement: 18 Hw Data CSECT Size: 0 Hw
[ ] Intrinsic [x] Procedure
Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM12D3

Function: Find the determinant of a 3 x 3 double precision matrix.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
DET(M), where M is a double precision 3 x 3 matrix.
```

[X] Other Library Modules:

MM14D3

Execution Time (microseconds):

Input Arguments:

Type	Precision	How Passed	Units
Matrix(3,3)	DP	R2→0th element	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

For singular matrices, an expected value of 0 may not be returned due to lost precision.

Also, exponent overflow range checking is not performed.

Algorithm:

Uses direct code, no loops to calculate determinant. See algorithm for MM12S3.

MM12SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM12SN Size of Code Area 138 Hw
Stack Requirement: 20 Hw Data CSECT Size: 0 Hw
[ ] Intrinsic [x] Procedure
Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM12SN

Function: Find the determinant of an n x n single precision matrix.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
DET(M), where M is a n x n single precision matrix, and
        n ≠ 3.
```

[ ] Other Library Modules:

Execution Time (microseconds): for n=2: 44.4
for n≥4:

$$\text{minimum time} = 47.8 + 7.8n^2 + \sum_{k=11}^{n-1} (37.6k^2 + 64.6k + 85.8)$$

$$\text{maximum time} = 47.8 + 7.8n^2 + \sum_{k=11}^{n-1} (41.6k^2 + 105.8k + 3.6n)$$

The minimum time occurs in the event that all matrix elements are positive and where no row or column switching is required at any point of the computation.

The maximum time occurs in the event that all matrix elements require complementing to obtain their absolute value, BIG changes on every comparison, and row and column switching are required at every point in the computation.

Input Arguments:

Type	Precision	How Passed	Units
Matrix(n,n)	SP	R2→0^{th} element	-
Matrix(n,n) workarea	SP	R4	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

```
DET = 1.0
FOR K = 1 TO N1 DO
   BIG = 0
   I1 = J1 = K
   FOR I = K TO N DO
      FOR J = K TO N DO                                find maximal element
         IF ABS(A(I,J))>BIG THEN DO
            BIG = ABS(A(I,J));
            I1 = I;
            J1 = J;
         END
      END
   END
   IF I1 ≠ K THEN DO
      DET = -DET
      FOR J = K TO N SWITCH(A(I1,J),A(K,J))      switch rows
   END
   IF J1 ≠ K THEN DO
      DET = -DET
      FOR I = K TO N SWITCH(A(I,J1),A(J,K));    switch columns
   END
   DET = DET*A(K,K)                          product of diagonal element
   FOR I = K + 1 TO N DO
      TEMP1 = -A(1,K)/A(K,K)                 reduce
      FOR J = K + 1 TO N DO
             A(I,J) = A(I,J) + A(K,J) * TEMP1
      END
   END
END
DET = DET*A(N,N)                             last diagonal element
If dim = 2, then special case:
   DET = A(1,1)*A(2,2)-A(1,2)*A(2,1)
```

MM12S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM12S3 Size of Code Area 26 Hw
Stack Requirement: 18 Hw Data CSECT Size: 0 Hw
[ ] Intrinsic [x] Procedure
Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM12S3
Function: Find the determinant of a single precision 3 x 3 matrix.

Invoked By:
[X] Compiler emitted code for HAL/S construct of the form:

```
DET(M), where M is a single precision 3 x 3 matrix.
```

[X] Other Library Modules:

MM14S3

Execution Time (microseconds): 116.0

Input Arguments:

Type	Precision	How Passed	Units
Matrix(3,3)	SP	R2→0th element	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

Uses direct inline code to calculate

$$\det = M11\ M_{22}\ M_{33} + M_{12}\ M_{23}\ M_{31} + M_{13}\ M_{21}\ M_{32}$$
$$- M31\ M_{22}\ M_{13} - M_{32}\ M_{23}\ M_{11} - M_{33}\ M_{21}\ M_{12}$$

MM13DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM13DN Size of Code Area 10 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[x] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM13DN

Function: Calculates TRACE of an n x n double precision matrix.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
TRACE(M),  where M is an n x n double precision matrix,
           and n ≠ 3.
```

[ ] Other Library Modules:

Execution Time (microseconds): 12.0 + 10.2n

Input Arguments:

Type	Precision	How Passed	Units
Matrix(n,n)	DP	R2→0th element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R2,R4,R5,R6,F0,F1.

Algorithm:

Creates a skip value of n+1; Uses loop counting down n-1 to zero, each pass summing a diagonal element of the matrix by using the skip value to increment from the previous diagonal element.

MM13D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM13D3 Size of Code Area 8 Hw
Stack Requirement: 0 Hw Data CSECT Size: 0 Hw
[x] Intrinsic [ ] Procedure
Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM13D3

Function: Calculates TRACE of a 3 x 3 double precision matrix.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
TRACE(M), where M is a 3 x 3 double precision matrix.
```

[ ] Other Library Modules:

Execution Time (microseconds): 19.8

Input Arguments:

Type	Precision	How Passed	Units
Matrix(3,3)	DP	R2→0th element	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R2,R4,F0,F1.

Algorithm:

Direct code, no loops to calculate $M_{11} + M_{22} + M_{33}$.

MM13SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM13SN Size of Code Area 8 Hw
Stack Requirement: 0 Hw Data CSECT Size: 0 Hw
[x] Intrinsic [ ] Procedure
Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM13SN
Function: Calculates TRACE of an n x n single precision matrix.

Invoked By:
[X] Compiler emitted code for HAL/S construct of the form:

```
TRACE(M),  where M is a single precision n x n matrix,
           n ≠ 3.
```

[ ] Other Library Modules:

Execution Time (microseconds): 8.8 + 6.2n

Input Arguments:

Type	Precision	How Passed	Units
Matrix(n,n)	SP	R2→0th element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:
Registers Unsafe Across Call: R2,R4,R5,R6,F0.

Algorithm:
Creates a skip value of n+1; uses loop counting down n-1 to zero, each pass summing a diagonal element of the matrix by using the skip value to increment from the previous diagonal element.

MM13S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM13S3 Size of Code Area 4 Hw
Stack Requirement: 0 Hw Data CSECT Size: 0 Hw
[x] Intrinsic [ ] Procedure
Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM13S3
Function: Calculates TRACE of a 3 x 3 single precision matrix.

Invoked By:
[X] Compiler emitted code for HAL/S construct of the form:

```
TRACE(M),   where M is a 3 x 3 single precision
            matrix.
```

[ ] Other Library Modules:

Execution Time (microseconds): 9.8

Input Arguments:

Type	Precision	How Passed	Units
Matrix(3,3)	SP	R2→0th element	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R2,R4,F0.

Algorithm:

Straight code to calculate $M_{11} + M_{22} + M_{33}$.

MM14DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM14DN Size of Code Area 288 Hw
Stack Requirement: 28 Hw Data CSECT Size: 2 Hw
[ ] Intrinsic [x] Procedure
Other Library Modules Referenced: MM15DN

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM14DN

Function: Inverts an n x n double precision matrix.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

M^{-1} or INVERSE(M), where M is an n x n double precision matrix, n ≠ 3.

[ ] Other Library Modules:

Execution Time (microseconds): for n=2: 173.8,
for n≥4: $63.0 + 129.5n + 43.0n^2 + 65.4n^3$

Input Arguments:

Type	Precision	How Passed	Units
Matrix(n,n)	DP	R4→0^{th} element	-
Integer(n)	SP	R5	-
Matrix(n,n) workarea	DP	R7	-

Output Results:

Type	Precision	How Passed	Units
Matrix	DP	R2→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
27	Singular matrix	Return identity matrix

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

Same as MM14SN, except that pivot element divide operation is done by multiplying by reciprocal to save time over use of long divide instruction.

MM14D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM14D3 Size of Code Area 144 Hw

Stack Requirement: 26 Hw Data CSECT Size: 2 Hw

[ ] Intrinsic [x] Procedure

Other Library Modules Referenced: MM12D3,MM15DN

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM14D3

Function: Inverts a 3 x 3 double precision matrix.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
M-1 or INVERSE(M), where M is a 3 x 3 double precision
                    matrix.
```

[ ] Other Library Modules:

Execution Time (microseconds): 795.4

Input Arguments:

Type	Precision	How Passed	Units
Matrix(3,3)	DP	R4→0th element	-

Output Results:

Type	Precision	How Passed	Units
Matrix(3,3)	DP	R2→0th element	-

Errors Detected:

Error #	Cause	Fixup
27	Attempted inverse of singular matrix	Return identity matrix

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

For singular matrices, the identity matrix and error message may not be returned due to lost precision in calculating the determinant (refer to MM12D3). A small determinant value could result in exponent overflow during the '1/determinant' calculation. Exponent overflow range checking is not performed.

Algorithm:

Explicit code, no loops; algorithm same as MM14S3 except that external routines used are MM12D3 and MM15DN.

MM14SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM14SN Size of Code Area 246 Hw
Stack Requirement: 20 Hw Data CSECT Size: 2 Hw
[ ] Intrinsic [x] Procedure
Other Library Modules Referenced: MM15SN

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM14SN

Function: Inverts a single precision n x n matrix.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

M^{-1} or INVERSE(M), where M is a single precision n x n matrix, $n \neq 3$.

[ ] Other Library Modules:

Execution Time (microseconds): for n=2: 107.6,
for $n \geq 4$: $52.0 + 39.2n + 10.5n^2 + 54.6n^3$

Input Arguments:

Type	Precision	How Passed	Units
Matrix(n,n)	SP	R4→0^{th} element	-
Integer(n)	SP	R5	-
Matrix(n,n) workarea	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
Matrix(n,n)	SP	R2→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
27	Matrix is singular	Return identity matrix

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

For K = 1, N
  find maximal element in row K to n, columns K to n
  save it as 'BIG' pivot element
  save its row # as ISW(K)
  save its column # as JSW(K)
  switch K^{th} and $ISW(K)^{th}$ row
  switch K^{th} and $JSW(K)^{th}$ column

divide K^{th} column except for K^{th} element by - BIG
reduce matrix
divide K^{th} row except for K^{th} element by BIG
replace pivot by reciprocal

DO K = N-1,1

interchange $JSW(K)^{th}$ and K^{th} rows
interchange $ISW(K)^{th}$ and K^{th} columns.

MM14S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM14S3 Size of Code Area 80 Hw
Stack Requirement: 18 Hw Data CSECT Size: 2 Hw
[ ] Intrinsic [x] Procedure
Other Library Modules Referenced: MM12S3,MM15SN

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM14S3
Function: Inverts a 3 x 3 single precision matrix.

Invoked By:
[X] Compiler emitted code for HAL/S construct of the form:
M^{-1} or INVERSE(M), where M is a 3 x 3 single precision matrix.

[ ] Other Library Modules:

Execution Time (microseconds): 458.8

Input Arguments:

Type	Precision	How Passed	Units
Matrix(3,3)	SP	R4$\rightarrow$0th element	-

Output Results:

Type	Precision	How Passed	Units
Matrix(3,3)	SP	R2$\rightarrow$0th element	-

Errors Detected:

Error #	Cause	Fixup
27	Attempted inverse of singular matrix	Return an identity matrix

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

Explicit code, no loops to calculate:

inverse $M = \frac{\mathrm{adj}(M)}{|M|}$, where $\mathrm{adj}M_{i,j} = \det M_{i\neq 3, j\neq 3}$ and $|M| = \det M$

uses external determinant routine (MM12S3) and in event of determinant of zero, calls identity matrix routine (MM15SN).

MM15DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM15DN Size of Code Area 18 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM15DN

Function: Creates an n x n double precision identity matrix.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

M^0, where M is an n x n double precision matrix.

[X] Other Library Modules:

MM14DN, MM14D3

Execution Time (microseconds): $15.6 + 5.0n + 11.2n^2$

Input Arguments:

Type	Precision	How Passed	Units
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Matrix(n,n)	DP	R1→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R4,R5,R6,R7,F0,F1,F2,F3.

Algorithm:

Uses two nested loops, each counting 1 to n.

Inner loop compares both loop indices; if equal, stores 1.0 at current row/column position; otherwise stores 0.0.

MM15SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM15SN Size of Code Area 14 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM15SN

Function: Creates an n x n identity matrix.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`M`^{0}`, where M is a single precision n x n matrix.`

[X] Other Library Modules:

MM14SN, MM14S3

Execution Time (microseconds): $10.0 + 5.2n + 9.6n^2$

Input Arguments:

Type	Precision	How Passed	Units
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Matrix(n,n)	SP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R4,R5,R6,R7,F0,F2.

Algorithm:

Uses two nested loops, each counting 1 to n.

Inner loop checks both loop indices; if equal, stores 1.0 at current row/column position; otherwise stores 0.0.

MM17D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM17D3 Size of Code Area 80 Hw

Stack Requirement: 20 Hw Data CSECT Size: 0 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM17D3

Function: Raises a 3 x 3 double precision matrix to a power.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

M^I, where M is a 3 x 3 double precision matrix and I is an integer literal >1.

[ ] Other Library Modules:

Execution Time (microseconds):

Exponent=2: 991.6

Exponent>2: 1071.2 * (# of significant zeros in exponent)
+2137.2 * (# of ones in exponent)
-2105.8

Input Arguments:

Type	Precision	How Passed	Units
Matrix(3x3)	DP	R4→0^{th} element	-
Integer(power)	SP	R6	-
Matrix(3,3) workarea	DP	R7	-

Output Results:

Type	Precision	How Passed	Units
Matrix(3,3)	DP	R2→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

Loads R5 with literal 3 and drops in MM17DN code.

MM17D3

Secondary Entry Name: MM17DN

Function: Raises an n x n double precision matrix to a power.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
M^I,  where M is an n x n double precision matrix and I
      is an integer literal >1.
```

[ ] Other Library Modules:

Execution Time (microseconds):

$27.8n^3 + 19.4n^2 + 6.2n + 43.4$ if power = 2.

124.2 + TMULT(KA) + TMOVE(KA-1) + 8.6KB + 3.4 KC if power>2.

where:

TMULT	=	$9.6 + 6.2n + 19.4n^2 + 27.8n^3$
KA	=	(((# significant 1s in exponent)-1)*2) + (# of significant 0s in exponent)
TMOVE	=	$10.2 + 11.0n^2$
KB	=	total number of significant 1s and 0s in exponent
KC	=	# of significant 1s in exponent.

Input Arguments:

Type	Precision	How Passed	Units
Matrix(n,n)	DP	R4→0^{th} element	-
Integer(n)	SP	R5	-
Integer(power)	SP	R6	-
Matrix(n,n) workarea	DP	R7	-

Output Results:

Type	Precision	How Passed	Units
Matrix(n,n)	DP	R2→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

Same as MM17SN.

MM17S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MM17S3 Size of Code Area 78 Hw
Stack Requirement: 20 Hw Data CSECT Size: 0 Hw
[ ] Intrinsic [X] Procedure
Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MM17S3
Function: Raises a 3 x 3 single precision matrix to a power.

Invoked By:
[X] Compiler emitted code for HAL/S construct of the form:

M^I, where M is a 3 x 3 single precision matrix and I is an integer literal >1.

[ ] Other Library Modules:

Execution Time (microseconds):

Exponent=2: 623.6
Exponent>2: 681.6 * (# significant zeros in exponent)
+1358.0 * (# ones in exponent)
-1305.0

Input Arguments:

Type	Precision	How Passed	Units
Matrix(3x3)	SP	R4→0^{th} element	-
Integer(power)	SP	R6	-
Matrix(3,3) workarea	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
Matrix(3,3)	SP	R2→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,F2,F3,F4,F5.

Algorithm:

Loads R5 with literal 3 and drops into MM17SN code.

MM17S3

Secondary Entry Name: MM17SN

Function: Raises an n x n single precision matrix to a power.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
M^I,  where M is a n x n single precision matrix and I is
      an integer literal >1.
```

[ ] Other Library Modules:

Execution Time (microseconds):

if power=2: then $15.6n^3 + 15.2n^2 + 5.8n + 43.8$

if power>2: same as in MM17DN except

$TMULT = 10.0 + 5.8n + 15.2n^2 + 15.6n^3$

$TMOVE = 10.2 + 8.6n^2$

Input Arguments:

Type	Precision	How Passed	Units
Matrix(n,n)	SP	R4→0^{th} element	-
Integer(n)	SP	R5	-
Integer(power)	SP	R6	-
Matrix(n,n)workareas	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
Matrix(n,n)	SP	R2→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,F2,F3,F4,F5.

Algorithm:

Let A = original matrix, R = result matrix, T = temporary matrix.

1. R = A A
2. locate first one bit in exponent, remove it, remember bit position
3. go to step 6
4. T = R
5. R = T T
6. Remove exponent bit at current position, increment position. If bit was 0 go to step 9.
7. T = R
8. R = T A
9. If any bits left in exponent, go to step 4, otherwise R is complete.

MV6DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MV6DN Size of Code Area 24 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MV6DN

Function: Multiplies a double precision m x n matrix by a length n double precision vector.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
M V,  where M is a double precision m x n matrix, V is
      a length n double precision vector, and m and/or
      n ≠ 3.
```

[ ] Other Library Modules:

Execution Time (microseconds):

Input Arguments:

Type	Precision	How Passed	Units
Matrix(m,n)	DP	R2→0th element	-
Vector(n)	DP	R3→0th element	-
Integer(m)	SP	R5	-
Integer(n)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Vector(m)	DP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0,F1,F2,F3,F4,F5.

Algorithm:

Uses 2 nested loops, outer loop selecting rows of matrix, inner loop summing products of vector elements with current row elements.

MV6D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MV6D3 Size of Code Area 24 Hw
Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MV6D3

Function: Multiplies a double precision 3 x 3 matrix by a length 3 double precision vector.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
M V, where M is a double precision 3 x 3 matrix and V is
     a double precision length 3 vector.
```

[ ] Other Library Modules:

Execution Time (microseconds): 304.4

Input Arguments:

Type	Precision	How Passed	Units
Matrix(3,3)	DP	R2→0th element	-
Vector(3)	DP	R3→0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	DP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0,F1,F2,F3.

Algorithm:

Uses 2 nested loops, outer loop selecting rows of matrix, inner loop summing products of vector elements with current row elements.

MV6SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MV6SN Size of Code Area 18 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MV6SN

Function: Multiplies a single precision m x n matrix by a length n single precision vector.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
M V,  where M is an m x n single precision matrix.  V is
      a length n single precision vector, m and/or n ≠ 3.
```

[ ] Other Library Modules:

Execution Time (microseconds): 11.2 + m(11.0 + 18.4n)

Input Arguments:

Type	Precision	How Passed	Units
Matrix(m,n)	SP	R2→ 0^{th} element	-
Vector(n)	SP	R3→ 0^{th} element	-
Integer(m)	SP	R5	-
Integer(n)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Vector(m)	SP	R1→ 0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0,F1,F2,F3,F4,F5.

Algorithm:

Uses 2 nested loops, outer loop selecting rows of matrix, inner loop summing products of vector elements with current row elements.

MV6S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MV6S3 Size of Code Area 30 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: MV6S3

Function: Multiplies a 3 x 3 single precision matrix by a length 3 single precision vector.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
M V,  where M is a single precision 3 x 3 matrix, and V
      is a single precision length 3 vector.
```

[ ] Other Library Modules:

Execution Time (microseconds): 137.6

Input Arguments:

Type	Precision	How Passed	Units
Matrix(3,3)	SP	R2→0th element	-
Vector(3)	SP	R3→0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	SP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,F0,F1,F2,F3.

Algorithm:

The product of each vector element and the current row element is summed and stored in the proper element output vector.

VM6DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VM6DN Size of Code Area 26 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VM6DN

Function: Multiplies length n double precision vector and n x m double precision matrix.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
V M,  where V is a double precision length n vector, n ≠
      3, and M is a double precision n x m matrix, n ≠ 3.
```

[ ] Other Library Modules:

Execution Time (microseconds): 23.2 + m(23.2 + 27.6n)

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	DP	R2→0th element	-
Matrix(n,m)	DP	R3→0th element	-
Integer(m)	SP	R6	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(m)	DP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0,F1,F2,F3,F4,F5.

Algorithm:

Uses two nested loops:

Outer loop selects matrix column.

Inner loop sums products of vector elements with matrix column elements.

VM6D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VM6D3 Size of Code Area 24 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VM6D3

Function: Multiplies a length 3 double precision vector by a 3 x 3 double precision matrix.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
V M,  where V is a double precision length 3 vector, and M
      is a double precision 3 x 3 matrix.
```

[ ] Other Library Modules:

Execution Time (microseconds): 227.8

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	DP	R2→0^{th} element	-
Matrix(3,3)	DP	R3→0^{th} element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	DP	R1→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,F0,F1,F2,F3,F4,F5.

Algorithm:

Saves pointer to input vector (R2) so that R2 can be used to address both input vector and matrix by appropriate loading.

Loops 3 times:

Loads elements of vector into F0,F2,F4;
Switches R2 to point to matrix;
Sums products of column elements with vector elements;
Restores R2 to point at vector;
Makes next pass.

VM6SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VM6SN Size of Code Area 22 Hw
Stack Requirement: 0 Hw Data CSECT Size: 0 Hw
[X] Intrinsic [ ] Procedure
Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VM6SN

Function: Multiply a length n single precision vector by a n x m single precision matrix.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
V M,  where V is a single precision n-vector, n ≠ 3.
      M is an n x m matrix, n ≠ 3.
```

[ ] Other Library Modules:

Execution Time (microseconds): 12.4 + m(19.2 + 18.2n)

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	SP	R2→0^{th} element	-
Matrix(n,m)	SP	R3→0^{th} element	-
Integer(m)	SP	R6	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(m)	SP	R1→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0,F1,F2,F3,F4,F5.

Algorithm:

Uses two nested loops; outer loop selecting matrix column, inner loop performs summation of products of vector elements and matrix column elements.

VM6S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VM6S3 Size of Code Area 16 Hw
Stack Requirement: 0 Hw Data CSECT Size: 0 Hw
[X] Intrinsic [ ] Procedure
Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: VM6S3

Function: Multiplies a length 3 single precision vector by a 3 x 3 single precision matrix.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
V M,  where V is a single precision length 3 vector.  M is
      a single precision 3 x 3 matrix.
```

[ ] Other Library Modules:

Execution Time (microseconds): 141.2

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	SP	R2→0th element	-
Matrix(3,3)	SP	R3→0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	SP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,F0,F1,F2,F3.

Algorithm:

Uses one loop, looping three times, each pass addressing new column of matrix for explicit multiplication and summing, by elements of vector and storing into result. R1 is setup to contain both input matrix and output vector pointer in its two halves. Then circular shifts are used to place appropriate pointer into high Hw for use as base.

VO6DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VMO6DN Size of Code Area 20 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VO6DN

Function: Performs vector outer product of two double precision vectors.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
V1 V2, where V1 and V2 are double precision vectors of
       length n and m respectively, where n and/or m ≠ 3.
```

[ ] Other Library Modules

Execution Time (microseconds): 12.8 + n(5.8 + 24.4m)

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	DP	R2 | 0^{th} element	-
Vector(m)	DP	R3→ 0^{th} element	-
Integer(n)	SP	R5	-
Integer(m)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Matrix(n,m)	DP	R1 → 0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0,F1,F4.

Algorithm:

Uses two loops based on the dimensions of the vectors:

Inner loop (indexing on 'n') multiplies element of V1 by each element of V2 creating a row of result matrix.

Outer loop (indexing on 'm') moves to next element of V1 and moves pointer to next row of result matrix.

VO6D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VMO6D3 Size of Code Area 22 Hw
Stack Requirement: 0 Hw Data CSECT Size: 0 Hw
[X] Intrinsic [ ] Procedure
Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VO6D3

Function: Computes vector outer product of length 3 double precision vectors.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
V1 V2,  where V1 and V2 are double precision length 3
        vectors.
```

[ ] Other Library Modules:

Execution Time (microseconds): 251.0

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	DP	R2→0th element	-
Vector(3)	DP	R3→0th element	-

Output Results:

Type	Precision	How Passed	Units
Matrix(3,3)	DP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,F0,F1.

Algorithm:

Same algorithm as VO6DN except that loop extents are set to literally 3.

```
DO FOR I = 3 TO 1 BY -1;
    DO FOR J = 3 TO 1 BY -1;
       M$(I,J) = V1$(I) V2$(J);
    END;
END;
```

VO6SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VMO6SN Size of Code Area 20 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VO6SN

Function: Calculates vector outer product of 2 single precision vectors.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
V1 V2, where V1 and V2 are single precision vectors of
       length n and m respectively, where n and/or m ≠ 3.
```

[ ] Other Library Modules:

Execution Time (microseconds): 14.2 + n(5.8 + 14.4m)

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	SP	R2→0^{th} element	-
Vector(m)	SP	R3→0^{th} element	-
Integer(n)	SP	R5	-
Integer(m)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Matrix(n,m)	SP	R1→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0,F1,F4,F5.

Algorithm:

Same as VO6DN

VO6S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VMO6SN Size of Code Area 20 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VO6S3

Function: Calculates vector outer product of 2 single precision length 3 vectors.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
V1 V2, where V1 and V2 are single precision length 3
       vectors.
```

[ ] Other Library Modules:

Execution Time (microseconds): 160.6

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	SP	R2→0^{th} element	-
Vector(3)	SP	R3→0^{th} element	-

Output Results:

Type	Precision	How Passed	Units
Matrix(3,3)	SP	R1→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,F0,F1.

Algorithm:

Same algorithm as VO6DN except that loop extents are set to literally 3.

VV0DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV0DN Size of Code Area 6 Hw
Stack Requirement: 0 Hw Data CSECT Size: 0 Hw
[X] Intrinsic [ ] Procedure
Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV0DN
Function: Initializes all elements of a double precision vector of length n, to the null vector (0).

Invoked By:
[X] Compiler emitted code for HAL/S construct of the form:

```
V=0, where V is a double precision vector.
```

[X] Other Library Modules:

VV10D3

Execution Time (microseconds): 7.0 + 5.1n

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-
Integer(n)	SP	F5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	DP	R1$\rightarrow$0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:
Registers Unsafe Across Call: R1,R4,R5,F0,F1.

Algorithm:
Uses loop counting down length (n). Stores zero into one element of vector on each pass through loop.

VV0DNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV0DNP Size of Code Area 6 Hw
Stack Requirement: 0 Hw Data CSECT Size: 0 Hw
[X] Intrinsic [ ] Procedure
Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV0DNP

Function: Fills a column of a double precision matrix with zeros.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$M_{*,n}$= 0, where M is a double precision matrix.

[ ] Other Library Modules:

Execution Time (microseconds): 7.0 + 7.2n

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-
Integer(outdel)	SP	R7	-
Integer(length)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(length)	DP	R1→ 0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R4,R5,R7,F0,F1.

Algorithm:

Loops 'length' times;

Each pass through loop stores zero into vector element pointed to by R1 and then increments R1 by outdel.

VV0SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV0SN Size of Code Area 6 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV0SN

Function: Generates a vector of length n, all of whose elements are zero, i.e. null vector.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
V = 0, where V is a single precision vector.
```

[X] Other Library Modules:

VV10S3

Execution Time (microseconds): 7.0 + 5.6n

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	SP	R1→ 0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R4,R5,F0.

Algorithm:

Uses loop counting down length of vector (n); Stores zero into one element of vector on each pass through loop.

VV0SNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV0SNP Size of Code Area 6 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV0SNP

Function: Moves a single precision scalar to all elements of a column of a single precision matrix.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$M_{*,n}$= S; where M is a single precision matrix and S is a single precision scalar.

[ ] Other Library Modules:

Execution Time (microseconds): 7.0 + 6.0n

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-
Integer(outdel)	SP	R7	-
Integer(length)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(length)	SP	R1→ 0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R4,R5,R7,F0,F1.

Algorithm:

Loops 'length' times;

Each pass through loop stores input scalar into vector element pointed to by R1 and then increments R1 by outdel.

VV1DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV1DN Size of Code Area 8 Hw
Stack Requirement: 0 Hw Data CSECT Size: 0 Hw
[X] Intrinsic [ ] Procedure
Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV1DN
Function: Moves a length n double precision vector. Also used to move matrices.

Invoked By:
[X] Compiler emitted code for HAL/S construct of the form:

```
X = Y,  where X is a length n double precision vector, Y
        is a length n double precision vector, n ≠ 3.
```

[ ] Other Library Modules:

Execution Time (microseconds): 4.2 + 10.2n

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	DP	R2→ 0^{th} element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	DP	R1→ 0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:
Registers Unsafe Across Call: R1,R2,R4,R5,F0,F1.

Algorithm:
Loop n times; using indexing, BCTB on length; load and store each element, last element first.

VV1D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV1D3 Size of Code Area 14 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV1D3

Function: Moves a length 3 partition of a double precision vector.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$X_{A\ TO\ B} = Y_{C\ TO\ D}$, where $X_{A\ TO\ B}$ and $Y_{C\ TO\ D}$ are length 3 partitions of double precision vectors.

[ ] Other Library Modules:

Execution Time (microseconds): 25.2

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	DP	R2→ 0^{th} element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	DP	R1→ 0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R4,F0,F1,F2,F3,F4,F5.

Algorithm:

Load, then store each element.

VV1D3P

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV1D3P Size of Code Area 18 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV1D3P

Function: Moves a length 3 double precision vector or row or column of a matrix to a vector or row or column of a matrix.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$V=M_{A\ TO\ B,\ C}$, where M is a double precision matrix, A TO B is a length 3 partition, and V is a 3-vector.

[ ] Other Library Modules:

Execution Time (microseconds):

46.0 if neither input nor output is contiguous.

48.4 if either input or output is contiguous.

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	DP	R2→ 0^{th} element	-
Integer(indel)	SP	R6	-
Integer(outdel)	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	DP	R1→ 0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Performs single setup of size and then uses VV1DNP.

Registers Unsafe Across Call: R1,R2,R4,R5,R6,R7,F0,F1.

Algorithm:

Initialize R5 with literal 3; Fall into VV1DNP routine; R6, R7 specify distance in Hw between input and output vector elements, respectively.

VV1D3P

Secondary Entry Name: VV1DNP

Function: Moves a length n double precision vector or row or column of a matrix to a row or column vector.

[X] Compiler emitted code for HAL/S construct of the form:

$V=M_{A\ TO\ B,\ C}$, where M is a double precision matrix, A TO B is a length n partition, and V is an n-vector, $n \neq 3$.

[ ] Other Library Modules:

Execution Time (microseconds):

11.4n + 10.2 if neither input nor output is contiguous.

11.4n + 12.6 if either input or output is contiguous.

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	DP	R2→0th element	-
Integer(indel)	SP	R6	-
Integer(outdel)	SP	R7	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	DP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R4,R5,R6,R7,F0,F1.

Algorithm:

Tests outdel, if 0, sets it to 4 (halfwords); Tests indel, if 0, sets it to 4 (halfwords); Loops 'length' times, adding indel to input pointer and outdel to output pointer each time. Each loop moves current input element to current output element.

VV1SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV1SN Size of Code Area 8 Hw
Stack Requirement: 0 Hw Data CSECT Size: 0 Hw
[X] Intrinsic [ ] Procedure
Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV1SN
Function: Moves a length n partition of a single precision vector. Also used to move matrices.

Invoked By:
[X] Compiler emitted code for HAL/S construct of the form:

$V1_{A\ TO\ B}=V2_{C\ TO\ D}$, where $V1_{A\ TO\ B}$ and $V2_{C\ TO\ D}$ are length n partitions of single precision vectors, $n \neq 3$.

[ ] Other Library Modules

Execution Time (microseconds): 4.2 + 7.8n

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	SP	R2→0th element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	SP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:
Registers Unsafe Across Call: R1,R2,R4,R5,F0,F1.

Algorithm:
Loop n times using indexing and BCTB on length. Load, then store each element, last element first.

VV1S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV1S3 Size of Code Area 8 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV1S3

Function: Moves a length 3 partition of a single precision vector.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$V1_{A\ TO\ B}=V2_{A\ TO\ B}$, where $V1_{A\ TO\ B}$ and $V2_{A\ TO\ B}$ are length 3 partitions of single precision vectors.

[ ] Other Library Modules

Execution Time (microseconds): 16.8

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	SP	$R2 \rightarrow 0^{th}$ element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	SP	$R1 \rightarrow 0^{th}$ element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R4,F0,F1,F2,F3,F4,F5.

Algorithm:

Simple Load-Store sequence for each element.

VV1S3P

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV1S3P Size of Code Area 14 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV1S3P

Function: Moves a single precision 3-vector (or row or column of a matrix) to a 3-vector (or row or column of a matrix), when elements are not contiguous.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$V=M_{A\ TO\ B,C}$, where M is a single precision matrix, A TO B is a length 3 partition, and V is a 3-vector.

[ ] Other Library Modules:

Execution Time (microseconds):

38.4 if neither input nor output is contiguous.
40.8 if either input or output is contiguous.

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	SP	R2→0th element	-
Integer(indel)	SP	R6	Hw
Integer(outdel)	SP	R7	Hw

Output Results:

Type	Precision	How Passed	Units
Vector(3)	SP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Performs simple setup of size for use by VV1SNP code.
Registers Unsafe Across Call: R1,R2,R4,R5,R6,R7,F0,F1.

Algorithm:

Initialize R5 with literal 3; Fall into VV1SNP routine; R6, R7 specify distance in Hw between input and output vector elements respectively.

VV1S3P

Secondary Entry Name: VV1SNP

Function: Moves a length n single precision vector or row or column of a matrix to a row or column vector.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$V=M_{A\ TO\ B,C}$, $M_{A\ TO\ B,C}=V$, or $M_{A\ TO\ B,C}=N_{A\ TO\ B,C}$ where M and N are single precision matrices, A TO B is a length n partition, and V is an n-vector, $n \neq 3$.

[ ] Other Library Modules:

Execution Time (microseconds):

8.6n + 10.2 if neither input nor output is contiguous.

8.6n + 12.6 if either input or output is contiguous.

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	SP	$R2 \rightarrow 0^{th}$ element	-
Integer(indel)	SP	R6	Hw
Integer(outdel)	SP	R7	Hw
Integer(length)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	SP	$R1 \rightarrow 0^{th}$ element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R4,R5,R6,R7,F0,F1.

Algorithm:

Tests outdel, if 0, sets it to 2 (halfwords).

Tests indel, if 0, sets it to 2 (halfwords).

Loops 'length' times, adding indel to input pointer and outdel to output pointer each time. Each loop moves current input element to current output element.

VV1TN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV1TN Size of Code Area 8 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV1TN

Function: Moves a length n partition of a double precision vector and converts it to single precision. Also used to move matrices.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$V1_{A\ TO\ B}=V2_{C\ TO\ D}$, where $V2_C$ is a length n partition of a double precision vector and $V1_{A\ TO\ B}$ is a length n partition of a single precision vector, $n \neq 3$.

[ ] Other Library Modules

Execution Time (microseconds): 4.2 + 9.0n

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	DP	R2→0^{th} element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	SP	R1→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R4,R5,F0,F1.

Algorithm:

Using indexing and BCTB on length, loops, loading long and storing short, last element first.

VV1T3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV1T3 Size of Code Area 12 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV1T3

Function: Moves a length 3 partition of a double precision vector and converts it to single precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$V1_{A\ TO\ B}=V2_{C\ TO\ D}$, where $V1_{A\ TO\ B}$ is a length 3 partition of a single precision vector and $V2_{C\ TO\ D}$ is a length 3 partition of a double precision vector.

[ ] Other Library Modules

Execution Time (microseconds): 21.2

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	DP	R2→0^{th} element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	SP	R1→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R4,F0,F1,F2,F3,F4,F5.

Algorithm:

Simple Load/Store for each element.

VV1T3P

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV1T3P   Size of Code Area 14 Hw

Stack Requirement: 0 Hw   Data CSECT Size: 0 Hw

[X] Intrinsic   [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV1T3P

Function: Moves a length 3 partition of a double precision vector or row or column of a matrix to a single precision vector or row or column of a matrix, when elements are not contiguous.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$V=M_{A\ TO\ B,C}$, where V is a single precision 3-vector and M is a length 3 partition of a double precision matrix $_{A\ TO\ B,C}$

[ ] Other Library Modules:

Execution Time (microseconds):

38.4 if neither input nor output is contiguous.
40.8 if either input or output is contiguous.

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	DP	R2→0th element	-
Integer(indel)	SP	R6	-
Integer(outdel)	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	SP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R4,R5,R6,R7,F0,F1.

Algorithm:

Loads R5 with literal 3,
Falls into VV1TNP routine.

VV1T3P

Secondary Entry Name: VV1TNP

Function: Moves a length n partition of a double precision vector or row or column of a matrix to a single precision length n row or column vector, when elements are not contiguous.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$V=M_{A\ TO\ B,C}$, where V is a length n single precision vector and $M_{A\ TO\ B,C}$ is a length n partition of a double precision matrix, $n \neq 3$.

[ ] Other Library Modules:

Execution Time (microseconds):

8.6n + 10.2 if neither input nor output is contiguous
8.6n + 12.6 if either input or output is contiguous.

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	DP	R2→0th element	-
Integer(outdel)	SP	R7	-
Integer(indel)	SP	R6	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	SP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R4,R5,R6,R7,F0,F1.

Algorithm:

If outdel=0, set to 2 (halfwords);
If indel=0, set to 4 (halfwords);
Loops 'length' times, adding indel to input pointer and outdel to output pointer each time. Each loop moves current input element to current output element.

VV1WN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV1WN Size of Code Area 10 Hw
Stack Requirement: 0 Hw Data CSECT Size: 0 Hw
[X] Intrinsic [ ] Procedure
Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV1WN
Function: Moves a length n partition of a single precision vector and converts it to a length n partition of a double precision vector.

Invoked By:
[X] Compiler emitted code for HAL/S construct of the form:

$V1_{A\ TO\ B}=V2_{C\ TO\ D}$, where $V2_C$ is a length n partition of a single precision vector, and $V1_{A\ TO\ B}$ is a length n partition of a double precision vector, $n \neq 3$.

[ ] Other Library Modules:

Execution Time (microseconds): 8.4 + 9.0n

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	SP	R2→0th element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	DP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:
Registers Unsafe Across Call: R1,R2,R4,R5,F0,F1.

Algorithm:
Clear F0,F1. Loop using indexing on BCTB, last element first.
Load short element into F0, Store long F0/F1 element.

VV1W3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV1W3 Size of Code Area 12 Hw
Stack Requirement: 0 Hw Data CSECT Size: 0 Hw
[X] Intrinsic [ ] Procedure
Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV1W3
Function: Moves a length 3 partition of a single precision vector and converts it to a length 3 partition of a double precision vector.

Invoked By:
[X] Compiler emitted code for HAL/S construct of the form:

$V1_{A\ TO\ B}=V2_{C\ TO\ D}$, where $V2_{C\ TO\ D}$ is a length 3 partition of a single precision vector, and $V1_{A\ TO\ B}$ is a length 3 partition of a double precision vector.

[ ] Other Library Modules:

Execution Time (microseconds): 23.8

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	SP	R2→0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	DP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:
Registers Unsafe Across Call: R1,R2,R4,F0,F1.

Algorithm:
Clear F1;
Then explicit code to load (SP) each element of input vector and store (DP) into each element of result vector.

VV1W3P

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV1W3P Size of Code Area 18 Hw
Stack Requirement: 0 Hw Data CSECT Size: 0 Hw
[X] Intrinsic [ ] Procedure
Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV1W3P

Function: Moves a length 3 partition of a single precision vector or row or column of a matrix, to a double precision vector or row or column of a matrix, when elements are not contiguous.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$V=M_{A\ TO\ B,C}$, where V is a double precision 3-vector and $M_{A\ TO\ B,C}$ is a length 3 partition of a single precision matrix.

[ ] Other Library Modules:

Execution Time (microseconds):

44.8 if neither input nor output is contiguous.
47.2 if either input or output is contiguous.

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	SP	R2→0^{th} element	-
Integer(indel)	SP	R6	-
Integer(outdel)	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	DP	R1→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Sets up length for use by VV1WNP.
Registers Unsafe Across Call: R1,R2,R4,R5,R6,R7,F0,F1.

Algorithm:

Loads R5 with literal 3, falls into VV1SNP.

VV1W3P

Secondary Entry Name: VV1WNP

Function: Moves a length n partition of a single precision row or column of a matrix to a double precision vector or row or column of a matrix, when elements are not contiguous.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$V=M_{A\ TO\ B,C}$, where V is a length n double precision vector and $M_{A\ TO\ B,C}$ is a length n partition of a single precision matrix, $n \neq 3$.

[ ] Other Library Modules:

Execution Time (microseconds):

10.2n + 15.0 if either input or output is contiguous.
10.2n + 12.6 if neither input nor output is contiguous.

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	SP	R2→0^{th} element	-
Integer(outdel)	SP	R7	Hw
Integer(indel)	SP	R6	Hw
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	DP	R1→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R4,R5,R6,R7,F0,F1.

Algorithm:

Clears F1. If outdel=0, set to 4 (halfwords); if indel=0, set to 2 (halfwords); Loop 'length' times, adding indel to input pointer and outdel to output pointer each time. Each loop moves current input element to current output element.

VV2DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV2DN   Size of Code Area 14 Hw

Stack Requirement: 0 Hw   Data CSECT Size: 0 Hw

[X] Intrinsic   [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV2DN

Function: Add two double precision vectors of length n. Also used to add two matrices.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
V1+V2, where V1 and V2 are double precision vectors of
       length ≠ 3.
```

[ ] Other Library Modules:

Execution Time (microseconds): 8.8 + 20.6n

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	DP	R2→0th element	-
Vector(n)	DP	R3→0th element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	DP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,F0,F1.

Algorithm:

Uses indexing in load, add, store sequence controlled by BCTB on length.

Loading of an element is done with two LE instructions instead of one LED due to addressing inadequacies of R3 which is the input pointer.

VV2D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV2D3 Size of Code Area 22 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV2D3

Function: Add two double precision 3-vectors.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
V1+V2, where V1 and V2 are double precision 3-vectors.
```

[ ] Other Library Modules:

Execution Time (microseconds): 51.4

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	DP	R2→0^{th} element	-
Vector(3)	DP	R3→0^{th} element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	DP	R1→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,F0,F1,F2,F3,F4,F5.

Algorithm:

Loads F0,F2,F4 with first half of each element of V2. Loads F1,F3,F5 with second half of each element of V2. } Due to addressing peculiarities of R3.

Adds double from V1 to F0, F2, F4;

Stores double into elements of result.

VV2SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV2SN Size of Code Area 10 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV2SN

Function: Add two single precision vectors of length n. Also used to add two matrices.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
V1+V2, where V1 and V2 are single precision vectors of
      length ≠ 3.
```

[ ] Other Library Modules:

Execution Time (microseconds): 8.4 + 13.6n

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	SP	R2→0th element	-
Vector(n)	SP	R3→0th element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	SP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,F0,F1.

Algorithm:

Uses indexing in load, add, store sequence controlled by BCTB on length.

VV2S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV2S3 Size of Code Area 12 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV2S3

Function: Adds two single precision 3-vectors.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
V1+V2, where V1 and V2 are single precision 3-vectors.
```

[ ] Other Library Modules:

Execution Time (microseconds): 29.6

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	SP	R2→0^{th} element	-
Vector(3)	SP	R3→0^{th} element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	SP	R1→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,F0,F2,F4.

Algorithm:

Loads elements of V1 into F0,F2,F4.
Adds element of V2 respectively.
Stores F0,F2,F4 into elements of result.

VV3DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV3DN Size of Code Area 16 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV3DN

Function: Subtracts one double precision length n-vector from another. Also used to subtract matrices.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
V1-V2, where V1 and V2 are double precision vectors of
       length ≠ 3.
```

[ ] Other Library Modules:

Execution Time (microseconds): 6.0 + 22.7n

Input Arguments:

Type	Precision	How Passed	Units
Vector(n) V1	DP	R2→0th element	-
Vector(n)V2	DP	R3→0th element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	DP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,F0,F1.

Algorithm:

Exchange contents of R2/R3 for addressing considerations. Uses indexed load, subtract, store sequence controlled by BCTB on length. Load of minuend elements is done with two LE instructions due to use of R3 as index.

VV3D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV3D3 Size of Code Area 24 Hw
Stack Requirement: 0 Hw Data CSECT Size: 0 Hw
[X] Intrinsic [ ] Procedure
Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV3D3

Function: Subtracts two double precision vectors of length 3.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`V1-V2, where V1 and V2 are double precision 3-vectors.`

[ ] Other Library Modules:

Execution Time (microseconds): 55.4

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)V1	DP	R2→0th element	-
Vector(3)V2	DP	R3→0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	DP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,F0,F1.

Algorithm:

Exchange contents of R2 and R3 for addressing considerations.
Load minuend elements into F0/F1, F2/F3, F4/F5 using two LE instructions each because of R3 addressing rules.
Subtract subtrahend elements.
Store results using STED into result location.

VV3SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV3SN Size of Code Area 10 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV3SN

Function: Subtracts one length n single precision vector from another. Also used to subtract matrices.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

V1-V2, where V1 and V2 are single precision vectors of length ≠ 3.

[ ] Other Library Modules:

Execution Time (microseconds): 8.4 + 13.6n

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)V1	SP	R2→0^{th} element	-
Vector(n)V2	SP	R3→0^{th} element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	SP	R1→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,F0,F1.

Algorithm:

Uses indexed load, subtract, store sequence controlled by a BCTB loop on 'length'.

VV3S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV3S3 Size of Code Area 12 Hw
Stack Requirement: 0 Hw Data CSECT Size: 0 Hw
[X] Intrinsic [ ] Procedure
Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV3S3
Function: Subtracts two single precision vectors of length 3.

Invoked By:
[X] Compiler emitted code for HAL/S construct of the form:

```
V1 - V2, where V1 and V2 are single precision
        3-vectors.
```

[ ] Other Library Modules:

Execution Time (microseconds): 29.6

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)V1	SP	R2→0th element	-
Vector(3)V2	SP	R3→0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	SP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,F0,F2,F4.

Algorithm:

Load minuend elements into F0,F2,F4.
Subtract subtrahend elements from F0,F2,F4 respectively.
Store F0,F2,F4 into result elements.

VV4DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV4DN Size of Code Area 8 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV4DN

Function: Multiplies each element of a double precision length n vector by a double precision scalar. Also used to multiply matrix by scalar.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
V S, where V is a double precision vector of length ≠ 3
     and S is a double precision scalar.
```

[ ] Other Library Modules:

Execution Time (microseconds): 7.0 + 23.4n

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	DP	R2→0^{th} element	-
Scalar	DP	F0	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	DP	R1→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R4,R5,F0,F1,F2,F3.

Algorithm:

Uses BCTB loop to count down 'length', performing load, multiply, store for each element.

VV4D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV4DN Size of Code Area 18 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV4D3

Function: Multiplies each element of a double precision vector of length 3 by a double precision scalar.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
V S,  where V is length 3 double precision vector and S
      is a double precision scalar.
```

[ ] Other Library Modules:

Execution Time (microseconds): 68.4

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-
Vector(3)	DP	R2→0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	DP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R4,F0,F1,F2,F3.

Algorithm:

Simple load, multiply, store sequence for each element.

VV4SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV4SN Size of Code Area 8 Hw
Stack Requirement: 0 Hw Data CSECT Size: 0 Hw
[X] Intrinsic [ ] Procedure
Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV4SN
Function: Multiplies a length n single precision vector by a single precision scalar. Also used to multiply matrix by scalar.

Invoked By:
[X] Compiler emitted code for HAL/S construct of the form:

```
V S,  where V is a single precision vector of length ≠
      3 and S is a single precision scalar.
```

[ ] Other Library Modules:

Execution Time (microseconds): 7.0 + 14.0n

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-
Vector(n)	SP	R2→0th element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	SP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:
Registers Unsafe Across Call: R1,R2,R4,R5,F0,F1,F2,F3.

Algorithm:
Uses BCTB loop to count down 'length', performing load, multiply, store for each element.

VV4S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV4S3 Size of Code Area 12 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV4S3

Function: Multiplies each element of a single precision 3-vector by a single precision scalar.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
V S,  where V is a single precision 3-vector, and S is a
      single precision scalar.
```

[ ] Other Library Modules:

Execution Time (microseconds): 38.4

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-
Vector(3)	SP	R2→0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	SP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R4,F0,F2,F3.

Algorithm:

Simple load, multiply, store for each element.

VV5DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV5DN Size of Code Area 24 Hw
Stack Requirement: 0 Hw Data CSECT Size: 2 Hw
[X] Intrinsic [ ] Procedure
Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV5DN
Function: Divides a double precision vector of length n by a double precision scalar. Also used to divide matrix by scalar.

Invoked By:
[X] Compiler emitted code for HAL/S construct of the form:

```
V/S, where V is a double precision vector of length ≠ 3
     and S is a double precision scalar
```

[ ] Other Library Modules:

Execution Time (microseconds): 37.0 + 24.2n

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-
Vector(n)	DP	R2→0th element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	DP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
25	Scalar argument is zero	Store original vector as result

Comments:
Registers Unsafe Across Call: R1,R2,R4,R5,F0,F1,F2,F3.

Algorithm:
Test F0; if zero, preset quotient to 1; otherwise, compute 1/S and then use BCTB loop to count down 'length' performing load, multiply (by 1/S), store sequence for each element.

VV5D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV5D3 Size of Code Area 34 Hw

Stack Requirement: 0 Hw Data CSECT Size: 2 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV5D3

Function: Divide each element of a double precision length 3 vector by a double precision scalar.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
V/S,  where V is a double precision 3-vector and S is a
      double precision scalar.
```

[ ] Other Library Modules:

Execution Time (microseconds): 98.4

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-
Vector(3)	DP	R2→0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	DP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
25	Scalar argument is zero	Store original vector as result

Comments:

Registers Unsafe Across Call: R1,R2,R4,F0,F1,F2,F3,F4,F5,F6,F7.

Algorithm:

Test F0; if zero, send error and set quotient to 1;

Otherwise, quotient 1/arg is calculated and then used in a simple load, multiply, and store sequence for each element.

VV5SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV5SN Size of Code Area 14 Hw

Stack Requirement: 0 Hw Data CSECT Size: 2 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV5SN

Function: Divides single precision vector of length n by a single precision scalar. Also used to divide matrix by scalar.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
V/S,  where V is a single precision vector of length ≠ 3
      and S is a single precision scalar.
```

[ ] Other Library Modules:

Execution Time (microseconds): 7.2 + 18.0n

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-
Vector(n)	SP	R2→0th element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	SP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
25	Scalar argument is zero	Store original vector as result

Comments:

Registers Unsafe Across Call: R1,R2,R4,R5,F0,F1,F2,F3.

Algorithm:

Test F0, if zero, set F0 to 1; Uses BCTB loop to count down 'length' performing load, divide, store sequences for each element.

VV5S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV5S3 Size of Code Area 18 Hw

Stack Requirement: 0 Hw Data CSECT Size: 2 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV5S3

Function: Divide each element of a single precision vector of length 3 by a single precision scalar.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
V/S, where V is a single precision 3-vector and S is a
     single precision scalar.
```

[ ] Other Library Modules:

Execution Time (microseconds): 50.6

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-
Vector(3)	SP	R2→0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	SP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
25	Scalar argument is zero	Store original vector as result

Comments:

Registers Unsafe Across Call: R1,R2,R4,F0,F1,F2,F3.

Algorithm:

Test F0; if zero, set F0 to floating point 1; then do a simple load, divide, and store sequence for each element.

VV6DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV6DN Size of Code Area 12 Hw

Stack Requirement: 0 Hw Data CSECT Size: 2 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV6DN

Function: Forms dot product of two double precision length n vectors.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
V1.V2 where V1 and V2 are double precision vectors of
     length n, n ≠ 3.
```

[ ] Other Library Modules

Execution Time (microseconds): 16.4 + 25.4n

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	DP	R2→0^{th} element	-
Vector(n)	DP	R3→0^{th} element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,F0,F1,F2,F3.

Algorithm:

Loads R3 into R1 for addressability advantages.

Performs:

$$\sum_{i=1}^{n} V1_i \, V2_i$$ by loops counting down n;

Each pass loads $V1_i$, multiplies by $V2_i$, and adds to accumulated sum in F0.

VV6D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV6D3 Size of Code Area 16 Hw

Stack Requirement: 0 Hw Data CSECT Size: 2 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV6D3

Function: Forms dot product of two double precision 3-vectors.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
V1.V2  where V1 and V2 are double precision 3-vectors.
```

[ ] Other Library Modules

Execution Time (microseconds): 71.8

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	DP	R2→0th element	-
Vector(3)	DP	R3→0th element	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R2,R3,R4,F0,F1,F2,F3.

Algorithm:

Moves R3 to R1 for addressability advantages.

Performs:

$$\sum_{i=1}^{n} V1_i\, V2_i$$ via straight line code, no loops, accumulating result in F0.

VV6SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV6SN Size of Code Area 12 Hw

Stack Requirement: 0 Hw Data CSECT Size: 2 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV6SN

Function: Forms dot product of two length n single precision vectors.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
V1.V2  where V1 and V2 are single precision n-vectors,
       n ≠ 3.
```

[ ] Other Library Modules

Execution Time (microseconds): 15.2 + 16.8n

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	SP	R2→0th element	-
Vector(n)	SP	R3→0th element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,F5,F0,F1,F2,F3.

Algorithm:

Moves R3 to R1 for addressability advantages.

Performs:

$$\sum_{i=1}^{n} V1_i \, V2_i$$

by a loop counting down n; Each pass loads $V1_i$, multiplies by $V2_i$, and adds to accumulated sum in F0.

VV6S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV6S3 Size of Code Area 10 Hw
Stack Requirement: 0 Hw Data CSECT Size: 0 Hw
[X] Intrinsic [ ] Procedure
Other Library Modules Referenced: NONE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV6S3

Function: Forms dot product of two single precision 3-vectors.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
V1.V2 where V1 and V2 are single precision
      3-vectors.
```

[ ] Other Library Modules

Execution Time (microseconds): 41.8

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	SP	R2→0^{th} element	-
Vector(3)	SP	R3→0^{th} element	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R2,R3,R4,F0,F1,F2,F3.

Algorithm:

Calculates $V1_1V2_1 + V1_2V2_2 + V1_3V2_3$ via direct code, no loops, accumulating result in F0.

VV7DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV7DN Size of Code Area 10 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV7DN

Function: Vector negate, double precision, length n. Also used to negate matrices.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
-V, where V is a double precision vector of length n,
    n ≠ 3.
```

[ ] Other Library Modules

Execution Time (microseconds): 7.0 + 11.4n

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	DP	R2→0th element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	DP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R4,R5,F0,F1.

Algorithm:

Uses loop to count down 'n', each pass performing load, negate, store sequence on current vector element.

VV7D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV7D3 Size of Code Area 20 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV7D3

Function: Vector negate, double precision for vectors of length 3.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
-V,  where V is a double precision 3-vector.
```

[ ] Other Library Modules

Execution Time (microseconds): 32.4

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	DP	R2→0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	DP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R4,F0,F1,F2,F3,F4,F5.

Algorithm:

Simple, direct code sequence, no loops. Performs 3 loads, 3 negations, 3 stores.

VV7SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV7SN Size of Code Area 10 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV7SN

Function: Vector negate, single precision length n. Also used to negate matrices.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
-V,  where V is a single precision vector of length n,
     n ≠ 3.
```

[ ] Other Library Modules

Execution Time (microseconds): 7.0 + 9.0n

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	SP	R2→0^{th} element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	SP	R1→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R4,R5,F0,F1.

Algorithm:

Uses loop to count down 'n', each pass performing load, negate, store sequence on current vector element.

VV7S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV7S3 Size of Code Area 14 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV7S3

Function: Vector negate, single precision for vectors of length 3.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
-V,  where V is a single precision 3-vector.
```

[ ] Other Library Modules

Execution Time (microseconds): 23.4

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	SP	R2→ 0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	SP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R4,F0,F1,F2,F3,F4,F5.

Algorithm:

Direct, inline code, no loops. Does 3 loads, 3 negations, 3 stores.

VV8D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV8D3 Size of Code Area 12 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV8D3

Function: Compares two double precision 3-vectors.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
IF X = Y…, where X and Y are single precision
           3-vectors.
```

[ ] Other Library Modules:

Execution Time (microseconds):

59.0 if X=Y;
16.2n + 24.6 if X≠Y
where n = 3 -(index of last non-matching pair of elements).

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	DP	R2→0^{th} element	-
Vector(3)	DP	R3→0^{th} element	-

Output Results:

Type	Precision	How Passed	Units
Equal/not equal	-	Condition code	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,F0,F1.

Algorithm:

Loads a literal 3 into R5, then drops into VV8DN code.

VV8D3

Secondary Entry Name: VV8DN

Function: Compares two double precision vectors of length n. Also used to compare matrices.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
IF X = Y..., where X and Y are double precision vectors of
             length n, n ≠ 3.
```

Execution Time (microseconds):

16.2n + 18.0 if X=Y;
16.2m + 22.2 if X≠Y,
where m = n-(index of last non-matching pair of elements)

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	DP	R2→0^{th} element	-
Vector(n)	DP	R3→0^{th} element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Equal/not equal	-	Condition code	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,F0,F1.

Algorithm:

Loads R3 into R1 for better addressability. Loops, counting down 'size', each pass compares values of one element of each vector. When first non-compare occurs, branch to return point is taken, exiting loop.

Condition code is set based upon whether count down loop reaches 0. Condition code of 00 indicates equality, 01 indicates inequality.

VV8S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV8S3 Size of Code Area 12 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV8S3

Function: Compares two single precision vectors of length 3.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
IF X = Y…, where X and Y are single precision
           3-vectors.
```

[ ] Other Library Modules:

Execution Time (microseconds):

42.8 if X=Y;

10.8n + 8.4 if X≠Y,

where n = 4 - (index of last non-matching pair of elements).

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	SP	R2→0th element	-
Vector(3)	SP	R3→0th element	-

Output Results:

Type	Precision	How Passed	Units
Equal/not equal	-	Condition code	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,F0,F1.

Algorithm:

Loads a literal 3 into R5, then continues with the VV8SN algorithm.

VV8S3

Secondary Entry Name: VV8SN

Function: Compares two single precision vectors of length n. Also used to compare matrices.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
IF X = Y…, where X and Y are single precision vectors of
           length n, n ≠ 3.
```

[ ] Other Library Modules:

Execution Time (microseconds):

10.8n + 8.0 if X=Y;
10.8m + 6.0 if X≠Y,
where m = n - (index of last non-matching pair of elements) + 1.

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	SP	R2→0th element	-
Vector(n)	SP	R3→0th element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Equal/not equal	-	Condition code	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,F0,F1.

Algorithm:

Loads R3 into R1 for better addressability. Loops, counting down 'size', each pass compares values of one element of each vector. When first non-compare occurs, branch to return point is taken, exiting loop. Condition code is set based upon whether count down loop reaches 0. Condition code of 00 indicates equality, 01 indicates inequality.

VV9S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV9S3 Size of Code Area 14 Hw

Stack Requirement: 18 Hw Data CSECT Size: 0 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: SQRT

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV9S3

Function: Calculates magnitude of length 3 single precision vector.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
ABVAL(V),  where V is a single precision 3-vector.
```

[ ] Other Library Modules

Execution Time (microseconds): 168.3

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	SP	R2→0th element	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3.

This routine will generate incorrect results if the input vector element is greater than SQRT(7.2370055773322600e + 75).

Algorithm:

Loads and multiplies each element of the input vector and adds to an accumulated value in F0.

VV10D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV10D3 Size of Code Area 70 Hw

Stack Requirement: 28 Hw Data CSECT Size: 2 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: DSQRT,VV0DN

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV10D3

Function: Creates unit vector of length 3 for input 3-vector in double precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`UNIT(V), where V is a double precision 3-vector.`

[ ] Other Library Modules

Execution Time (microseconds): 402.7

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	DP	R4→0^{th} element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	DP	R2→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
28	Input vector has all elements=0	Return input vector

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

This routine will generate incorrect results if the input vector element is greater then SQRT(7.2370055773322600e + 75).

Algorithm:

Loads R5 with literal 3, then continues with the VV10DN algorithm.

VV10D3

Secondary Entry Name: VV9D3

Function: Calculates magnitude of length 3 double precision vector.

Invoked By:

- [X] Compiler emitted code for HAL/S construct of the form:

```
ABVAL(V), where V is a double precision 3-vector.
```

- [ ] Other Library Modules

Execution Time (microseconds): 300.2

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	DP	R2→0^{th} element	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

This routine will generate incorrect results if the input vector element is greater than SQRT(7.2370055773322600e + 75).

Algorithm:

Loads R5 with literal 3, then continues with the VV9DN algorithm.

VV10D3

Secondary Entry Name: VV9DN

Function: Calculates magnitude of length n double precision vector.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
ABVAL(V), where V is a double precision vector of length n,
          n ≠ 3.
```

Execution Time (microseconds): 226.6 + 24.4n

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	DP	R2→0^{th} element	-
Integer(n)	SP	R5	

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

This routine will generate incorrect results if the input vector element is greater than SQRT(7.2370055773322600e + 75).

Algorithm:

Uses loop counting down size (n), each pass squaring an element of input vector and adding to accumulated value in F0; after loop, calls DSQRT to obtain final result in F0.

VV10D3

Secondary Entry Name: VV10DN

Function: Creates unit vector of length n for input vector of length n in double precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
UNIT(V), where V is a double precision vector of length n,
         n ≠ 3.
```

[ ] Other Library Modules

Execution Time (microseconds): 259.7 + 47.8n

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	DP	R4→0^{th} element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	DP	R2→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
28	Every element of input vector is 0	Return input vector

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

This routine will generate incorrect results if the input vector element is greater than SQRT(7.2370055773322600e + 75).

Algorithm:

Uses loop to sum squares of elements of input vector. Calls DSQRT to get square root of sum. Uses loop to divide each element of input vector by square root value and store into result vector.

VV10S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VV10S3 Size of Code Area 46 Hw

Stack Requirement: 24 Hw Data CSECT Size: 2 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: SQRT,VV0SN

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VV10S3

Function: Creates unit vector of length 3 for input 3-vector in single precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
UNIT(V), where V is a single precision 3-vector.
```

[ ] Other Library Modules

Execution Time (microseconds): 236.4

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	SP	R4→0^{th} element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	SP	R2→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
28	Input vector has all elements=0	Return input vector

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3.

This routine will generate incorrect results if the input vector element is greater than SQRT(7.2370055773322600e + 75).

Algorithm:

Loads R5 with literal 3, then continues with VV10SN algorithm.

VV10S3

Secondary Entry Name: VV9SN

Function: Calculates magnitude of single precision vector of length n.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`ABVAL(V)`, where V is a single precision vector of length n, n ≠ 3.

[ ] Other Library Modules

Execution Time (microseconds): 118.9 + 14.0n

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	SP	R2→0th element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3.

This routine will generate incorrect results if the input vector element is greater than SQRT(7.2370055773322600e + 75).

Algorithm:

Uses loop counting down size (n), each pass squaring an element of input vector and adding to accumulated value in F0. After loop, calls SQRT to obtain final result in F0.

VV10S3

Secondary Entry Name: VV10SN

Function: Creates unit vector of length n for input vector of length n in single precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
UNIT(V), where V is a single precision vector of length n,
         n ≠ 3.
```

Execution Time (microseconds): 130.6 + 32.8n

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	SP	R4→0th element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	SP	R2→0th element	-

Errors Detected:

Error #	Cause	Fixup
28	Sum of squares of all elements=0	Return zero vector

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3.

This routine will generate incorrect results if the input vector element is greater than SQRT(7.2370055773322600e + 75).

Algorithm:

Uses loop to sum squares of elements of input vector. Calls SQRT to get square root of sum. Uses loop to divide each element of input vector by square root return value and store into result vector.

VX6D3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VX6D3 Size of Code Area 36 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VX6D3

Function: Forms cross product of 2 double precision 3-vectors.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
X*Y, where X and Y are double precision vectors of
     length 3.
```

[ ] Other Library Modules:

Execution Time (microseconds) 137.6

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	DP	R2→0^{th} element	-
Vector(3)	DP	R3→0^{th} element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	DP	R1→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,F0,F1,F2,F3,F4,F5.

Algorithm:

Direct code, no loops, to calculate cross product:

$(X_2Y_3 - X_3Y_2,\ X_3Y_1 - X_1Y_3,\ X_1Y_2 - X_2Y_1)$

VX6S3

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VX6S3 Size of Code Area 22 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VX6S3

Function: Performs vector cross product of two single precision length 3 vectors.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
X*Y, where X and Y are single precision vectors of
     length 3.
```

[ ] Other Library Modules:

Execution Time (microseconds): 78.0

Input Arguments:

Type	Precision	How Passed	Units
Vector(3)	SP	R2→0th element	-
Vector(3)	SP	R3→0th element	-

Output Results:

Type	Precision	How Passed	Units
Vector(3)	SP	R1→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,F0,F1,F2,F3.

Algorithm:

Direct code, no loops, to calculate.

$$(X_2Y_3 - X_3Y_2, X_3Y_1 - X_1Y_3, X_1Y_2 - X_2Y_1)$$

6.3.4 Character Routine Descriptions

This subsection presents those routines which manipulate character data. Routines which convert to and from character data are not included here. Such routines are found under Section 6.3.6. (Miscellaneous Routine Descriptions).

CASPV

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CASPV Size of Code Area 64 Hw

Stack Requirement: 0 Hw Data CSECT Size: 2 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: CASPV

Function: Assigns a partition of a character string to a temporary string in a virtual accumulator.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`…C$(I TO J)…, where C is a Character string.`

[X] Other Library Modules:

CPASP

Execution Time (microseconds):

if p = 0: 43.8

if p > 0:

52.0	+	3.8	(if I is even)
	+	9.4k	(if k is odd)
	+	13.1k	(if I is even)
where	p = minimum (J-I+1, 255)		
	k = ceiling (P/2)		

Input Arguments:

Type	Precision	How Passed	Units
Character	-	R2→descriptor	-
Integer(I)	SP	R5	-
Integer(J)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Character (temporary)	-	R1→descriptor	-

Errors Detected:

Error #	Cause	Fixup
17	Indices out-of-bounds for input string	Set out-of-bounds index to first or last character of string

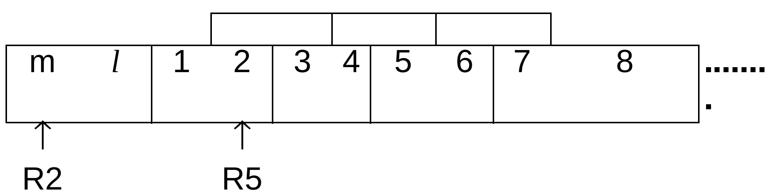
Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6.

Algorithm:

Several checks for possible errors are made before the transfer of characters is actually done. The index to the first character (I) is checked to be not less than 1. If it is, then set to 1. The index to the last character (J) is checked to be less than the length of the source string. If it is greater, then the index is set equal to the current length. Next, if J < I, the fixup is the NULL string. If the input actually is the NULL string, then no error is signaled. Finally, if the partition length exceeds the max length of the destination string, then the partition is truncated. All that remains to be done is the halfword-by-halfword transfer. The character count is incremented by one before dividing by two so that the halfword count is rounded to the next highest halfword if the character count was odd.

If I (the first character index) is odd then the transfer is straightforward. If even, then there are alignment problems to work around. The odd byte of the first halfword to move must not be moved, so halfwords crossing the "natural" halfword boundary are moved instead.



CASPV

Secondary Entry Name: CASP

Function: Assigns a partition of a character string to a receiver string.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`C1 = C2` $_{I\ TO\ J}$, `where C1 and C2 are character variables, and I and J are integers.`

[ ] Other Library Modules:

Execution Time (microseconds):

if p = 0: 41.0

if p > 0:

49.2	+	.8	(if p = maxlength(C1))
	+	3.8	(if I is even)
	+	9.4k	(if I is odd)
	+	13.1k	(if I is even)
where		p = minimum (J-I+1, maxlength(C1))	
		k = ceiling (P/2).	

Input Arguments:

Type	Precision	How Passed	Units
Character(C2)	-	R2→descriptor	-
Integer(I)	SP	R5	-
Integer(J)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Character(C1)	-	R1→descriptor	-

Errors Detected:

Error #	Cause	Fixup
17	Index out-of-bounds for input string	Set out-of-bounds index to first or last character of string

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6.

Algorithm:

Same as CASPV, except destination is a variable instead of a temporary.

CASV

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CASV Size of Code Area 28 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: CASV

Function: Character assign for output; assigns string from data to I/O buffer area.

Invoked By:

[ ] Compiler emitted code for HAL/S construct of the form:

[X] Other Library Modules:

COUTP, CINP

Execution Time (microseconds):

if C2 is null string: 29.2

if C2 $\neq$ null string: 40.2 + 9.4 (ceiling (P/2-1)) + .8
(if length(C2)>maxlength(C1)),
where p=minimum (length(C2), maxlength(C1)).

Input Arguments:

Type	Precision	How Passed	Units
Character	-	R2→descriptor	-

Output Results:

Type	Precision	How Passed	Units
Character	-	R1→descriptor	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5.

Algorithm:

First, the max length of the destination string is set to 255. Then, the length descriptor halfword of both the source string and the destination string are examined. The min of the max length of the destination and the current length of the source is taken as the new currlength of the destination. Next, the number of halfwords to move is found by incrementing the character count by one (in case the character count is odd) and dividing by two. If the source is a null string, the routine exits. If the character count is odd, the last byte in the string is moved anyway since it is always ignored. The assignment is made by moving the string halfword-by-halfword to the location specified by the destination pointer.

CASV

Secondary Entry Name: CAS

Function: Character assignment, non-partitioned.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`C1 = C2, where C1 and C2 are character strings.`

[X] Other Library Modules:

CIN

Execution Time (microseconds):

if input is null string: 32.0
if input ≠ null string: 43 + 9.4 * (ceiling(P/2-1)),
where p = length of input character string.

Input Arguments:

Type	Precision	How Passed	Units
Character(C2)	-	R2→descriptor	-

Output Results:

Type	Precision	How Passed	Units
Character(C1)	-	R1→descriptor	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5.

Algorithm:

Same as CASV, except MAXLEN of destination is not set to 255, but left with original MAXLEN value.

CATV

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CATV Size of Code Area 76 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: CATV

Function: Catenates two character strings and stores into a temporary.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
X||Y
```

[ ] Other Library Modules:

Execution Time (microseconds):

Times depend on whether first source string = destination string and whether the first source string has an odd character count creating an alignment problem.

if X is null string and Y is null: 52.2

if X and Y are not both null: XTIME + YTIME

XTIME: if X is null string: 24.0
if X ≠ null string: 29.8 + 9.4 * (ceiling (P/2))
where p = length(X).

YTIME: if Y is null string: 27.8
if Y is ≠ null string:
52.1 + 14.1 * (ceiling(Q/2-1)) ⎤ if p is odd
+ 6.0 (if P+Q is odd) ⎦
32.3 + 9.4 * (ceiling (Q/2)) if p is even
where Q= minimum (length(Y), 255-P).

Input Arguments:

Type	Precision	How Passed	Units
Character(X)	-	R2→descriptor	-
Character(Y)	-	R3→descriptor	-

Output Results:

Type	Precision	How Passed	Units
Character(temporary)	-	R1→descriptor	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0,F1.

Algorithm:

The lengths of the source strings are checked against the destination string for legal values. The second source string may be truncated if its length + that of the first source string exceed the length of the destination. If the first source string and the destination string are the same string (found by comparing addresses), then only the second source string is moved. After checking these things, the routine needs only to actually move the strings. The first is a straight halfword-by-halfword move. If its length is odd, then there is the alignment problem to contend with. The second string is moved starting where the first one left off.

See description of CASPV for what is done when the first source string has an odd character count.

CAT

Secondary Entry Name: CAT

Function: Catenates two character strings and stores into a third string.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

Not used yet.

[ ] Other Library Modules:

Execution Time (microseconds): N/A

Input Arguments:

Type	Precision	How Passed	Units
N/A	-	R2→descriptor	-
N/A	-	R3→descriptor	-

Output Results:

Type	Precision	How Passed	Units
N/A	-	R1→descriptor	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

N/A

Algorithm:

N/A

CINDEX

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CINDEX Size of Code Area 52 Hw

Stack Requirement: 18 Hw Data CSECT Size: 0 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: GTBYTE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: CINDEX

Function: Performs HAL/S INDEX function: finds occurrence of one character string within another.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`INDEX(A,B), where A and B are character strings; B is searched for within A.`

[ ] Other Library Modules:

Execution Time (microseconds):

if A is null: 32.8

if B is null: 38.0

if length(B) > length(A): 44.8

if result = 0:

$$\text{time} = 38.0 + \left[\sum_{I=1}^{JA} \sum_{J=1}^{KA_I} (15.4 + KB_J + KB_{J+I-1}) + 16.4 + KB_I\right]$$

where JA = 2 * (length(C1) - length(C2))+1

KA_I= # of compares required to determine that

C1$(length(C2) at I) $\neq$ C2

KB_X= 14.4 if X is even

15.6 if X is odd

if result $\neq$ 0:

$$\text{time} = 29.6 + \left[\sum_{I=1}^{\text{result}} \sum_{J=1}^{KA_I} (15.4 + KB_J + KB_{J+I-1}) + 16.4 + KB_I\right]$$

where KA_I= # of comparisons required to determine that
C1$(length(C2) at I) ≠ C2 if I ≠ result.
length(C2) if I = result.

KB_X is as above

Input Arguments:

Type	Precision	How Passed	Units
Character(A)	-	R2→descriptor	-
Character(B)	-	R4→descriptor	-

Output Results:

Type	Precision	How Passed	Units
Integer	SP	R5	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R5,F0,F1,F2,F3,F4,F5.

Algorithm:

1) If either string is null, return zero.
2) Set pointer to first character of A.
3) If size of B exceeds size of A, beyond A pointer, return zero.
4) Loop on size of B, comparing elements of A and B beginning at current A pointer; on non-equality go to step 6.
5) Comparison loop in 4 succeeded, return current A pointer.
6) Increment A pointer by one byte, go to step 3.

CLJSTV

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CLJSTV Size of Code Area 40 Hw

Stack Requirement: 18 Hw Data CSECT Size: 2 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: GTBYTE,STBYTE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: CLJSTV

Function: Left justifies a character string to a specified length by

1) padding on the right with blanks if too short;
2) truncating on the right if too long.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`LJUST(A,B)`, where A is a character string and B is an integer.

[ ] Other Library Modules:

Execution Time (microseconds):

34.0	+	2.8	(if B<255)
	+	2.0	(if n>0)
	+	40.8n	
	+	1.6	(if n is odd)
	+	0.4	(if m≤0)
	+	1.0	(if m is odd and n is even)
	-	1.0	(if m is odd and n is odd)
	+	23.8m	

where n = length(A)
m = maximum(B-n,0)

Input Arguments:

Type	Precision	How Passed	Units
Character(A)	-	R4→descriptor	-
Integer(B)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Character(temporary)	-	R2→descriptor	-

Errors Detected:

Error #	Cause	Fixup
18	Input string length greater than requested size or B<0	Truncate input string to specified size

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:

Compares requested length to 255 and retains smaller as L; compares L with input string length:

- if greater, truncates on right to length L and moves to output;
- if same, moves input string unchanged to output;
- if less, pads on right with blanks and moves to output.

CPAS

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CPAS Size of Code Area 80 Hw

Stack Requirement: 20 Hw Data CSECT Size: 2 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: GTBYTE,STBYTE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: CPAS

Function: Assigns a character string to a partition of another string.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$C2_{I\ TO\ J}$ = C1, where C1 and C2 are character strings

[X] Other Library Modules:

CPASP, CINP

Execution Time (microseconds):

$$34.2 + KA + \sum_{k=1}^{LHP} (5.6 + K_{CLOUT+K}) + KD + \sum_{k=1}^{NCHAR} (7.6 + KC_{I+K-1} + KF_k) + KE + \sum_{k=1}^{RHP} (5.6 + KC_{I+LIN+K-1}) + KG$$

where

LOUT = length(C2) before assignment

LIN = length(C1)

KA = 25.4 if J≤LOUT
34.0 if J>LOUT

LPART = J-I+1 (length of partition)

KB = 9.2 if LPART>0 and LPART≤LIN
13.8 if LPART>0 and LPART>LIN
0 otherwise

LHP = I-LOUT-1

KC_X = 19.2 if X is odd
17.2 if X is even

KD = 4.0 if LHP$\leq$0
0 otherwise

NCHAR = MINIMUM(LPART,LIN)
KE = .8 if NCHAR = 0   Note that in summations if start_index > end index
0 if otherwise   then summation goes to 0.

KF_X = 15.6 if X is odd
14.4 if X is even

RHP = LPART-LIN
KG = .4 if RHP$\leq$0
0 otherwise

Input Arguments:

Type	Precision	How Passed	Units
Character string(source)(C1)	-	R4→descriptor	-
Integer(I)	-	R5	-
Integer(J)	-	R6	-

Output Results:

Type	Precision	How Passed	Units
Character(destination)(C2)	-	R2→descriptor	-

Errors Detected:

Error #	Cause	Fixup
17	Index out-of-bounds for destination string or J<I-1	Set out-of-bounds to first or last character of destination

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:

First, the length of the partition is compared to the length of the source. If the source is longer, truncate it. If the destination partition is longer, then pad with blanks. The character count is determined and the string is moved byte-by-byte with the GTBYTE and STBYTE routines.

CPASP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CPASP Size of Code Area 18 Hw

Stack Requirement: 146 Hw Data CSECT Size: 0 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: CASPV,CPAS

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: CPASP

Function: Assigns a partition of a character string into a partition of another character string.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$C2_{K\ TO\ L} = C1_{I\ TO\ J}$; C1 and C2 are character strings and I,J,K,L are integers.

[ ] Other Library Modules:

Execution Time (microseconds): 42.8 + time for CASPV and CPAS.

Input Arguments:

Type	Precision	How Passed	Units
Character(source)(C1)	-	R4→descriptor	-
Integer(I)	SP	R5	-
Integer(J)	SP	R6	-
Integer(K||L)	(SP||SP)	R7	-

Output Results:

Type	Precision	How Passed	Units
Character(destination)(C2)	-	R2→descriptor	-

Errors Detected:

Error #	Cause	Fixup
17	Subscript of character string out of bounds	Set out-of-bounds value to first or last character of associated string

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:

The input partition is put into a VAC by the CASPV routine. The index arguments of the destination string and pointers are set up for the CPAS routine, that then moves the contents of the VAC into the destination string.

CPR

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CPR Size of Code Area 46 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None.

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: CPR

Function: Compares two character strings for '=' or '≠' and sets condition code.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
IF C1 = C2…, where C1 and C2 are character strings.
```

[X] Other Library Modules:

CPRA

Execution Time (microseconds):

If C1 and C2 do not halfword compare and K$\geq$2:

setup + 11.6J + 12.9

If K is even or K = 0 and C1 and C2 halfword compare 1 up till the K^{th} character:

setup + 11.6n + 20.1

If K is odd and C1 = C2 up till the K^{th} character:

setup + 11.6n + 29.9

If K is odd and only the last characters compared differ

setup + 11.6n + 20.3

where:

K	=	minimum(length(C1), length(C2))
setup	=	23 + 0.4 (if length(C2)<length(C1))
J	=	number of matching halfword compares
n	=	floor (K/2)

Input Arguments:

Type	Precision	How Passed	Units
Character(C1)	-	R2→descriptor	-
Character(C2)	-	R3→descriptor	-

Output Results:

Type	Precision	How Passed	Units
Equal/not equal	-	Condition code	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

In order to not change the condition code after the comparisons and before exiting, instructions that change the c.c. are replaced by those that do not change it. For example, LH is replaced by IHL and SLL.

Registers Unsafe Across Call: R2,R3,R4,R5,R6.

Algorithm:

See CPRC entry.

CPRC

Secondary Entry Name: CPRC

Function: Compares two character strings for collating sequence and sets condition code.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
IF C1 < C2..., or any other relational operator, except
               '=', or '≠', where C1 and C2 are character
               strings.
```

[ ] Other Library Modules:

Execution Time (microseconds): Same as CPR

Input Arguments:

Type	Precision	How Passed	Units
Character(C1)	-	R2→descriptor	-
Character(C2)	-	R3→descriptor	-

Output Results:

Type	Precision	How Passed	Units
Relation	-	Condition code	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

See CPR

Registers Unsafe Across Call: R2,R3,R4,R5,R6.

Algorithm:

Find the smaller of the lengths of the two strings to be compared. Compare this many characters halfword-by-halfword, and compare the upper bytes of the last halfwords separately if the character count is odd. If any of these comparisons are unequal, then return the resultant condition code. If all are equal, then compare the lengths of the two strings, and return the resultant code.

CPRA

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CPRA   Size of Code Area 22 Hw

Stack Requirement: 22 Hw   Data CSECT Size: 0 Hw

[ ] Intrinsic   [X] Procedure

Other Library Modules Referenced: CPR

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: CPRA

Function: Compares arrays of character strings when the arrays are located in structures for '=' or '≠' and sets the condition code.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
IF S1 = S2, where S1 and S2 are structures, one of whose
            nodes is a length n array of character
            strings.
```

[ ] Other Library Modules:

Execution Time (microseconds):

$$23.2 + \sum_{k=1}^{NCM_P} (18.2 + CPRTIME_k) - 14.2 \text{ (if arrays are not equal)}$$

where

NCMP = number of elements in arrays if arrays are equal, index of first nonmatching character strings in arrays if arrays not equal.

$CPRTIME_X$ = time in CPR for S1.C$(X:) and S2.C$(X:) where C is the node for the array of character strings.

Input Arguments:

Type	Precision	How Passed	Units
Character array	-	R2→0^{th} element	-
Character array	-	R3→0^{th} element	-
Integer(#Hw in ea. string)	SP	R6	-
Integer(# of array cells)	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
Equal/not equal	-	Condition code	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: None

Algorithm:

Pointers to character strings within the array are set, then CPR routine called. If all pairs of strings within the array are equal, result of CPRA is "equal", otherwise the result is "not equal".

CRJSTV

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CRJSTV   Size of Code Area 46 Hw

Stack Requirement: 18 Hw   Data CSECT Size: 2 Hw

[ ] Intrinsic   [X] Procedure

Other Library Modules Referenced: GTBYTE,STBYTE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: CRJSTV

Function: Right-adjusts a character string to a specified length by:

1) padding on left with blanks if too short;
2) truncating on left if too long.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`RJUST(A,B), where A is a character string, and B is an integer.`

[ ] Other Library Modules:

Execution Time (microseconds):

$$34.8 + KA + \sum_{k=1}^{NBLANK} (6.8 + KB_k) + KC$$

$$+ \sum_{k=1}^{NCHAR} (5.6 + KD_k + KB_{NBLANK+K}) + KE$$

where:

KA = 0 if B>255
2.8 if B<255

NBLANK = B-length(A) if B>length(A)
0 otherwise

KB_X = 19.2 if X is odd
17.2 if X is even

KC = 1.2 if NBLANK>0
0 otherwise

NCHAR = length(A)

KD_X = 15.6 if X is odd
14.4 if X is even

KE = .4 if NCHAR=0
0 otherwise

Input Arguments:

Type	Precision	How Passed	Units
Character(A)	-	R4→descriptor	-
Integer(B)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Character(temporary)	-	R2→descriptor	-

Errors Detected:

Error #	Cause	Fixup
18	Input string length greater than input size or B<0	Truncate input string on left to proper size

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:

Compares requested length to 255 and retains smaller as L; Compares current length with L:

if greater, truncates on left and moves to input;
if same, moves string to output;
if less, pads on left and moves input string to output.

CTRIMV

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CTRIMV Size of Code Area 94 Hw

Stack Requirement: 18 Hw Data CSECT Size: 0 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: GTBYTE,STBYTE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: CTRIMV

Function: Implements HAL/S TRIM function - strips leading and trailing blanks from a character string.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
TRIM(C), where C is a character string.
```

[ ] Other Library Modules:

Execution Time (microseconds):

If length(C) = 0: 30.4

If length(C) = 1 and C is a blank: 64.0

If length(C) = 1 and C is not a blank: 102.8

If length(C) > 1 and all blank

$$44.6 + KA + 13.2KB$$

where:

KA = 0 if length(C) is even
19.4 if length(C) is odd

KB = floor (length(C)/2)

If length(C) > 1 and not all blank

$$60.4 + KA + 13.2 * KB + KC + KD(11.6 * KE + 13.6 + KF)$$

$$+ \sum_{k=1}^{NCHAR} (39.2 + KG_{KH+k-1} + KI_k)$$

where:

KA	=	0 if length(C) is even
		19.4 if length(C) is odd and C$(#) is blank
		18.4 if length(C) is odd and C$(#)≠blank and C$(#)≠null
		22.4 if length(C) is odd and C$(#)≠blank and C$(#)=null
KB	=	# halfwords = blank||blank at the beginning of C
KC	=	0 if index of first non blank character is odd and this character = null
		4.8 if index of first non blank character is odd and this character = null
		9.0 if index of first non blank character is even.

KD	=	0 if length(C) is odd and (C$(#)≠blank, 1 otherwise.
KE	=	# halfwords = blank||blank at the end of C
KF	=	0 if index of last untrimmed character is even and this character ≠ null 3.6 if index of last untrimmed character is even and this character = null 4.4 if index of last untrimmed character is odd.
NCHAR	=	length of result.
KG_X	=	0 if X is even 2.0 if X is odd
KH	=	index of first non blank character
KI_X	=	0 if X is odd 1.2 if X is even

Input Arguments:

Type	Precision	How Passed	Units
Character(C)	-	R4→descriptor	-

Output Results:

Type	Precision	How Passed	Units
Character(temporary)	-	R2→descriptor	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:

Because there are no character or byte compare instructions on the AP-101, the routine first tests length of string. If odd, it sets R7 to 1. "Length" is shifted right 1, resulting in length in # of halfwords (-1 if odd). Compares first halfword with ~~bb~~, continues comparing consecutive halfwords of string with ~~bb~~, until a halfword that is not equal to ~~bb~~ is found. Then tests this halfword to see if first byte is ~~b~~. Adds length of string in halfwords to pointer to string, resulting in a pointer to end of string. Compares last halfword of string with ~~bb~~. If equal, then moves pointer back a halfword and again compares. When a halfword not equal to ~~bb~~ is found, the halfword is tested to see if it is C ~~b~~ or CC (where C stands for any character). Length of string is appropriately adjusted and routine branches to a character move loop.

6.3.5 Array Function Routine Descriptions

This subsection presents those routines which are classed as "ARRAY FUNCTIONS" by the *HAL/S Language Specification*. These are routines which operate upon arrayed arguments and produce a single element result.

DMAX

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DMAX Size of Code Area 10 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: DMAX

Function: Finds maximum value in a double precision scalar array.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`MAX(<DP array>)`

[ ] Other Library Modules:

Execution Time (microseconds):

17.6L + 14.6m + 11.4, where L = # of times CURRMAX changes;
M = # of times CURRMAX does not change, L+M = (# of elements in array) 1.

Input Arguments:

Type	Precision	How Passed	Units
Scalar array	DP	R2→0th element	-
Integer(size)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R2,R4,R5,F0,F1.

Algorithm:

A loop is set up to compare each element of the array to the current max. Initially, the first element is CURRMAX. Each subsequent element of greater value replaces the former CURRMAX. The counter is decremented after each comparison. The value of CURRMAX when the counter is zero is the max of the array and is passed back to the calling program.

DMIN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DMIN Size of Code Area 10 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: DMIN

Function: Finds minimum value in a double precision scalar array.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
MIN(<DP scalar array>)
```

[ ] Other Library Modules:

Execution Time (microseconds):

17.6L + 14.6m + 11.4, where L = # of times CURRMIN changes;
M = # of times CURRMIN does not change, L+M = (# of elements in array)-1.

Input Arguments:

Type	Precision	How Passed	Units
Scalar array	DP	R2→0th element	-
Integer(size)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R2,R4,R5,F0,F1.

Algorithm:

Similar to MAX functions, except register contains the current minimum and is changed when an element in the array has a smaller value than CURRMIN.

DPROD

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DPROD Size of Code Area 14 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: DPROD

Function: Calculates the product of the elements of a double precision scalar array.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
PROD(<DP scalar array>)
```

[ ] Other Library Modules:

Execution Time (microseconds):

20.6n + 6.2 if product is not zero, where n = # of elements in the array. 20.6m + 2.6 if product is zero, where m is the index into the linear representation of the array of the first zero element.

Input Arguments:

Type	Precision	How Passed	Units
Scalar array	DP	R2→0th element	-
Integer(size)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R2,R4,R5,F0,F1.

Algorithm:

Similar to the algorithm for the SUM functions. An accumulator is initialized to one. The value in the accumulator is multiplied by each element of the array; the result of each multiplication is saved in the accumulator. After each multiplication, the result is checked for a zero product. If the product is ever zero, the routine exits and returns to zero.

DSUM

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DSUM Size of Code Area 6 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: DSUM

Function: Calculates the sum of the elements of a double precision scalar array.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
SUM(<DP scalar array>)
```

[ ] Other Library Modules:

Execution Time (microseconds):

7.2+11.6n, where n=# of elements in the array.

Input Arguments:

Type	Precision	How Passed	Units
Scalar array	DP	R2→0th element	-
Integer(size)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R2,R4,R5,F0,F1.

Algorithm:

An accumulator (F0, F1) is initialized to zero. Each element of the array is added to the accumulator in a loop based upon the array size.

EMAX

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: EMAX Size of Code Area 8 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: EMAX

Function: Finds maximum value in a single precision scalar array.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
MAX(<SP scalar array>)
```

[ ] Other Library Modules:

Execution Time (microseconds):

9.8 + 10.8m + 12.2L, where m = # of times CURRMAX does not change; L = # of times CURRMAX changes; and M+L = (# of elements in array)-1.

Input Arguments:

Type	Precision	How Passed	Units
Scalar array	SP	R2→0^{th} element	-
Integer(size)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R2,R4,R5,F0.

Algorithm:

Same as DMAX except that operations are all single precision.

EMIN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: EMIN Size of Code Area 8 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: EMIN

Function: Finds minimum value in a single precision scalar array.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
MIN(<SP scalar array>)
```

[ ] Other Library Modules:

Execution Time (microseconds):

9.8 + 10.8m + 12.2L, where m = # of times CURRMIN does not change; L = # of times CURRMIN changes; and M+L = (# of elements in array)-1.

Input Arguments:

Type	Precision	How Passed	Units
Scalar array	SP	R2→0^{th} element	-
Integer(size)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R2,R4,R5,F0.

Algorithm:

Same as DMIN, except operations are in single precision floating point.

EPROD

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: EPROD   Size of Code Area 10 Hw

Stack Requirement: 0 Hw   Data CSECT Size: 0 Hw

[X] Intrinsic   [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: EPROD

Function: Calculates product of elements of a single precision scalar array.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
PROD(<SP scalar array>)
```

[ ] Other Library Modules:

Execution Time (microseconds):

13.2n + 4.6 if product is not zero, where n = # of elements in the array. 13.2m + 1.4 if product is zero, where m = index into the linear representation of the array of the first zero element.

Input Arguments:

Type	Precision	How Passed	Units
Scalar array	SP	R2→0th element	-
Integer(size)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R2,R4,R5,F0,F1.

Algorithm:

Same as DPROD.

ESUM

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: ESUM Size of Code Area 6 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: ESUM

Function: Calculates sum of elements of a single precision scalar array.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
SUM(<SP scalar array>)
```

[ ] Other Library Modules:

Execution Time (microseconds):

5.2 + 6.6n, where n = # of elements in the array.

Input Arguments:

Type	Precision	How Passed	Units
Scalar array	SP	R2→0th element	-
Integer(size)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R2,R4,R5,F0,F1.

Algorithm:

Same as DSUM.

HMAX

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: HMAX Size of Code Area 8 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: HMAX

Function: Finds maximum value in a single precision integer array.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
MAX(<SP integer array>)
```

[ ] Other Library Modules:

Execution Time (microseconds):

11.0 + 7.8m + 9.2k, where m = # of times CURRMAX does not change; k = # of times CURRMAX changes; and m+k = (# of elements in array)-1.

Input Arguments:

Type	Precision	How Passed	Units
Integer array	SP	R2→0th element	-
Integer(size)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Integer	SP	R5	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R2,R4,R5,R6.

Algorithm:

Same as DMAX, except that all operations deal with halfword integers.

HMIN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: HMIN Size of Code Area 8 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: HMIN

Function: Finds minimum value in a single precision integer array.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
MIN(<SP integer array>)
```

[ ] Other Library Modules:

Execution Time (microseconds):

11.0 + 7.8m + 9.2k, where m = # of times CURRMIN does not change; k = # of times CURRMIN changes; and m+k = (# of elements in array)-1.

Input Arguments:

Type	Precision	How Passed	Units
Integer array	SP	$R2 \rightarrow 0^{th}$ element	-
Integer(size)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Integer	SP	R5	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R2,R4,R5,R6.

Algorithm:

Same as DMIN, except that operations are for halfword integers.

HPROD

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: HPROD Size of Code Area 11 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: HPROD

Function: Calculates product of elements of a single precision integer array.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
PROD(<SP integer array>)
```

[ ] Other Library Modules:

Execution Time (microseconds):

12.4n + 5.8 if product is not zero, where n = # of elements in array. 12.4m + 2.2 if product is zero, where m = index into the linear representation of the array of the first zero element.

Input Arguments:

Type	Precision	How Passed	Units
Integer array	SP	R2→0^{th} element	-
Integer(size)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Integer	SP	R5	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R2,R4,R5,R6.

Algorithm:

Same as DPROD.

HSUM

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: HSUM Size of Code Area 6 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: HSUM

Function: Calculates sum of elements of a single precision integer array.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
SUM(<SP integer array>)
```

[ ] Other Library Modules:

Execution Time (microseconds):

4.4 + 5.4n, where n = # of elements in the array.

Input Arguments:

Type	Precision	How Passed	Units
Integer array	SP	R2→0^{th} element	-
Integer(size)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Integer	SP	R5	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Warning:

The HSUM routine will return incorrect results if the sum of the elements in the single precision integer array is greater than 32767 or less than -32768. Fixed-point overflow occurs for this range, but will not be detected if the program status word is set to mask out the overflow.

Registers Unsafe Across Call: R2,R4,R5,R6.

Algorithm:

Same as DSUM.

IMAX

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: IMAX Size of Code Area 8 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: IMAX

Function: Finds maximum value in a double precision integer array.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
MAX(<DP integer array>)
```

[ ] Other Library Modules:

Execution Time (microseconds):

11.1 + 7.8m + 4.3k, where m = # of times CURRMAX does not change, k = # of times CURRMAX changes, and m+k = # of elements in array-1.

Input Arguments:

Type	Precision	How Passed	Units
Integer array	DP	R2→0^{th} element	-
Integer(size)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R2,R4,R5,R6.

Algorithm:

Same as DMAX, except that all operations are on fullword integers.

IMIN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: IMIN Size of Code Area 8 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: IMIN

Function: Finds minimum value in a double precision integer array.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
MIN(<DP integer array>)
```

[ ] Other Library Modules:

Execution Time (microseconds):

11.1 + 7.8m + 9.3k, where m = # of times CURRMIN does not change, k = # of times CURRMIN changes, and m+k = # of elements in array-1.

Input Arguments:

Type	Precision	How Passed	Units
Integer array	DP	R2→0th element	-
Integer(size)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R2,R4,R5,R6.

Algorithm:

Same as DMIN, except that all operations are done for fullword integers.

IPROD

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: IPROD Size of Code Area 22 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: IPROD

Function: Calculates product of elements in a double precision integer array.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
PROD(<DP integer array>)
```

[ ] Other Library Modules:

Execution Time (microseconds):

17.0L + 21.6m + 5.8 if product is not zero, where L = # of positive intermediate products; m = # of negative intermediate products; L+m = # of elements in array. 17.0L + 21.6m + 19.6 if product is not zero, where L and m are as above, L+m = (index into linear representation of the array of the first zero element)-1.

Input Arguments:

Type	Precision	How Passed	Units
Integer array	DP	R2→0^{th} element	-
Integer(size)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R2,R4,R5,R6,R7.

Algorithm:

Same basic algorithm as DPROD, however, special detection of an overflow condition is performed: For fullword integer multiplication, the overflow indicator is only set when -1 is multiplied by -1. The result after each multiplication is checked for an overflow by testing the first 32 bits of the 64 bit result for all zeros or ones. If the result does overflow 32 bits, then a fixed point overflow is forced by adding a very large number to the first register of the pair (the register with the overflowing bits).

ISUM

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: ISUM Size of Code Area 6 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: ISUM

Function: Calculates sum of elements in a double precision integer array.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
SUM(<DP integer array>)
```

[ ] Other Library Modules:

Execution Time (microseconds):

4.4 + 5.4n, where n = # of elements in array.

Input Arguments:

Type	Precision	How Passed	Units
Integer array	DP	R2→0^{th} element	-
Integer(size)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R2,R4,R5,R6.

Algorithm:

Same as DSUM.

6.3.6 Miscellaneous Routine Descriptions

This subsection presents those routines which do not fall easily into the previous five sections. These encompass conversion routines as well as "service" routines used by other library members.

BTOC

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: BTOC Size of Code Area 28 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: BTOC

Function: Conversion from bit data to character data.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`CHARACTER`$_{@BIN}$`(<bit string>).`

[ ] Other Library Modules:

Execution Time (microseconds): 161.0 (for 16-bit string)

Input Arguments:

Type	Precision	How Passed	Units
Bit	-	R5	-
Integer(length)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Character	-	R2→descriptor	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7.

Algorithm:

First, unwanted bits are shifted out of the string, using the length argument. Then, bits are shifted one by one out of the top of R5 into the bottom of R4, where they are shifted to bit positions 15 and 31 and converted to character format. The output string is stored halfword by halfword, with the length taken directly from the input length.

CSHAPQ

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CSHAPQ Size of Code Area 40 Hw

Stack Requirement: 18 Hw Data CSECT Size: 4 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: CTOH,CTOI,CTOE,CTOD

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: CSHAPQ

Function: Shapes arrayed character data to arrayed numeric data of an explicit type and precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$\texttt{INTEGER}_{N}\texttt{(CA)}$, $\texttt{INTEGER}_{@DOUBLE,N}\texttt{(CA)}$, $\texttt{SCALAR}_{N}\texttt{(CA)}$, $\texttt{SCALAR}_{@DOUBLE,N}\texttt{(CA)}$, `where CA is a character array of length n of CHARACTER(m).`

[ ] Other Library Modules:

Execution Time (microseconds):

For halfword INTEGER conversion:

$$25.2 + \sum_{k=1}^{n} (19.6 + CTOH_K)$$

where $CTOH_K$ = time in CTOH for the K^{th} conversion.

For fullword INTEGER conversion:

$$24.8 + \sum_{k=1}^{n} (20.2 + CTOI_K)$$

where $CTOI_K$ = time in CTOI for the K^{th} conversion.

For fullword SCALAR conversion:

$$25.2 + \sum_{k=1}^{n} (19.6 + CTOE_K)$$

where $CTOE_K$ = time in CTOE for the K^{th} conversion.

For double-word SCALAR conversion:

$$22.8 + \sum_{k=1}^{n} (22.8 + CTOD_K)$$

where $CTOD_K$ = time in CTOD for the K^{th} conversion.

Input Arguments:

Type	Precision	How Passed	Units
Character array(n)	-	R4→1st element	-
Integer(n)	SP	R5	-
Integer(type *)	SP	R6	-
Integer(m)	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
Array(n)	Type	R2	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

type*: 0→H conversion
1→I conversion
2→E conversion
3→D conversion

Algorithm:

The address of one of 4 internal loops is loaded from a table using the type code as an index and control is passed to that loop. The four internal loops are similar in action: a call is made to the appropriate character conversion routine (CTOH, CTOI, CTOE, CTOD) followed by the appropriate store (STH, ST, STE, STED) into the result array, followed by instructions to bump both the character and result array pointers, looping on n.

CSLD

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CSLD Size of Code Area 246 Hw

Stack Requirement: 22 Hw Data CSECT Size: 4 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: CSLD

Function: Loads bit pattern of a character string.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
B=SUBBIT(C) where B is a bit string and C is a character
            string.
```

[ ] Other Library Modules:

Execution Time (microseconds):

if length(C) = 0: 28.8
if length(C) > 0: 56.3 + 0.8 (if length(C) > 4)

Input Arguments:

Type	Precision	How Passed	Units
Char string	-	R2→descriptor	-

Output Results:

Type	Precision	How Passed	Units
Bit string	Length 32	R5	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

If input string is null, the 0 bit string is returned; no error.

Registers Unsafe Across Call: R5,F0,F1.

Algorithm:

The first character is set to 1 by clearing R5. The character width is set to the current length of the string. For the rest, see the description under entry CSLDP, after the character partition checking, at the point marked [A].

CSLD

Secondary Entry Name: CPSLD

Function: Loads specified bits of a character string.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
BIT_STRING = SUBBIT FIRST TO LAST (CHAR_STRING);
```

where the 'TO' subscript may be replaced by the 'AT' subscript under rules given by matrix types

[ ] Other Library Modules:

Execution Time (microseconds): 71.8

Input Arguments:

Type	Precision	How Passed	Units
Char string	-	R2→descriptor	-
Integer(first bit)	SP	R5	-
Integer(last bit)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Bit string	Length(32)	R5	-

Errors Detected:

Error #	Cause	Fixup
30	Subbit partition out of range	1) If first bit<1, set to 1 (keep constant partition width by adjusting last bit) 2) If first or last bit>last bit of character string, set equal to last bit of char string

Comments:

If input string is null, ERROR 30 is sent and the 0 bit string is returned.

Registers Unsafe Across Call: R5,F0,F1.

Algorithm:

The subbit partition is tested for validity before anything else is done. All ERROR 30s are sent during these tests. 4 halfwords containing the required partition are loaded into a register pair. Unwanted bits are shifted off the top (left shift count = first bit-1), and the bottom (right shift count = 64-width), leaving the required string in R5.

CSLD

Secondary Entry Name: CPSLDP

Function: Loads selected bits of a partitioned character string.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
BIT_STRING = SUBBIT_A TO B(CHAR_STRING_C TO D);
```

where either or both of the 'TO' subscripts may be replaced by 'AT' subscripts under rules given by matrix types

[ ] Other Library Modules:

Execution Time (microseconds): 98.6 + 9.2 (if C is even)

Input Arguments:

Type	Precision	How Passed	Units
Char string	-	R2→descriptor	-
Integer(C)	SP	R5	-
Integer(D)	SP	R6	-
Integer(A||B)	(SP||SP)	R7	-

Output Results:

Type	Precision	How Passed	Units
Bit string	Length(32)	R5	-

Errors Detected:

Error #	Cause	Fixup
17	Character subscript out of legal range	(See CSLDP)
30	Subbit partition out of legal range	(See CPSLD)

Comments:

Registers Unsafe Across Call: R5,F0,F1.

Algorithm:

The character partition is checked for validity, and the 0 bit string is returned if last character < first character. Then the subbit partition is checked, resetting the bit pointer to point 8 bits farther on if the character partition begins in the second character of a halfword. 4 halfwords containing the required partition are loaded into register pair R4-R5. Unwanted bits are shifted off the top (shift count = relative 1st bit-1) and the bottom (shift count = 64-bit width), leaving the desired string in R5.

CSLD

Secondary Entry Name: CPSST

Function: To store a bit string into specified bits of a character string.

Invoked By:

☐ Compiler emitted code for HAL/S construct of the form:

```
SUBBIT_{A TO B}(CHAR_STRING) = BIT_STRING;

      where the 'TO' subscript may be replaced with the
      'AT' subscript under rules given by matrix types
```

☐ Other Library Modules:

Execution Time (microseconds): 114.4

Input Arguments:

Type	Precision	How Passed	Units
Bit string	-	R4	-
Integer(first bit)	SP	R5	-
Integer(last bit)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Char string	-	R2→descriptor	-

Errors Detected:

Error #	Cause	Fixup
30	Subbit partition out of legal range	(See CPSLD)

Comments:

CPSST cannot change the current length of the input character string. In particular, a null character string input will result in a null string output.

Registers Unsafe Across Call: R5,F0,F1.

Algorithm:

Set character partition width to the current length of the character string. If it is 0, exit immediately after sending ERROR 30.

[B] Test subbit partition for validity, and send ERROR 30 if anything is bad. Find the first halfword containing the specified partition. The first bit relative to that halfword, and the bit partition width thus:

bit width = last bit- first bit + 1
first halfword = 1 + (first bit - 1)/16
relative first bit = first bit - 16(first halfword - 1)

Load 4 halfwords, beginning with the first halfword of the partition, into register pair R4-R5.

Prepare a mask with 0s in the specified bit positions and 1s elsewhere as the 1s complement of:

$$(2^{\text{bit width}}-1)(2^{64\text{-relative last bit}})$$

where relative last bit = relative first bit + bit width - 1.

Use this mask to mask out the old bits in R4-R5. Shift the input bit string left by (64-relative last bit) positions to align it with the specified bit positions. Then OR it into the contents of R4-R5. Store this back into the character string, and exit.

CSLD

Secondary Entry Name: CPSSTP

Function: To store a bit string into specified bits of a partitioned character string.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$SUBBIT_{A\ TO\ B}(CHAR\_STRING_{C\ TO\ D})$ = BIT_STRING;

where either or both of the 'TO' subscripts may be replaced by 'AT' subscripts under rules given by matrix types

[ ] Other Library Modules:

Execution Time (microseconds): 145.0 + 9.2(if C is even)

Input Arguments:

Type	Precision	How Passed	Units
Integer(C)	SP	R5	-
Integer(D)	SP	R6	-
Integer(A||B)	(SP||SP)	R7	-
Bit string	-	R4	-

Output Results:

Type	Precision	How Passed	Units
Character string	-	R2→descriptor	-

Errors Detected:

Error #	Cause	Fixup
17	Character subscript out of legal range	(See CSSTP)
30	Subbit partition out of legal range	(See CPSST)

Comments:

CPSSTP cannot change the current length of the input character string.

Registers Unsafe Across Call: R5,F0,F1.

Algorithm:

Check character partition for validity, give any error 17s necessary, and exit if last char < first char or currlen = 0. Reset character pointer to 1 halfword before the first halfword of the specified partition, bumping first and last bits by 8 if the first character is even (so lies in low-order 8 bits of the halfword) after checking validity of first bit, and sending error 30 if it is < 1. Then continue as [B] of CPSST.

CSLD

Secondary Entry Name: CSLDP

Function: Loads bit pattern of a partitioned character string.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
BIT_STRING = SUBBIT_A TO B(CHAR_STRING_WIDTH AT FIRST);
```

where the 'AT' subscript may be replaced by the 'TO' subscript under rules given by matrix types

[ ] Other Library Modules:

Execution Time (microseconds): 69.7 + 0.8 (if WIDTH > 4)
+ 4.0 (if FIRST is even)

Input Arguments:

Type	Precision	How Passed	Units
Character string	-	R2→descriptor	-
Integer(first char)	SP	R5	-
Integer(last char)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Bit string	Length(32)	R5	-

Errors Detected:

Error #	Cause	Fixup
17	Character subscript out of legal range	1. If < 1, set to 1 2. If > length of string, set to length(string) 3. If last char < first char, return 0 string

Comments:

0 bit string returned if last character specified < first character specified (with ERROR 17).

Registers Unsafe Across Call: R5,F0,F1.

Algorithm:

The character partition is checked for validity before anything else is done. All error 17s are sent during this phase. [A] The partition width is checked, and if it is $\leq$ 0, the zero string is returned in R5. If greater than 4, it is set to 4. The address of the halfword containing the first character of the partition is found by adding 1/2(1+first character) to the address of the first halfword of the string. This halfword and the next two halfwords are loaded into the low half of R4, and the high and low halves of R5, respectively. Unwanted bits are masked off the left and shifted off the right (shift count = 48-8*width), and the desired bit string is left in R5.

CSLD

Secondary Entry Name: CSST

Function: Stores a bit string into a character string.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
SUBBIT(C) = B, where B is a BIT string and C is a character
                string.
```

[ ] Other Library Modules:

Execution Time (microseconds):

if length(C) = 0: 26.6
if length(C) > 0: 135.8 + 1.0 (if length(C) > 4)

Input Arguments:

Type	Precision	How Passed	Units
Bit string	-	R4	-

Output Results:

Type	Precision	How Passed	Units
Char string	-	R2→descriptor	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

If the length of the input character string is 0, no error is given, and nothing is changed. CSST cannot change the length of the input string.

Registers Unsafe Across Call: R5,F0,F1.

Algorithm:

The first character is set to 1 by clearing R5, the character width is set to the current length of the string. Processing continues as at [A] in the description of the algorithm at entry CSSTP.

CSLD

Secondary Entry Name: CSSTP

Function: To store a bit string into a partitioned character string.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
SUBBIT_{C TO D}(CHAR_STRING_{A TO B}) = BIT_STRING;
```

where either or both of the 'TO' subscripts may be replaced by 'AT' subscripts under rules given by matrix types

[ ] Other Library Modules:

Execution Time (microseconds):

148 + KA + KB, where KA = 1.0 if B-A>4, 0 otherwise
KB = 9.2 if A is even, 0 otherwise

Input Arguments:

Type	Precision	How Passed	Units
Bit string	-	R4	-
Integer(first char)	SP	R5	-
Integer(last char)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Char string	-	R2→descriptor	-

Errors Detected:

Error #	Cause	Fixup
17	Character subscript out of legal range	(See CSLDP), except if last char < first char, then leave input string unchanged

Comments:

Registers Unsafe Across Call: R5,F0,F1.

CSSTP cannot change the current length of the character string: gives error if subscript is out of current legal range.

Algorithm:

The character partition is checked for validity, and ERROR 17 is sent if anything is bad.

[A] The character partition width is checked. If it is ≤ 0, then the input character string is returned unchanged. If > 4, then it is set to 4. The first bit and last bit are determined as:

First bit = 1 + 8*(first character-1)

Last bit = First bit + 8*character width - 1

The first bit, last bit, and character width of the string are then sent to [B] under entry CPSST.

CSTRUC

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CSTRUC Size of Code Area 12 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: CSTRUC

Function: Compares two structures and returns result (= or ≠) in condition code.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
If S1 = S2 THEN..., where S1 and S2 are structures.
```

[ ] Other Library Modules:

Execution Time (microseconds):

5.4 + 10.4n, n = # halfwords compared.

Input Arguments:

Type	Precision	How Passed	Units
Structure	-	R2→1st Hw	-
Structure	-	R3→1st Hw	-
Integer(count)	SP	R5	Hw

Output Results:

Type	Precision	How Passed	Units
Equal/not equal	-	Condition code	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

In order that the correct code is in the condition code on exit, it is reset immediately before branching back to the calling program (BCRE does not set the condition code). An exclusive OR of a register with itself sets condition code to '00'(=). An OR of a nonzero register (R4 is used because, as the return address register, it is always assumed to be nonzero) resets condition code to '11'(≠).

Registers Unsafe Across Call: R2,R3,R4,R5,R6.

Algorithm:

The two structures are compared halfword by halfword until a pair does not match, or all of the halfwords are compared and found to be equal. The condition code is set by the compare instruction.

CTOB

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CTOB Size of Code Area 32 Hw

Stack Requirement: 18 Hw Data CSECT Size: 2 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: GTBYTE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: CTOB

Function: Conversion from character string data to bit string.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$BIT_{@BIN}(C)$, where C is a character string.

[ ] Other Library Modules:

Execution Time (microseconds):

$$25.8 + \sum_{k=1}^{NCHAR} (27.8 + KA_k + KB + KC)$$

where:

NCHAR	=	length(C)
KA_X	=	1.2 if X is odd 0 if X is even
KB	=	6.0 if C$(K)='1' 0 otherwise.
KC	=	4.4 if C$(K)=blank 0 otherwise.

Input Arguments:

Type	Precision	How Passed	Units
Character	-	R2→descriptor	-

Output Results:

Type	Precision	How Passed	Units
Bit	Length=31 implicitly	R5	-

Errors Detected:

Error #	Cause	Fixup
29	Input string not in standard format	Return zero bit string

Comments:

If the input string includes more than 32 digits, then high-order bits will be lost. Null string input causes an error.

Registers Unsafe Across Call: R5,F0,F1.

Algorithm:

Characters are examined one by one. Blanks are ignored. When a '1' is encountered, a '1' bit is shifted into the low-order bit of the result register. When a '0' is encountered, a '0' bit is shifted into the low-order bit of the result register.

CTOE

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CTOE Size of Code Area 319 Hw

Stack Requirement: 38 Hw Data CSECT Size: 2 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: GTBYTE

ENTRY POINT DESCRIPTIONS

Primary Entry Name: CTOE

Function: Performs internal character to single precision scalar conversion.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
SCALAR(<character string>) or
SCALAR@SINGLE(<character string>)
```

[X] Other Library Modules

CSHAPQ

Execution Time (microseconds):

88.4 + 11.0 * (floor * (# leading blanks/2))
+ 12.0 (if # leading blanks odd)
+ 10.2 * (# trailing blanks)
+ 2.0 (if + sign)
+ 7.0 (if - sign)
+ 59.6 * (# significant digits where S_k < X'4E19999A')
+ 17.6 * (# significant digits)
+ 47.2 (if at least 1 significant digit)
+ 62.4 * (# significant digits where S_k > X'4E19999A')
+ 20.6 (if decimal point)
+ 9.6 (if no exponents of any kind)
+ 40.2 (if any exponents)
+ 9.6 * (# E type exponents)
+ 15.2 * (# H type exponents)
+ 18.2 * (# B type exponents)
+ 9.8 * (# exponents)
+ 37.8 * (# additional exponents)
+ 0.2 * (# exponents with `+' sign)
+ 7.8 * (# exponents with `-' sign)
+ 24.6 * (total number of exponent digits)
+ 22.8 (if any B or H exponents)

\+ 7.6 * (total B exponent mod 4)
\+ 14.0 (if p=0)

$$
\left.\begin{array}{l}
+ \ (17.8 + 27.8\ \mathrm{div}(|p|,23))\ \text{(if p positive)} \\
+ \ (18.8 + 28.8\ \mathrm{div}(p,23))\ \text{(if p positive)} \\
+ \ 23.2 * ((\#\ \text{significant zeroes in the binary} \\
\quad \text{representation of } |p| \bmod 23) - 1 \\
\quad \text{(if } |p| \bmod 23 \text{ is even)} \\
+ \ 36.2 * ((\#\ \text{significant ones in the binary} \\
\quad \text{representation of } |p| \bmod 23) - 1 \\
\quad \text{(if } |p| \bmod 23 \text{ is odd)} \\
+ \ 14.2\ \text{(if } |p| \bmod 23 \neq 0) \\
+ \ 28.0 \\
+ \ 1.6\ \text{(if } p < 0)
\end{array}\right\} \text{if } p \neq 0
$$

\+ 14.4 * ((# of even indexed characters after leading blanks) + 1)
\+ 15.6 * ((# of odd indexed characters after leading blanks) + 1)

where p = total of E type exponents - (# significant digits after decimal point).

Input Arguments

Type	Precision	How Passed	Units
Character	-	R2→descriptor	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
20	Input string not in standard format	Return 0

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

First, leading and trailing blanks are stripped from the input string, and an error is signaled if the string has length = 0 or consists entirely of blanks. Next, a scalar is constructed from the digits of the input string to the left of the exponent. The construction proceeds as follows:

First, we set $S_0 = 0$. Now, at the k^{th} step, $k \geq 1$, we let $S_k=10*S_{k-1}+d_k$, where d_k is the k^{th} digit in the string. All calculations are double scalar. When S_k becomes $\geq$ X'4E19999A', all further digits are insignificant and are scanned for validity but otherwise ignored.

This yields a scalar which may be incorrect by a power-of-10 multiple, but otherwise represents the decimal number of the left of the exponent. As for the power of 10, if a decimal point is encountered while scanning the input string, a count is kept of how many digits there are to the right of the decimal point in the input string. The negative of this count is stored in temporary location COUNTE for later use.

Next, the type of exponent (if any) is determined, and the value is calculated with a simple fixed-point calculation ($ab_{10} = 10a + b$) and added to COUNTH, COUNTB, or COUNTE accordingly as the type of exponent is hexadecimal, binary, or decimal. Continue this process as long as there are remaining exponents.

If the end of the string is reached with no invalid characters found, then the scalar is modified according to the exponents already computed. First, the power-of-2 exponents are combined as 4H + B, since

$$16^H * 2^B = 2^{4H + B}.$$

The high part of this (power of 16) is added to the exponent of the scalar, while the low 2 bits control a loop in which the scalar is doubled 0-3 times.

Next, the decimal exponent, which has been combined with the correction for the decimal point in the input, is used as a power of 10 in the standard way of taking integral powers. The scalar intermediate is multiplied or divided by this result according to the sign of the exponent, completing the conversion.

CTOE

Secondary Entry Name: CTOD

Function: Performs internal character to double precision scalar conversion.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`SCALAR`$_{\texttt{@DOUBLE}}$`(<character string>)`

[X] Other Library Modules

CSHAPQ

Execution Time (microseconds): Time is computed by CTOE formula - 1.8.

Input Arguments:

Type	Precision	How Passed	Units
Character	-	R2→descriptor	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0,F1	-

Errors Detected:

Error #	Cause	Fixup
20	Input string not in standard format	Return 0

Comments:

Registers Unsafe Across Call: F0,F1,F2,F3,F4,F5.

Algorithm:

Same routine as CTOE; all conversions result in a double precision value of which the portion in F1 is discarded when single precision is desired by the caller of this routine.

CTOI

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CTOI Size of Code Area 104 Hw

Stack Requirement: 20 Hw Data CSECT Size: 2 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: GTBYTE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: CTOI

Function: Converts a character string to a double precision integer.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
INTEGER@DOUBLE(<character string>) or
BIT@DEC(<character string>).
```

[X] Other Library Modules:

CSHAPQ

Execution Time (microseconds):

k + 11.0 * (floor *(# leading blanks/2))
+ 18.6 (if # leading blanks odd)
+ 9.4 (if '-' sign)
+ 10.6 (if first character is a number)
+ 15.6 * (# even index characters after leading blanks)
+ 14.4 * (# odd index characters after leading blanks)
+ 13.0 (if # trailing blanks > 0)
+ 8.4 * (# trailing blanks)
+ 28.2 * ((# non blank characters) - 1)

where k = 72.6

Input Arguments:

Type	Precision	How Passed	Units
Character	-	R2→descriptor	-

Output Results:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Errors Detected:

Error #	Cause	Fixup
22	Input string not in standard format	Return 0

Comments:

Registers Unsafe Across Call: R5,F0,F1.

Algorithm:

First, leading blanks are stripped from the input string. If a minus sign is encountered, a flag is set. The basic conversion proceeds as follows: Initialize i_0 to 0. At step k, $k \geq 1$, let $i_k = 10.1_{k-1} + d_k$, where d_k is the k^{th} digit in the input string. At the end, this fixed-point calculation gives a fullword integer. This sign is tacked on, and the result is shifted left 16 bits if a halfword answer is required.

CTOI

Secondary Entry Name: CTOH

Function: Converts a character string to a single precision integer.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`INTEGER@SINGLE(<character string>)`

[X] Other Library Modules:

CSHAPQ

Execution Time (microseconds): Same as for CTOI, except k = 74.4.

Input Arguments:

Type	Precision	How Passed	Units
Character	-	R2→descriptor	-

Output Results:

Type	Precision	How Passed	Units
Integer	SP	R5	-

Errors Detected:

Error #	Cause	Fixup
22	Input string not in standard format	Return 0

Comments:

Registers Unsafe Across Call: R5,F0,F1.

Algorithm:

See CTOI.

CTOI

Secondary Entry Name: CTOK

Function: Converts a character string to a 32-bit string for use with the @DEC of the BIT conversion function.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
BIT@DEC(<character string>).
```

[ ] Other Library Modules:

Execution Time (microseconds):

85.8 + 11.0 * (floor * (# leading blanks/2))
+ 18.6 (if # leading blanks odd)
+ 15.6 * (# even index characters after leading blanks)
+ 14.4 * (# odd index characters after leading blanks)
+ 13.0 (if # trailing blanks > 0)
+ 8.4 * (# trailing blanks)
+ 28.2 * (# non blank characters - 1)

Input Arguments:

Type	Precision	How Passed	Units
Character	-	R2→descriptor	-

Output Results:

Type	Precision	How Passed	Units
Bit string	32-bit	R5	-

Errors Detected:

Error #	Cause	Fixup
22	Input string not in standard format	Return 0

Comments:

Registers Unsafe Across Call: R5,F0,F1.

Algorithm:

See CTOI.

CTOX

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CTOX Size of Code Area 60 Hw

Stack Requirement: 18 Hw Data CSECT Size: 4 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: GTBYTE

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: CTOX

Function: Conversion from character string to bit string, hexadecimal radix.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$BIT_{@HEX}(C)$, where C is a character string.

[ ] Other Library Modules:

Execution Time (microseconds):

$$32.0+\sum_{k=1}^{NCHAR}(33.6 + KA_k + Kb_k)$$

where:

NCHAR	=	length(C)'
KA_X	=	0 if C\$(X) is alphabetic
		6.8 if C\$(X) is numeric
KB_X	=	1.2 if X is odd
		0 if X is even

Input Arguments:

Type	Precision	How Passed	Units
Character	-	R2→descriptor	-

Output Results:

Type	Precision	How Passed	Units
Bit	32 bits	R5	-

Errors Detected:

Error #	Cause	Fixup
32	String not in standard hexadecimal conversion format	Return 0

Comments:

Imbedded blanks, or leading or trailing blanks, are all considered invalid characters. An input string too long to be accommodated in 32 bits will cause high order bits to be lost.

Registers Unsafe Across Call: R5,F0,F1.

Algorithm:

Characters are fetched one by one. In the CTOX entry, characters 'A' - 'F' are converted to their bit equivalents, characters '0' - '9' are passed on to the common section, and an error is signaled if the input character lies between X'39' and X'41' (DEU characters '9' and 'A' respectively). In the CTOO entry, an error is sent if the character is greater than X'37' (DEU character '7'). Other characters are passed to the common section for translation.

In the common section, decimal digits 0-9 (0-7 for octal) are translated to their bit equivalents, and an error is sent if the character precedes '0' in the collating sequence. These bit equivalents, and the ones passed from the CTOX section, are shifted into the low-order 4 bits (3 for octal) of the result register.

CTOX

Secondary Entry Name: CTOO

Function: Conversion from character string to bit string, octal radix.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`BIT@OCT(C), where C is a character string.`

[ ] Other Library Modules:

Execution Time (microseconds):

$$33.4 + \sum_{k=1}^{NCHAR} (34.2 + KA_k),$$

where:

NCHAR = length(C),

KA_X = 1.2 if X is odd

0 otherwise.

Input Arguments:

Type	Precision	How Passed	Units
Character	-	R2→descriptor	-

Output Results:

Type	Precision	How Passed	Units
Bit	32-bits	R5	-

Errors Detected:

Error #	Cause	Fixup
31	String not in standard octal conversion format	Return 0

Comments:

Registers Unsafe Across Call: R5,F0,F1.

Algorithm:

See CTOX.

DSLD

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DSLD Size of Code Area 22 Hw

Stack Requirement: 18 Hw Data CSECT Size: 2 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: DSLD

Function: Loads specified bits of a double precision scalar.

Invoked By:

☒ Compiler emitted code for HAL/S construct of the form:

```
SUBBIT(SCALAR_VAR@DOUBLE);
SUBBITFIRST TO LAST(SCALAR_VAR@DOUBLE).
```

☐ Other Library Modules:

Execution Time (microseconds): 36.5

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	R2→Scalar	-
Integer(first bit)	SP	R5	-
Integer(last bit)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Bit string	Length 32	R5	-

Errors Detected:

Error #	Cause	Fixup
30	Subbit partition out of range	1) If first bit<1, set to 1 2) If last bit>64, set to 64

Comments:

Registers Unsafe Across Call: R5

Algorithm:

Get the double word operand in register pair R2-R3. If first bit - 1 < 0, then give ERROR 30 and set to 0. Use first bit 1 as left shift count to eliminate unwanted high order bits. Compute 64 - last bit + first bit - 1, and give ERROR 30 and set to 0 if it is < 0. Use this as right shift count to justify bit string in R3. Return contents of R3.

DSST

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: DSST Size of Code Area 54 Hw

Stack Requirement: 18 Hw Data CSECT Size: 2 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: DSST

Function: Stores a bit string into selected bits of a double precision scalar.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$\texttt{SUBBIT}_{\texttt{A AT B}}\texttt{(DOUBLE\_SCALAR) = BIT\_STRING}$

[ ] Other Library Modules:

Execution Time (microseconds): 64.6

Input Arguments:

Type	Precision	How Passed	Units
Integer(A)	SP	R5	-
Integer(B)	SP	R6	-
Bit string	-	R7	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	R2→Scalar	-

Errors Detected:

Error #	Cause	Fixup
30	Subbit partition illegal	1) If first bit < 1, set to 1 2) If last bit > 64, set to 64

Comments:

Registers Unsafe Across Call: None

Algorithm:

Check first bit. If < 1, send error 30 and set to 1. Save the last bit, and get the partition width as last bit - first bit + 1. Create a mask of width = partition width as $2^{width} - 1$. If last bit $\leq$ 64, shift left by 64 - last bit. If last bit > 64, send error 30 and set last bit to 64 by shifting right by last bit - 64. Then invert the (doubleword) mask. Mask out the selected bits of the operand in storage. Then, shift the input bit string to the right position (left 64 - last bit, or right last bit - 64), and OR to the operand in storage, completing the operation.

ETOC

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: ETOC Size of Code Area 320 Hw

Stack Requirement: 32 Hw Data CSECT Size: 64 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: ETOC

Function: Converts a single precision scalar to standard internal character-string format for a scalar.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
CHARACTER(SCALAR_VAR)
```

[ ] Other Library Modules:

Execution Time (microseconds): 336.9

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Character string	Length=14	R2→descriptor	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Maxlength of output string is ignored.

Registers Unsafe Across Call: F0,F1,F2.

Algorithm:

Clear F1 to convert to double precision. Determine the sign and get the absolute value of the input. If input = 0, output string is 'b0.0' padded with blanks to length 14 (length 23 for DTOC).

The next operation reduces the exponent of the scalar to X'41' keeping track of the change in exponent that this requires. Since $\log_{10}x = (\log_{10}16)(\log_{16}x)$, this is done by getting (exponent - 65)*$\log_{10}16$ and using this as an exponent of 10, dividing the scalar by the result. It is possible for this to be off by 1, so another pass is made before continuing. At this point, the number is between 1 and 16. If it is greater than or equal to 10, multiply by 1/10 and record the exponent as one greater.

This causes the first decimal digit of the number to be the first hexadecimal digit of the scalar, in bits 8-11 of F0. This is stored, together with a blank if the value is ≥ 0, or a '-' if the value < 0. The remaining mantissa is in fractional form in F0-F1. This hexadecimal fraction is converted to decimal digit-by-digit by successive multiplication by 10. One digit is generated and stored with the decimal point, then 6 digits are stored in the next three halfwords.

The sign of the exponent (as calculated above) is tested, and either 'E+' or 'E-' is stored in the next halfword. Two decimal digits of exponent are stored in the last halfword.

<u>ETOC</u>

Secondary Entry Name: <u>DTOC</u>

Function: Converts a double precision scalar to standard internal character-string format for scalar.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
CHARACTER(DOUBLE_SCALAR)
```

[ ] Other Library Modules:

Execution Time (microseconds): 602.5

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0-F1	-

Output Results:

Type	Precision	How Passed	Units
Character string	Length=23	R2→descriptor	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,F1,F2.

Algorithm:

Similar to ETOC, except of course that F1 is not zeroed. Also, rather than 6 digits being stored in the loop, 14 are computed and stored. The exponent section also looks different, as one more digit is stored with the exponent, changing its alignment, thus storing successively '<digit>E', '± <digit>', '<digit><garbage>' in the last 3 halfwords.

ETOH

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: ETOH Size of Code Area 14 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic - Sector 0 [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: ETOH

Function: Converts single precision scalar value to single precision integer.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
I = S;  where I is a single precision integer, and S is a
        single precision scalar.
```

[X] Other Library Modules:

QSHAPQ

Execution Time (microseconds): 15.4

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Output Results:

Type	Precision	How Passed	Units
Integer	SP	R5	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Convert overflow (Error number 3:10) will occur if S is outside of the range, $-32768 \leq S \leq 32768$, (hex'C4800000' ≤ S ≤ hex'447FFFFF')

Warnings:

1) The ETOH routine returns incorrect results when the input argument is between 32767.5 and 32768. Fixed-point overflow occurs for this range, but will not be detected if the program status word is set to mask out the overflow.

2) Due to the use of CVFX instruction, the precision is lost during the conversion for the following input arguments (S) and inputs will be rounded incorrectly. I denotes the integer portion of S.
   a. 1≤ABS(I), and the fractional portion is slightly greater than 0.5.
      For example: S = hex '41x8000y'; where 1≤x≤F, 1≤y≤F.
   b. I = 0, and the fractional portion is slightly greater than 0.5.
      For example: S = hex '408000yy'; where 01≤yy≤FF.

Registers Unsafe Across Call: R5,F0.

Algorithm:

The six most significant hex digits of the scalar argument are converted to a fullword integer value. The 4 most significant hex digits of the integer value are left in the top halfword of the fixed point register after rounding the fraction. When the fractional portion of the result is less than or equal to 0.5 the result is rounded down, otherwise the result is rounded up.

ETOH

Secondary Entry Name: DTOH

Function: Converts a double precision scalar value to a single precision integer.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
I = D;  where I is a single precision integer, and D is a
        double precision scalar.
```

[ ] Other Library Modules:

Execution Time (microseconds): 17.4

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Output Results:

Type	Precision	How Passed	Units
Integer	SP	R5	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Warnings:

1) The DTOH routine returns incorrect results when the input argument (S) is in the range of 32767.50390625 ≤ S < 32768 (hex '447FFF8100000000' ≤ S < hex'447FFFFFFFFFFFFF'). Fix point overflow occurs for this range, but no error will be generated if the fixed point overflow interrupt is masked in the program status word.
2) Due to the use of the CVFX instruction, the precision is lost during the conversion for the following input arguments (S) and inputs will be rounded incorrectly. I denotes the integer portion of S.
   a. 1 ≤ ABS(I), and the fractional portion is slightly greater than 0.5. For example: S = hex '41x8000yzzzzzzzz'; where 1 ≤ x ≤ F, 1 ≤ y ≤ F, and z is any number.
   b. I = 0, and the fractional portion is slightly greater than 0.5. For example: S = hex '408000yyzzzzzzzz'; where 01 ≤ yy ≤ FF and z is any number.

Registers Unsafe Across Call: R5,F0.

Algorithm:

See ETOH.

GTBYTE

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: GTBYTE Size of Code Area 14 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: GTBYTE

Function: Fetches one character from a character string. Used for character manipulation by other library routines.

Invoked By:

[ ] Compiler emitted code for HAL/S construct of the form:

[X] Other Library Modules:

CPAS, CTOE, CLJSTV, CINDEX, CRJSTV, CTOB, CTOI, CTOX, CRTIMV

Execution Time (microseconds): 14.4 to obtain lower byte
15.6 to obtain upper byte

Input Arguments:

Type	Precision	How Passed	Units
Pointer	-	R2→1 Hw in front of Hw to fetch from	-
Flag (Which byte to fetch)	-	Lower half of R2: 00-upper byte, X'8000'-lower byte	-

Output Results:

Type	Precision	How Passed	Units
Single character	-	R5- upper halfword	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R2,R4,R5,F0.

Algorithm:

The halfword off of the pointer is loaded into a register for manipulation. If the flag indicates the upper byte is requested, the register is shifted right 8 bits and the lower half of the register is cleared to leave only the desired byte in the upper halfword of the register. If the flag indicates the lower byte is requested, then the first byte of the register is cleared. The flag is reset to indicate the upper byte if the lower byte was requested and vice versa, and the pointer is updated if the fetched byte was even. This is done now since GTBYTE is usually called a number of times from a loop.

ITOC

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: ITOC   Size of Code Area 104 Hw

Stack Requirement: 28 Hw   Data CSECT Size: 0 Hw

[ ] Intrinsic   [X] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: ITOC

Function: Converts a fullword integer into standard internal character string format for integers.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
CHARACTER(FULLWORD_INTEGER)
```

[ ] Other Library Modules:

Execution Time (microseconds): 254.6

Input Arguments:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Output Results:

Type	Precision	How Passed	Units
Character string	-	R2→descriptor	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

No leading zeros or leading or trailing blanks. Maxlength of output area ignored.

Registers Unsafe Across Call: None

Algorithm:

Digits are generated one by one. Thus: Let I = input integer. Then:

$d_k = I_k - 10(I_k/10)$ integer multiply and divide.

$I_{k+1} = (I_k - d_k)/10.$

The process terminates when $I_k = 0$. As pairs of digits are generated, they are stored, right to left, in a temporary output area. The temporary result is then given a sign if necessary and moved to the output area. If an odd number of characters were generated, the move is with 8 bits offset for left alignment.

ITOC

Secondary Entry Name: HTOC

Function: Converts a halfword integer into standard internal character-string format for integers.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`CHARACTER(HALFWORD_INTEGER)`

[ ] Other Library Modules:

Execution Time (microseconds): 189.6

Input Arguments:

Type	Precision	How Passed	Units
Integer	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Character string	-	R2→descriptor	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

No leading zeroes or leading or trailing blanks. Maxlength of output area ignored.

Registers Unsafe Across Call: None.

Algorithm:

Shift right algebraic 16 to convert single integer to double. Then proceed as in ITOC.

ITOD

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: ITOD   Size of Code Area 24 Hw

Stack Requirement: 0 Hw   Data CSECT Size: 0 Hw

[X] Intrinsic - Sector 0   [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: ITOD

Function: Converts a double precision integer to a double precision scalar.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
D = I; where D is a double precision scalar, and I is a
       double precision integer.
```

[X] Other Library Modules:

QSHAPQ

Execution Time (microseconds): 15.6

Input Arguments:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R4,R5,F0,F1.

Algorithm:

If the input argument is non-negative, then the register (F0) is loaded with HEX'4E800000'. F1 is then loaded with the value of the input argument. Next, the value HEX'4E800000 00000000' is subtracted from the (F0, F1) register pair and the result is returned to the calling program.

For negative values, the algorithm is the same, except the input argument is complemented before any other operations are performed. Also, the value used when manipulating the (F0, F1) register pair is HEX'CE800000 00000000' instead of HEX'4E800000 00000000'.

ITOE

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: ITOE   Size of Code Area: 6 Hw

Stack Requirement: 0 Hw   Data CSECT Size: 0 Hw

[X] Intrinsic - Sector 0   [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: ITOE

Function: Converts a double precision integer to a single precision scalar.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
S = I; where S is a single precision scalar, and I is
       a double precision integer
```

[X] Other Library Modules:

QSHAPQ

Execution Time (microseconds): 12.0

Input Arguments:

Type	Precision	How Passed	Units
Integer	DP	R5	-

Output Results:

Type	Precision	How Passed	Units
Scalar	SP	F0	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R4,R5,F0,F1.

Algorithm:

The integer argument is converted to floating point by the CVFL instruction and the binary point is adjusted by multiplication by a scale factor.

KTOC

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: KTOC Size of Code Area 70 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: KTOC

Function: Performs bit-string to character conversion with decimal radix.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$\texttt{CHARACTER}_{\texttt{@DEC}}\texttt{(BIT\_STRING)}$

[ ] Other Library Modules:

Execution Time (microseconds): 262.5 (for 16 bits)

Input Arguments:

Type	Precision	How Passed	Units
Bit string	-	R5	-
Integer(length of bit string)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Character string	-	R2→descriptor	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Maxlength of output area ignored. No leading or trailing blanks. "Sign bit" of input string ignored.

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0,F1.

Algorithm:

The length of the character string is computed as:

$$1 + (\log_{10}2)(\text{bit length}) \text{ truncated to an integer}$$

A halfword count is computed from this as:

halfword count = (1 + character count)/2

Decimal digits are generated one at a time, from right to left, thus: Let I_0=input string as unsigned integer,

$d_k = I_k - 10(I_k/10)$ integer multiply and divide

$I_{k+1} = (I_k - d_k)/10$

The process terminates when the halfword count is reached, with two digits stored per halfword. At the end, if an odd number of characters have been stored, the string must be shifted one character to the left for proper alignment.

QSHAPQ

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: QSHAPQ Size of Code Area 74 Hw

Stack Requirement: 18 Hw Data CSECT Size: 0 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: ETOH(DTOH), ROUND(DTOI), ITOE, ITOD

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: QSHAPQ

Function: Shapes data of a given type and precision to data of an explicit type and precision.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
Used by the INTEGER, SCALAR, MATRIX, and VECTOR shaping functions.
```

[ ] Other Library Modules:

Execution Time (microseconds): 42.6 + 31.8n, n = # times transferred.

Input Arguments:

Type	Precision	How Passed	Units
Integer/scalar	SP/DP	R2→first Hw	-
Integer(flag)	DP	R6: upper half for input data, lower half for output	-
Integer(count)	SP	R5: number of times to transfer	-

Output Results:

Type	Precision	How Passed	Units
Integer/scalar	SP/DP	R1→first Hw	-

Errors Detected:

Error #	Cause	Fixup
15	Scalar too large for integer conversion	Set to maximum representable value

Comments:

QSHAPQ is called only if more than one item of the same data type must be shaped. If only one item must undergo conversion, the conversion functions (DTOI, ETOI, ITOD, HTOE, etc.) are used.

Registers Unsafe Across Call: F0,F1.

Algorithm:

The flags for the input and output data are examined. The appropriate 'LOAD' routine is executed to load one item to be shaped. The appropriate 'STORE', or in some cases, 'CONVERT AND STORE' routine stores the shaped data item in the area pointed to by the destination pointer. The source pointer is updated after each load; the destination pointer is updated after each store. Data is shaped and transferred item-by-item.

The values of the flags (R6 upper and lower) are:

0 = HW integer
1 = FW integer
2 = FW scalar
3 = DW scalar

RANDOM

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: RANDOM   Size of Code Area 46 Hw

Stack Requirement: 18 Hw   Data CSECT Size: 2 Hw

[ ] Intrinsic   [X] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: RANDOM

Function: Generates random number with uniform distribution in range (0.0, 1.0).

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
...RANDOM...
```

[ ] Other Library Modules:

Execution Time (microseconds): 54.4

Input Arguments:

Type	Precision	How Passed	Units
None			

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0/F1	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

The original SEED(F'1435') is declared as a data constant. To allow storage into this "constant" for updating SEED, the storage protection is turned off for SEED.

Registers Unsafe Across Call: F0,F1,F2,F3.

Algorithm:

Multiply F'65539' by SEED. SEED originally=F'1435', but is updated on each pass through RANDOM. Use the least significant 32 bits of this product (SEED x 65539) to form the new SEED. If the result is $\geq$ 0, then RESULT = new SEED. If RESULT < 0, then new SEED = RESULT - NEGMAX, where NEGMAX = X'80000000'. The positive new SEED is saved for future use, and is also converted to a floating point number for present computation of a random number. Multiply the floating point value by 2^{-31} to produce a random number in the range (0.0, 1.0).

RANDOM

Secondary Entry Name: RANDG

Function: Generates random number from Gaussian distribution, mean zero, variance one.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
...RANDOMG...
```

[ ] Other Library Modules:

Execution Time (microseconds): 575.8

Input Arguments:

Type	Precision	How Passed	Units
None			

Output Results:

Type	Precision	How Passed	Units
Scalar	DP	F0/F1	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Same as RANDOM.

Registers Unsafe Across Call: F0,F1,F2,F3.

Algorithm:

RANDG uses the formula $Y = \sum_{i=1}^{12} X_i - 6.0$, where X_i is a random number generated by RANDOM.

STBYTE

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: STBYTE Size of Code Area 22 Hw

Stack Requirement: 0 Hw Data CSECT Size: 0 Hw

[X] Intrinsic [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: STBYTE

Function: Stores one character into a character string; Used for character manipulation by other library routines.

Invoked By:

[ ] Compiler emitted code for HAL/S construct of the form:

[X] Other Library Modules:

CLJSTV, CPAS, CRJSTV, CTRIMV

Execution Time (microseconds): 19.2 to store in upper byte.
17.2 to store in lower byte.

Input Arguments:

Type	Precision	How Passed	Units
Single character	-	R5	-
Flag (which byte to store into)	-	Lower Hw of R1: 00-upper byte, X'8000' to lower byte	-

Output Results:

Type	Precision	How Passed	Units
Pointer	-	R1→Hw to store into	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: R1,R4,R5,F0.

Algorithm:

The flag is tested for an even or odd byte to store into. If odd (upper), the flag is set to indicate even (lower) for the probable loop that STBYTE is in. Then, the byte is inserted into the upper byte of the appropriate halfword. If the flag indicates an even byte to store into, then the byte is inserted into the lower byte of the appropriate halfword. The flag is set to 0 to indicate that the next time around the loop, the byte will be odd. The pointer is updated to the next halfword.

XTOC

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: XTOC   Size of Code Area 68 Hw

Stack Requirement: 0 Hw   Data CSECT Size: 0 Hw

[X] Intrinsic   [ ] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: XTOC

Function: Converts bit string to a string of hexadecimal characters.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$\mathtt{CHARACTER}_{\mathtt{@HEX}}\mathtt{(BIT\_STRING)}$

[ ] Other Library Modules:

Execution Time (microseconds):

35.9 + 32.2 * (# of digits 0-9) + 33.9 * (# of letters A-F),
where 8 * ((# of digits)+(# of letters))= # bits.

Input Arguments:

Type	Precision	How Passed	Units
Bit string	-	R5	-
Integer(length of Bit string)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Character string	-	R2→descriptor	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Output string length depends on input string length. The maxlength of the output area is ignored.

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0.

Algorithm:

A character count is determined as the integer part of (bit length +3)/4. The bit string is positioned in register pair R4-R5 with the first hexadecimal digit in bits 12-15 of R4 thus:

1) Clear R4; string right-justified in R5 on input.
2) Compute greatest multiple of 4 in 52 - bit length.
3) Use result of 2) as a shift count to shift R4-R5 left double.

Compute a halfword count for use as a loop counter:

halfword count = (1 + character count)/2

The character count is stored in the descriptor halfword as the current length of the output string. Digits are generated by shifting left 4 and stored two at a time in the output string, after converting DEU format by adding X'30' to each digit. Exit when proper number of halfwords have been stored.

XTOC

Secondary Entry Name: OTOC

Function: Converts a bit string into a string of octal characters.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`CHARACTER`$_{@OCT}$`(BIT_STRING)`

[ ] Other Library Modules:

Execution Time (microseconds): 46.2 + 32.3*(# of digits),
where 6 *(# of digits) = # bits + 2.

Input Arguments:

Type	Precision	How Passed	Units
Bit string	-	R5	-
Integer(length of bit string)	SP	R6	-

Output Results:

Type	Precision	How Passed	Units
Character string	-	R2→descriptor	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Output string length depends on input string length. The maxlength of the output area is ignored.

Registers Unsafe Across Call: R1,R2,R3,R4,R5,R6,R7,F0.

Algorithm:

First, a character count is determined as the integer part of (bit length + 2)/3. The bit string is positioned in register pair R4-R5 with the first octal digit in bits 13-15 of R4 as follows:

1) Begin with R4 clear and the string right-aligned in R5.
2) Compute the shift count as 51-3(character count) and
3) shift R4-R5 left double by this amount.

Complete a halfword count for use as a loop counter as:

halfword count = (1 + character count)/2

The character count is stored in the descriptor halfword of the output string. Then, digits are generated in a loop, two at a time, by shifting R4-R5 left double 3 bits and adding X'30' to give the appropriate DEU character. As pairs of digits are assembled, they are stored into the output string, and exit is taken when the proper number of halfwords have been stored.

6.3.7 REMOTE Routine Descriptions

This subsection describes those routines which perform operations on REMOTE data. REMOTE data is data which may reside in a sector of AP-101 core which is neither sector 0 nor the current data sector indicated in the Program Status Word at the time the routine is called. In order to ensure addressability of such data, these routines are passed, instead of pointers directly to their arguments, pointers to complete address constants, or "ZCONs", containing both the address of the argument and the number of the sector in which it resides. These complete address constants, together with a special AP-101 addressing mode, allow access to any area of AP-101 core without changing bits in the Program Status Word.

REMOTE routines are invoked (rather than the normal versions of the same routines) when at least one of the arguments of the routine has the REMOTE attribute. Aggregate data types (VECTOR, MATRIX, STRUCTURE and CHARACTER types) and arrays of these data types currently have REMOTE routines. For arrays of single/double precision scalars or integers, no REMOTE routines exist and a severity 2 FT108 error message is emitted when an attempt is made to pass the REMOTE argument to a function.

CASRPV

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name CASRPV   Size of Code Area 86 Hw

Stack Requirement: 22 Hw   Data CSECT Size: 2 Hw

[ ] Intrinsic   [X] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: CASRPV

Function: Assigns a partition of a REMOTE character string into a temporary character string in a virtual accumulator.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

…$C2_{I\ TO\ J}$ … `where C2 is a REMOTE character variable and result is a temporary string.`

[ ] Other Library Modules:

Execution Time (microseconds):

If P = 0 and length(C2) = 0: 76.8

If P > 0 and I is odd: 89.0 + 15.8 * ceiling(n/2)

If P > 0 and I is even: 94.2 + 21.2 * ceiling(n/2)

where P = J - I + 1

n = minimum (p,255) Input Arguments:

Type	Precision	How Passed	Units
Integer(I)	SP	R5	-
Integer(J)	SP	R6	-
Character(C2)	-	R4→ZCON→descriptor	-

Output Results:

Type	Precision	How Passed	Units
Character(temporary)	-	R2→ZCON→descriptor	-

Errors Detected:

Error #	Cause	Fixup
17	Specified partition outside current string range	Set bad partition pointer(s) to the limits of the current string
	Last character indicator less than first character indicator and output string is not null	Return null string

Comments:

Registers Unsafe Across Call: None

Algorithm:

Set maxlength of result to 255. Test position of first character of partition. If < 1 then send error and set to 1.

Compare position of last character of partition. If it is > maxlength, reset to maxlength, and send error. Compare first and last positions. If last < first, then if input string is null do not send error. If input string is not null, send error and set result to null string. Make sure partition length does not exceed the maxlength of the destination string. If it does, truncate it. Increment character count before dividing by 2 to round resulting halfword count to next highest halfword. If position of first character of partition is odd, then transfer halfword by halfword. Otherwise, it is necessary to line characters up into right halves of halfwords by shifting.

CASRPV

Secondary Entry Name: CASRP

Function: REMOTE character assignment to declared data, partitioned input.

Invoked By:

☒ Compiler emitted code for HAL/S construct of the form:

`C1 = C2` $_{I\ TO\ J}$`;` where C1 and/or C2 are REMOTE character data.

☐ Other Library Modules:

Execution Time (microseconds):

If P = 0 and length(C2) = 0: 69.4

If P > 0 and I is odd: setup + 15.8 * (ceiling(n/2))

If P > 0 and I is even: setup + 5.2 + 21.2 * (ceiling(n/2))

where P = J- I + 1

setup = 81.6 if P≥max length(C1)

82.4 if P>max length(C1)

n = minimum (P, max length(C1))

Input Arguments:

Type	Precision	How Passed	Units
Integer(I)	SP	R5	-
Integer(J)	SP	R6	-
Character(C2)	-	R4→ZCON→descriptor	-

Output Results:

Type	Precision	How Passed	Units
Character(C1)	-	R2→ZCON→descriptor	-

Errors Detected:

Error #	Cause	Fixup
17	Same causes and fix-ups as CASRPV	-

Comments:

Registers Unsafe Across Call: None

Algorithm:

Same as CASRPV except maxlength of resultant string is used as passed and not set to 255.

CASRV

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CASRV   Size of Code Area 36 Hw

Stack Requirement: 18 Hw   Data CSECT Size: 0 Hw

[ ] Intrinsic   [X] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: CASRV

Function: Assigns a REMOTE character string into a temporary character string in a virtual accumulator.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
C1 = C2 where C1 and/or C2 is a REMOTE character string, C1
        is a temporary.
```

[ ] Other Library Modules:

Execution Time (microseconds):

If n = 0: 59.6

If n > 0: 60.8 + 12.6 * (ceiling(n/2))

where n = length(C2)

Input Arguments:

Type	Precision	How Passed	Units
Character(C2)	-	R4→ZCON→descriptor	-

Output Results:

Type	Precision	How Passed	Units
Character(temporary)	-	R2→ZCON→descriptor	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: None

Algorithm:

Sets maxlength of result to 255. If the current length of the input string > maxlength of result, set current length of result to maxlength. Otherwise, set current length of result to current length of input. Find # of halfwords to move by shifting right 1 # of characters. Move halfword by halfword. If there is an odd # of characters, last byte moved is garbage.

<u>CASRV</u>

Secondary Entry Name: <u>CASR</u>

Function: Assigns a REMOTE character string to a character variable.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

`C1 = C2, where C1 and/or C2 is a REMOTE character string.`

[ ] Other Library Modules:

Execution Time (microseconds):

If n = 0: 52.6

If n > 0: 51.8+12.6*(ceiling(n/2))+.8 (if length(C2) > maxlength(C1)),

where n = minimum(length(C2), maxlength(C1)).

Input Arguments:

Type	Precision	How Passed	Units
Character string	-	R4→ZCON→descriptor	-

Output Results:

Type	Precision	How Passed	Units
Character string	-	R2→ZCON→descriptor	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: None

Algorithm:

Same as CASRV, but do not set maxlength of result to 255.

CPASR

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CPASR Size of Code Area 132 Hw

Stack Requirement: 24 Hw Data CSECT Size: 2 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: CPASR

Function: Remote character assignment to a partitioned receiver.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$C1_{I\ TO\ J}$ = C2, where C1 or C2 is a REMOTE character string.

[X] Other Library Modules:

CPASRP

Execution Time (microseconds):

$$77.9 + KA + KB + \sum_{k=1}^{LHP}(6.0 + K_{CLOUT+K}) + KD$$

$$+ \sum_{k=1}^{NCHAR}(8.4 + K_{Ek} + KC_{I+K-1}) + KF$$

$$+ \sum_{k=1}^{RHP}(6.0 + KC_{I+LIN+K-1}) + KG$$

where:

LOUT =	length(C1) before assignment
LIN =	length(C2)
KA =	0 if J $\leq$ LOUT
	13.0 if J > LOUT
LPART =	J - I + 1
KB =	15.8 if LPART > 0 and LIN $\leq$ LPART
	12.0 if LPART > 0 and LIN > LPART
	0 if LPART = 0
LHP =	I-LOUT-1 if I > LOUT + 1
	0 otherwise
KC_X =	19.8 if X is odd
	20.2 if X is even

KD = 3.2 if LHP = 0 and I is odd
4.2 if LHP = 0 and I is even
1.0 if LHP > 0 and LOUT is odd
0 if LHP > 0 and LOUT is even

NCHAR = minimum(LPART,LIN)

KE_X = 13.8 if X is odd
14.4 if X is even

KF = -0.8 if NCHAR > 0
0 if NCHAR = 0

RHP = LPART-LIN if LPART > LIN
0 otherwise

KG = 0 if RHP > 0
0.4 if RHP = 0

Note: If any of LHP, NCHAR, RHP is zero, then that respective summation is also zero.

Input Arguments:

Type	Precision	How Passed	Units
Integer(I)	SP	R5	-
Integer(J)	SP	R6	-
Character(C2)	-	R4→ZCON→descriptor	-

Output Results:

Type	Precision	How Passed	Units
Character(C1)	-	R2→ZCON→descriptor	-

Errors Detected:

Error #	Cause	Fixup
17	Index of first character < 1	Set to 1
	Index of last character > max length of receiver	Set to max length
	Index of last character < index of first character	Return receiver unchanged

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:

If R5 < 1 then send error and set to 1.

If R6 > max length then send error and set to max length.

If R6 > current length of receiver, then update current length of receiver.

If R6 < R5 then send error and exit immediately. Otherwise, move partition character by character.

CPASRP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CPASRP Size of Code Area 16 Hw

Stack Requirement: 146 Hw Data CSECT Size: 0 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: CPASR, CASRPV

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: CPASRP

Function: Remote character string assignment of partitioned input to partitioned output.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$C1_{i\ TO\ j} = C2_{k\ to\ l}$ where C1 and C2 are character strings.

[ ] Other Library Modules:

Execution Time (microseconds):

$$132.3 + KA + KB + KC + \sum_{k=1}^{LHP}(6.0 + KD_{OUTLEN+K}) + KE$$

$$+ \sum_{k=1}^{NCHAR}(8.4 + KF_k + KD_{I+K-1}) + KG$$

$$+ \sum_{k=1}^{RHP}(6.0 + KD_{I+INLEN+K-1}) + KH$$

where:

INPART = L - K + 1 if L $\geq$ K
0 otherwise

INLEN = minimum(INPART, 255)

KA = 76.8 if INPART = 0 and length(C2) = 0
89.0 + 15.8(ceiling(INLEN(2)) if inpart > 0 and K is odd
94.2 + 21.2(ceiling(INLEN(2)) if inpart > 0 and K is even

OUTLEN = length(C1) before assignment

KB = 0 if J $\leq$ OUTLEN
13.0 if J > OUTLEN

OUTPART = J - I + 1 if J $\geq$ I, 0 otherwise

KC = 15.8 if OUTPART > 0 and INLEN = OUTPART
12.0 if OUTPART > 0 and INLEN¹ OUTPART
0 if OUTPART = 0

LHP = I - OUTLEN - 1 if I > OUTLEN + 1, 0 otherwise

KD_X = 9.8 if X is odd
20.2 if X is even

KE =	3.2	if LHP = 0 and I is odd
	4.2	if LHP = 0 and I is even
	1.0	if LHP > 0 and OUTLEN is odd
	0	if LHP > 0 and OUTLEN is even
NCHAR =	minimum(OUTPART,INLEN)	
KF_X =	13.8	if X is odd
	14.4	if X is even
KG =	-0.8	if NCHAR > 0
	0	if NCHAR = 0
RHP =	OUTPART - INLEN if OUTPART > INLEN, 0 otherwise	
KH =	0	if RHP > 0
	0.4	if RHP = 0

Note: If any of LHP, NCHAR, RHP is zero, then the respective summation is also zero.

Input Arguments:

Type	Precision	How Passed	Units
Character (C2)	-	R4→ZCON→descriptor	-
Integer(k)	SP	R5	-
Integer(l)	SP	R6	-
Integer(i||j)	(SP||SP)	R7	-

Output Results:

Type	Precision	How Passed	Units
Character(C1)	-	R2→ZCON→descriptor	-

Errors Detected:

Error #	Cause	Fixup
17	Subscript of character string out of bounds	Set out-of-bounds value to first or last character of associated string

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:

Saves pointer to result in work area, loads address of vac in R1, and branches to CASRPV. Returns, loads result address in R1, loads arg 3 and arg 7 in R5 and R6 respectively and branches to CPASR, and returns.

CSTR

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: CSTR Size of Code Area 18 Hw

Stack Requirement: 18 Hw Data CSECT Size: 0 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: CSTR

Function: Comparison of REMOTE structures.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
IF S1 = S2... Where S1, S2, or both is a REMOTE structure
              occupying n halfwords.
```

[ ] Other Library Modules:

Execution Time (microseconds):

22.8 + 14.8n	if structures compare, where n = # of halfwords in structure.
19.6 + 14.8n	if structures do not compare, where n = index of first non-matching halfwords in structures.

Input Arguments:

Type	Precision	How Passed	Units
Structure (left comparand) S1	-	R2→ZCON→first Hw	-
Structure (right comparand) S2	-	R4→ZCON→first Hw	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Equal/not equal	-	Condition Code	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: None.

Algorithm:

Compares structures, halfword by halfword, until two are found that are different or the end of the structure is reached. If inequality is found then set CC to 1 and return. If equal, then set CC to 0 and return.

MR0DNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MR0DNP Size of Code Area 16 Hw

Stack Requirement: 20 Hw Data CSECT Size: 0 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: MR0DNP

Function: Moves a scalar value to all positions of a partition of a REMOTE double precision matrix.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$M_{A\ TO\ B,C\ TO\ D}$=X; where M is a double precision REMOTE matrix, and either or both 'TO' subscripts may be replaced by the 'AT' subscripts under rules given for matrix types. The indices A,B,C, and D must be literal values. X is a scalar.

[ ] Other Library Modules:

Execution Time (microseconds): 22.8 + n (5.6 + 9.8m)

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-
Integer(n)	SP	R5	-
Integer(m)	SP	R6	-
Integer(outdel)	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
Matrix(n,m)	DP	R2→ZCON→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:

Same as MR0SNP except use 4 * (# columns) as row length in halfwords, and use double precision store.

MR0SNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MR0SNP Size of Code Area 16 Hw

Stack Requirement: 20 Hw Data CSECT Size: 0 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: MR0SNP

Function: Moves a scalar value to all positions of a partition of a REMOTE single precision matrix.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$M_{A\ TO\ B,C\ TO\ D}$=X; where M is a REMOTE single precision matrix and either or both 'TO' subscripts may be replaced by the 'AT' under rules given for matrix types. The indices A,B,C, and D must be literal values. X is a scalar.

[ ] Other Library Modules:

Execution Time (microseconds): 22.8 + n (5.6 + 8.6m) for an n x m partition.

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-
Integer(n)	SP	R5	-
Integer(m)	SP	R6	-
Integer(outdel)	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
Matrix(n,m)	SP	R2→ZCON→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0.

Algorithm:

Find row length in halfwords by SLL # columns, 1 add row length to outdel

```
Loop:   Indexing on # rows, using BCTB
        Loop:   Indexing on # columns, using BCTB
                Store scalar in pointed to output element
        End.
        Add outdel (with row size) to output pointer
End.
```

MR1DNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MR1DNP   Size of Code Area 22 Hw

Stack Requirement: 22 Hw   Data CSECT Size: 0 Hw

☐ Intrinsic   ☒ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: MR1DNP

Function: Moves a partition of a double precision matrix to a partition of a double precision matrix. At least one of the matrices has the REMOTE attribute.

Invoked By:

☒ Compiler emitted code for HAL/S construct of the form:

$M1=M2_{A\ TO\ B,C\ TO\ D}$;

$M1_{A\ TO\ B,C\ TO\ D}=M2$;

where M1 and M2 are double precision matrices, and at least one of M1 and M2 is REMOTE and either or both 'TO' subscripts may be replaced by 'AT' subscripts under rules given for matrix types. The indices A,B,C, and D must be literal values.

☐ Other Library Modules:

Execution Time (microseconds): 28.4 + n (8.2 + 15m) for an n x m partition.

Input Arguments:

Type	Precision	How Passed	Units
Matrix(n,m)	DP	R4→ZCON→0th element	-
Integer(rows)	SP	R5	-
Integer(columns)	SP	R6	-
Integer(indel, outdel)	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
Matrix(n,m)	DP	R2→ZCON→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:

Same as MR1SNP, except use double precision loads and stores and use 4*(# columns) as row length.

MR1SNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MR1SNP   Size of Code Area 22 Hw

Stack Requirement: 22 Hw   Data CSECT Size: 0 Hw

[ ] Intrinsic   [X] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: MR1SNP

Function: Moves a partition of a single precision matrix to a partition of a single precision matrix. Either or both matrices have the REMOTE attribute.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$M1=M2_{A\ TO\ B,C\ TO\ D}$; $M1_{A\ TO\ B,C\ TO\ D}=M2$; where M1 and M2 are single precision matrices, and at least one of M1 and M2 is REMOTE and either or both 'TO' subscripts may be replaced by 'AT' subscripts under rules given for matrix types. The indices A,B,C, and D must be literal values.

[ ] Other Library Modules:

Execution Time (microseconds): 28.4 + n (8.2 + 12.6m) for an n x m partition.

Input Arguments:

Type	Precision	How Passed	Units
Matrix(n,m)	SP	R4→ZCON→0^{th} element	-
Integer(n)	SP	R5	-
Integer(m)	SP	R6	-
Integer(indel, outdel)	(SP||SP)	R7	-

Output Results:

Type	Precision	How Passed	Units
Matrix(n,m)	SP	R2→ZCON→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0.

Algorithm:

Separate indel and outdel into separate registers. Find row size, in halfwords, of result matrix by shifting left 1,

columns
Add rowsize to indel
Add rowsize to outdel
Loop: Indexing on # of rows of output, and using BCTB
Loop: Indexing on # of columns of input, using BCTB
load (single precision) pointed to input element
store (single precision) pointed to output element
End.
Add indel (with row size added) to input pointer
Add outdel (with row size added) to output pointer
End.

MR1TNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MR1TNP Size of Code Area 24 Hw

Stack Requirement: 22 Hw Data CSECT Size: 0 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: MR1TNP

Function: Moves a partition of a double precision matrix to a partition of a single precision matrix. At least one of the matrices has the REMOTE attribute.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$M1=M2_{A\ TO\ B,C\ TO\ D}$;

$M1_{A\ TO\ B,C\ TO\ D}=M2$;

where M1 is a single precision matrix, M2 is a double precision matrix, and at least one of M1 and M2 is REMOTE and either or both 'TO' subscripts may be replaced by 'AT' subscripts under rules given for matrix types. The indices A,B,C, and D must be literal values.

[ ] Other Library Modules:

Execution Time (microseconds): 31.2 + n (7.6 + 13.8m) for an n x m partition.

Input Arguments:

Type	Precision	How Passed	Units
Matrix(n,m)	DP	R4→ZCON→0th element	-
Integer(rows)	SP	R5	-
Integer(columns)	SP	R6	-
Integer(indel, outdel)	(SP||SP)	R7	-

Output Results:

Type	Precision	How Passed	Units
Matrix(n,m)	SP	R2→ZCON→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:

Same as MR1SNP, except use double precision load for index alignment, and use 4 * (# columns) as the length in halfwords of double precision partition.

MR1WNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MR1WNP   Size of Code Area 24 Hw

Stack Requirement: 22 Hw   Data CSECT Size: 0 Hw

[ ] Intrinsic   [X] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: MR1WNP

Function: Moves a partition of a single precision matrix to a partition of a double precision matrix. Either or both matrices have REMOTE attribute.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$M1=M2_{A\ TO\ B,C\ TO\ D}$; $M1_{A\ TO\ B,C\ TO\ D}=M2$; where M1 is a double precision matrix, M2 is a single precision matrix, and at least one of M1 and M2 is REMOTE and either or both 'TO' subscripts may be replaced by 'AT' subscripts under rules given for matrix types. The indices A,B,C, and D must be literal values.

[ ] Other Library Modules:

Execution Time (microseconds): 32.8 + n (8.2 + 13.8m) for an n x m partition.

Input Arguments:

Type	Precision	How Passed	Units
Matrix	SP	R4→ZCON→0th element	-
Integer(rows)	SP	R5	-
Integer(columns)	SP	R6	-
Integer(indel, outdel)	(SP,SP)	R7	-

Output Results:

Type	Precision	How Passed	Units
Matrix	DP	R2→ZCON→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:

Same as MR1SNP, except use double precision stores after zeroing the low half of the floating point register.

MSTR

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: MSTR Size of Code Area 42 Hw

Stack Requirement: 18 Hw Data CSECT Size: 0 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: MSTR

Function: Moves a structure to or from a REMOTE location.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
S1=S2... where S1 or both S1 and S2 are REMOTE
         structures occupying n halfwords.
```

[ ] Other Library Modules:

Execution Time (microseconds): 16.8 + 15n

Input Arguments:

Type	Precision	How Passed	Units
Structure(S2)	-	R4→ZCON→first Hw	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Structure(S1)	-	R2→ZCON→first Hw	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: None.

Algorithm:

Moves structure halfword by halfword.

VR0DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VR0DN Size of Code Area 6 Hw

Stack Requirement: 18 Hw Data CSECT Size: 0 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VR0DN

Function: Moves a scalar to all elements of a double precision vector with the REMOTE attribute.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
V=0; where V is a REMOTE double precision vector.
```

[ ] Other Library Modules:

Execution Time (microseconds): 16.4 + 9.2n, where n = size of vector.

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	DP	R4→ZCON→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:

Same as VR0SN, except use double precision store.

VR0DNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VR0DNP Size of Code Area 10 Hw

Stack Requirement: 18 Hw Data CSECT Size: 0 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VR0DNP

Function: Moves a scalar to all elements of a column of a double precision matrix.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$M_{*,I}$=0; where M is a double precision REMOTE matrix.

[ ] Other Library Modules:

Execution Time (microseconds): 21.2 + 10.0n, n = length of vector result.

Input Arguments:

Type	Precision	How Passed	Units
Scalar	DP	F0	-
Integer(n)	SP	R5	-
Integer(outdel)	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	DP	R2→ZCON→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0, F1.

Algorithm:

Same as VR0SNP, except use double precision stores.

VR0SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VR0SN Size of Code Area 6 Hw

Stack Requirement: 18 Hw Data CSECT Size 0 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VR0SN

Function: Moves a scalar to all elements of a single precision vector with the REMOTE attribute.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
V=0; where V is a REMOTE single precision vector.
```

[ ] Other Library Modules:

Execution Time (microseconds): 16.4 + 8n, n = size of vector.

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	SP	R2→ZCON→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0.

Algorithm:

Store elements in reverse order using the input length both as an index and to control the loop.

VR0SNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VR0SNP Size of Code Area 10 Hw

Stack Requirement: 18 Hw Data CSECT Size: 0 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VR0SNP

Function: Moves a scalar to all elements of a column of a single precision matrix.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$M_{*,I}$=0; where M is a single precision REMOTE matrix.

[ ] Other Library Modules:

Execution Time (microseconds): 21.2 + 8.8n, n = length of vector result.

Input Arguments:

Type	Precision	How Passed	Units
Scalar	SP	F0	-
Integer(n)	SP	R5	-
Integer(outdel)	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	SP	R2→ZCON→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0.

Algorithm:

Store elements one at a time, adding outdel to the pointer after each store.

VR1DN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VR1DN Size of Code Area 8 Hw

Stack Requirement: 18 Hw Data CSECT Size: 0 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VR1DN

Function: Moves a double precision vector to a double precision vector, where at least one of the vectors has the REMOTE attribute.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
V2=V1; where V1 or V2 has been declared a REMOTE vector,
       and V1, V2 are both double precision.
```

[ ] Other Library Modules:

Execution Time (microseconds): 16.4 + 15n, n = length of vector.

Input Arguments:

Type	Precision	How Passed	Units
Vector	DP	R4→ZCON→0^{th} element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	DP	R1→ZCON→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:

Same as VR1SN, except use double precision loads and stores.

VR1DNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VR1DNP Size of Code Area 20 Hw

Stack Requirement: 18 Hw Data CSECT Size: 0 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VR1DNP

Function: Moves a double precision vector to a double precision vector when elements of source or receiver are not contiguous, and at least one has the REMOTE attribute.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$V=M_{*,J}$; where V has been declared a double precision vector, M is a double precision matrix, and V or M

$M_{*,J}=V$; is REMOTE.

[ ] Other Library Modules:

Execution Time (microseconds) : 17.0n + 29.6 if neither input nor output is contiguous.
17.0n + 30.4 if either input or output is contiguous,
where n = length of vector.

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	DP	R4→ZCON→0th element	-
Integer(n)	SP	R5	-
Integer(indel)	SP	R6	-
Integer(outdel)	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	DP	R2→ZCON→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:

Same as VR1SNP, except if indel or outdel = 0, sets to 4, and does double precision loads and stores.

VR1SN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VR1SN Size of Code Area 8 Hw

Stack Requirement: 18 Hw Data CSECT Size: 0 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VR1SN

Function: Moves a single precision vector to a single precision vector where at least one of the vectors has the REMOTE attribute.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
V1=V2; where V1 and/or V2 are REMOTE and both V1 and V2
       are single precision.
```

[ ] Other Library Modules:

Execution Time (microseconds): 16.4 + 12.6n, n = length of vector.

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	SP	R4→ZCON→0th element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	SP	R2→ZCON→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0.

Algorithm:

Loops n times, using length both as index and to control the loop. Load, then store, each element in turn.

VR1SNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VR1SNP Size of Code Area 20 Hw

Stack Requirement: 18 Hw Data CSECT Size: 0 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VR1SNP

Function: Moves a single precision vector to a single precision vector when elements of source or receiver are not contiguous and at least one has the REMOTE attribute.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$V = M_{*,J};$ where V is a single precision vector, M is a single precision matrix, and V or M has the REMOTE attribute.

$M_{*,J} = V;$

[ ] Other Library Modules:

Execution Time (microseconds):

14.6n + 30.4 if either input or output is contiguous.

14.6n + 29.6 if neither input nor output is contiguous, where n=length of vector.

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	SP	R4→ZCON→0^{th} element	-
Integer(n)	SP	R5	-
Integer(indel)	SP	R6	-
Integer(outdel)	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
Vector	SP	R2→ZCON→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:

If outdel = 0, sets it to 2.

If indel = 0, sets it to 2.

Loops 'length' times, moving one element each loop. Adds indel to input pointer and outdel to output pointer after each move.

VR1TN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VR1TN   Size of Code Area 8 Hw

Stack Requirement: 18 Hw   Data CSECT Size: 0 Hw

[ ] Intrinsic   [X] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VR1TN

Function: Moves a double precision vector to a single precision vector, where at least one of the vectors has the REMOTE attribute.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
V1=V2;  where V1 is a single precision vector, V2 is a
        double precision vector, and at least one of V1
        and V2 is REMOTE.
```

[ ] Other Library Modules:

Execution Time (microseconds): 16.4n + 13.8n, n = length of vector.

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	DP	R4→ZCON→0th element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector	SP	R2→ZCON→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:

Same as VR1SN, except use double precision loads.

VR1TNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VR1TNP Size of Code Area 20 Hw

Stack Requirement: 18 Hw Data CSECT Size: 0 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VR1TNP

Function: Moves a double precision vector to a single precision vector, when elements of source or receiver are not contiguous, and at least one of them has the REMOTE attribute.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

$V=M_{*,J}$; where V is a single precision vector, M is a double precision matrix, and V or M is REMOTE.

[ ] Other Library Modules:

Execution Time (microseconds): 15.8n + 30.4 if either input or output is contiguous.
15.8n + 29.6 if neither input nor output is contiguous, where n=length of vector.

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	DP	R4→ZCON→0^{th} element	-
Integer(n)	SP	R5	-
Integer(indel)	SP	R6	-
Integer(outdel)	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
Vector	SP	R2→ZCON→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:

Same as VR1SNP, except if indel = 0, sets it to 4, and does double precision loads.

VR1WN

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VR1WN Size of Code Area 10 Hw

Stack Requirement: 18 Hw Data CSECT Size: 0 Hw

[ ] Intrinsic [X] Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VR1WN

Function: Moves a single precision vector to a double precision vector, where at least one of the vectors has the REMOTE attribute.

Invoked By:

[X] Compiler emitted code for HAL/S construct of the form:

```
V1=V2; where V1 or V2 are remote, V1 double precision, and
       V2 single precision.
```

[ ] Other Library Modules:

Execution Time (microseconds): 20.6 + 13.8n, n = length of vector.

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	SP	R4→ZCON→0^{th} element	-
Integer(n)	SP	R5	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	DP	R2→ZCON→0^{th} element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:

Same as VR1SN, except use double precision store with low half of floating register zeroed.

VR1WNP

HAL/S-FC LIBRARY ROUTINE DESCRIPTION

Source Member Name: VR1WNP Size of Code Area 22 Hw

Stack Requirement: 18 Hw Data CSECT Size: 0 Hw

☐ Intrinsic ☒ Procedure

Other Library Modules Referenced: None

ENTRY POINT DESCRIPTIONS:

Primary Entry Name: VR1WNP

Function: Moves a single precision vector to a double precision vector, when elements of source or receiver are not contiguous, and at least one of them has the REMOTE attribute.

Invoked By:

☒ Compiler emitted code for HAL/S construct of the form:

$V=M_{*,J}$; where V is a double precision vector, M is a single precision matrix, and V or M is REMOTE.

☐ Other Library Modules:

Execution Time (microseconds): 15.8n + 31.2 if either input or output is contiguous.

15.8n + 32.0 if neither input nor output is contiguous.

Input Arguments:

Type	Precision	How Passed	Units
Vector(n)	SP	R4→ZCON→0th element	-
Integer(n)	SP	R5	-
Integer(indel)	SP	R6	-
Integer(outdel)	SP	R7	-

Output Results:

Type	Precision	How Passed	Units
Vector(n)	DP	R2→ZCON→0th element	-

Errors Detected:

Error #	Cause	Fixup
None		

Comments:

Registers Unsafe Across Call: F0,F1.

Algorithm:

Same as VR1SNP, except if outdel=0, sets it to 4, and uses double precision stores, after clearing the low half of the floating point register.

Table 6-1 Index of Library Entry Name

7.0 System Interfaces

This section deals with characteristics and behavior of the HAL/S-FC compiler as related to the environment in which the compiler operates. Specifically, these items are in relation to the host computer in which the compiler is executed.

7.1 Internal System Interfaces

The HAL/S-FC compiler is designed to operate under OS/360 MVT or an equivalent operating system (such as OS/VS2 on IBM 370 equipment). The compiler was developed under Release 21.6 of OS and uses many of the features of that system.

7.1.1 Macro Instructions

All operating system communication is performed via standard assembler language macro instructions as provided with OS MVT. The following list contains the names of all macros executed directly by the HAL/S-FC compiler.

ABEND	BLDL	CHECK	CLOSE	DCB
DCBD	DELETE	FIND	FREEMAIN	FREEPOOL
GET	GETBUF	GETMAIN	LOAD	NOTE
OPEN	POINT	PUT	READ	RETURN
SAVE	SPIE	STIMER	STOW	TIME
TTIMER	WRITE			

The forms of some of these macros require further explanation:

FREEMAIN - All FREEMAIN macros use the SP parameter to indicate subpool 22. Both freeing of single elements of storage and freeing of an entire subpool are performed.

GETMAIN - All requests for main storage are made with the SP operand specifying subpool 22. GETMAINs are done for both single elements of storage of specific size and once, during compiler initialization, for a variable region using the VC form of GETMAIN. This initialization GETMAIN obtains the largest contiguous element of memory available in the job step region. This memory (assigned to subpool 22) is used to hold executable compiler code and as a data area for the compiler.

STIMER - The STIMER macro with the TASK option is used to start an accounting of CPU time used by the compiler.

TTIMER - The TTIMER macro is used to test the TASK interval timer as started by the STIMER macro to determine elapsed CPU time at various points in a compilation.

7.1.2 Dynamic Invocation of the Compiler

The HAL/S-FC compiler may be dynamically invoked by another processing program. The details of this interface are controlled by the *HAL/SDL ICD*.

The dynamic invocation capability allows:

- specification of a parameter string to be acted upon by the compiler,
- specification of an alternate DDNAME list for those DD cards referenced by the compiler, and
- specification of communication areas in which the compiler will supply information to the invoking program.

The compiler takes the following actions to restore its environment upon return to the program which performed the invocation.

- All DCBs opened by the compiler are closed and any automatically acquired buffers are FREEPOOLed.
- All GETMAINed storage is FREEMAINed.
- The SPIE exit (if any) is restored to its status upon entering the compiler.

7.1.3 OS/360 Access Methods

In performing input/output processing the HAL/S-FC compiler uses the OS Data Management Access Methods:

BSAM QSAM BPAM

No other access methods are used, and all datasets manipulated by the compiler are standard OS/360 datasets.

7.2 User or External System Interfaces

The majority of ways in which users of the HAL/S-FC compiler interact with the compiler are described in Sections 2 through 5 of this document. However, the primary vehicle for user communication with this system is Job Control Language which is a part of the compiler's interface to the system in which it operates. This subsection describes the two areas of external or user interfaces to the system:

1) user-defined options acted upon by the compiler, and
2) the JCL with which the user defines the compiler's data and hence the environment in which the compiler is to operate.

7.2.1 User-defined Options

The HAL/S-FC compiler has a number of optional features which may be exercised by the user. These options are indicated via key word parameters passed to the compiler in the standard OS/360 method. The options are either passed to the compiler during dynamic invocation as described in the *HAL/SDL ICD*, or are passed via the PARM field on the EXEC card in the JCL invoking the compiler. A list of these options and their effects may be found in Section 5 of the *HAL/S-FC User's Manual*.

7.2.2 Job Control Language Specification

JCL is the means by which any user of the compiler defines the set of data upon which the compiler is to operate. This JCL is therefore the first interface of the user and the compiler. Once this set of data is specified, all other interfaces with the user are through this data in the manner described in preceding chapters. The remainder of this subsection consists of two parts:

1) a listing of some typical JCL for compiler invocation; and
2) a chart describing the uses, presumed attributes, and access methods for all DD cards.

```
//HALFC      PROC OPTION=,LEVEL=HALS101                         00010000
//HAL        EXEC PGM=MONITOR,REGION=350K,TIME=1,               00020000
//           PARM='NOZCON,&OPTION'                              00030000
//STEPLIB    DD DISP=SHR,DSN=&LEVEL..MONITOR                    00040000
//PROGRAM    DD DISP=SHR,DSN=&LEVEL..COMPILER                   00050000
//SYSPRINT   DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,              00060000
//           BLKSIZE=3458)                                      00070000
//LISTING2   DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,              00080000
//           BLKSIZE=3458)                                      00090000
//OUTPUT3    DD UNIT=SYSDA,DISP=(MOD,PASS),SPACE=(CYL,(1,1)),   00100000
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=400),               00110000
//           DSN=&&HALOBJ                                       00120000
//OUTPUT4    DD SYSOUT=B,DCB=(RECFM=FB,LRECL=80,                00130000
//           BLKSIZE=400)                                       00140000
//OUTPUT5    DD DISP=(MOD,PASS),DSN=&&HALSDF,                   00150000
//           SPACE=(TRK,(2,2,1)),UNIT=SYSDA,                    00160000
//           DCB=(RECFM=F,LRECL=1680,BLKSIZE=1680)              00170000
//OUTPUT6    DD DISP=(MOD,PASS),DSN=&&TEMPLIB,                  00180000
//           SPACE=(TRK,(2,2,1)),UNIT=SYSDA,                    00190000
//           DCB=(RECFM=FB,LRECL=80,BLKSIZE=1680)               00200000
//OUTPUT7    DD DUMMY,DCB=(RECFM=FM,LRECL=121,BLKSIZE=121)      00210000
//ERROR      DD DISP=SHR,DSN=&LEVEL..ERRORLIB                   00220000
//FILE1      DD UNIT=SYSDA,SPACE=(CYL,3)                        00230000
//FILE2      DD UNIT=SYSDA,SPACE=(CYL,3)                        00240000
//FILE3      DD UNIT=SYSDA,SPACE=(CYL,3)                        00250000
//FILE4      DD UNIT=SYSDA,SPACE=(CYL,3)                        00260000
//FILE5      DD UNIT=SYSDA,SPACE=(CYL,3)                        00270000
//FILE6      DD UNIT=SYSDA,SPACE=(CYL,3)                        00280000
```

Typical JCL for Compiler Invocation

Compiler DDNAMES, Uses, and Requirements

DDNAME	FUNCTION	DEVICE REQUIREMENTS	LRECL	RECFM	BLKSIZE	BUFNO[3]	DSORG	ACCESS METHOD, MACRF
PROGRAM	Executable compiler phases	direct access magnetic tape	7200	F	7200	0	PS	BSAM, R
SYSPRINT	Primary listing	printer intermediate storage	133	FBA	3458[1]	1	PS	QSAM, PL
LISTING2	Secondary unformatted listing	printer intermediate storage	133	FBA	3458[1]	1	PS	QSAM, PL
OUTPUT3	Object module output	direct access magnetic tape	80	FB	400[1]	1	PS	QSAM, PL
OUTPUT4	Duplicate object module output	direct access magnetic tape	80	FB	400[1]	1	PS	QSAM, PL
OUTPUT5	Simulation data file output	direct access	1680[4]	F[4]	1680[4]	0	PO	BPAM, W
OUTPUT6	Template search and creation	direct access	80[2]	FB	1680[2]	1	PO	BPAM, WR
OUTPUT7	Pseudo-assembly listing for linkedit ABSLIST function	direct access magnetic tape	133	FBM	3458[1]	1	PS	BSAM, PL
ERROR	Compiler error message retrieval	direct access	80	FB	400	1	PO	BPAM, R
FILE1	Phase 1, Optimizer HALMAT work file Auxiliary HALMAT file	direct access	7200	F	7200	0	PS	BSAM, RWP
FILE2	Literal communication area	direct access	1560	F	1560	0	PS	BSAM, RWP
FILE3	Phase 1 Init/Const work area Phase 2 code gen. work area	direct access	1600	F	1600	0	PS	BSAM, RWP
FILE4	Phase 2 HALMAT work file	direct access	7200	F	7200	0	PS	BSAM, RWP
FILE5	Phase 3 paging area	direct access	1680	F	1680	0	PS	BSAM, RWP
FILE6	Statement data communication area	direct access	512	F	512	0	PS	BSAM, RWP
SYSIN	Primary source input	intermediate storage	80≤ LRECL≤ 132	FB	legal multiple of LRECL[1]	1	PS	QSAM, GL
INCLUDE	Secondary source input	direct access	80≤ LRECL≤ 132	FB	legal multiple of LRECL[1]	1	PO	BPAM, R
ACCESS	ACCESS Rights control	direct access	80[2]	FB	1680[2]	1	PO	BPAM, R

NOTES:

1. BLKSIZE value may be altered by user to any installation-legal value.
2. Compiler will use LRECL and BLKSIZE supplied by user.
3. BUFNO may be specified by user for any PS type datasets.
4. Defaults are shown; Records are always written as 1680 blocks but user-supplied attributes will be retained.

8. 0 PASS/BFS Differences

8.1 Introduction and Background

This section outlines the differences between the Primary Avionics Software System (PASS) and the Backup Flight Software (BFS) versions of the HAL/S-FC merged compiler as of release BFS 7v0. The merging of PASS and BFS compiler source code was authorized via CR11114. BFS 7v0 was a merge of the separate PASS and BFS compiler source code designed to reduce the sustained engineering costs of maintaining two separate compilers. The PASS compiler version 23v2 was used as a baseline for adding BFS version 6v0 unique code and certain PASS and BFS unique discrepancy reports (DRs) and change requests (CRs). Although much of the source became common between the two compilers, there exist differences that can be attributed to required PASS/BFS interfaces and certain desired but not required compiler features. For more information, refer to the *Backup Operating System Interface Control Document (BOS ICD, OV102)*. BFS 7v0 is functionally equivalent to PASS 23v2 except for the differences described in the following sections.

8.2 Interface Differences (Required)

8.2.1 Operating Systems (BOS vs. FCOS)

The BFS compiler system interfaces with the Backup Operating System (BOS) while the PASS compiler interfaces with the Flight Computer Operating System (FCOS).

The SVCI statement is implemented for BFS and must be the last statement before the CLOSE statement in a HAL/S program. SVCI is not implemented for PASS since an automatic SVC is generated.

The BFS compiler system contains Initial Entry processing using a carry bit parameter passed in by the BOS. HAL/S programs use this to determine if a task was

1) inactive prior to current entry
2) active, but normal sequence of execution was interrupted
3) OFF - active, normal sequence of execution

All code areas, as well as constants and literals in data areas, are protected for BFS. PASS groups all data areas in Control Sections (CSECTs) and cannot selectively protect a portion of a CSECT. Therefore, data areas are unprotected for PASS.

Real-time statements SCHEDULE, TERMINATE, CANCEL, WAIT, UPDATE PRIORITY, SIGNAL, SET, RESET, SEND ERROR, RUNTIME, CLOCKTIME, DATE, PRIO, ERRGRP, and ERRNUM are disallowed in BFS due to the BOS's synchronous nature. FCOS is asynchronous, therefore the aforementioned functions are allowed in PASS.

BFS program and task names are formed with a "$" appended to the front of a seven character (maximum) non-underscored HAL/S program or task name. PASS program and task names are formed with two possible characters ($0-$9) appended to the front of a six character name.

UPDATE blocks and EXCLUSIVE procedures or functions are allowed in PASS, however they are disallowed in BFS.

Block definitions are generated differently for BFS and PASS. When a BFS program and task is invoked, the stack address is already in Register 0 (R0). Additionally, an SVC 15(3) instruction is always generated for alternate entry processing for BFS. For PASS, the stack address has to be loaded into R0 at the beginning of a program or task and no SVC 15(3) instruction is generated (no-SDL only).

8.2.2 Linkage Editors (PILOT vs. AP101)

At the start of the HAL/S compiler development, BFS used a different hardware platform than PASS. The BFS system used the ECLIPSE, while PASS used the IBM 360 mainframe. Therefore, the BFS object code format was different and the PILOT linker was created. The ECLIPSE system was found to be too slow, so BFS switched to the IBM platform. However, BFS still uses PILOT as a linker versus the AP101 linker. Thus, the BFS system generates PILOT-formatted object code, while the PASS system generates IBM 360-formatted object code.

8.2.3 Compiler Features

Major Function ID (MFID) is a Type 2 option passed to the PASS compiler that PASS flight software (FSW) uses for grouping modules together for job related purposes. The MFIDs are stored in the #E CSECT (Process Directory Entry) bits 11-15 in the 6th half word. The PASS compiler will generate a Process Directory Entry for all programs and tasks within a compilation unit. FCOS must have a Process Directory Entry (#E). The BFS compiler does not generate a #E CSECT (and likewise a MFID) since this CSECT is used for scheduling, canceling, or terminating events which are unimplemented features in the BFS compiler.

Specifying the NOSCAL option in BFS specifically inhibits the use of the SCAL and SRET instructions for subroutine linkage, even if the MICROCODE option was also chosen. MICROCODE and NOSCAL together thus cause BAL linkage to be used instead of the SCAL/SRET instructions. If NOMICROCODE was specified, neither SCAL nor NOSCAL has any effect.

The DATA_REMOTE directive is restricted in BFS (CR11142, BFS 8.0) due to BOS incompatibility, even though the source code to use it exists in the compiler. If used by BFS, a "B102-UNIMPLEMENTED FEATURE OF HAL/S CALLED FOR", severity 2 error message is emitted.

BFS does not support the NAME TASK and NAME PROGRAM constructs.

8.3 Compiler Feature Differences (Not Required)

8.3.1 Changes Due To CRs/DRs

CR8301 was implemented in PASS release 20v1. This CR changed the Branch instruction BCR to BCRE, and is PASS unique since its implementation would cause object code changes in Backup Flight Software.

CR8348 was implemented in PASS release 20v1. This CR changes Address Constants (ADCONs) offsets, condenses branch instructions and is PASS unique since its implementation would cause object code changes generated by Backup Flight Software.

DR101925, implemented in BFS release 7v0, changed the object file from OUTPUT8 to OUTPUT3. This DR also changed the data set organization of OUTPUT3 in the Monitor member, MONITOR, from sequential to partitioned for BFS since BFS compiler source contains a Monitor call that requires OUTPUT3 to be partitioned. This differs from PASS which uses sequential organization to support the PASS FSW tool Program Maintenance Facility (PMF).

CR11114 changed the Monitor member COMPOPT to set apart the option bits used by each compiler.

CR11114 added the ERRORLIB members PR1-PR5 and DR106214 added ERRORLIB member PR6. These error messages can only be emitted by the BFS compiler since the invocation of these members are contained in BFS unique code.

BIX loop combining related to optimization was made PASS specific since its implementation would cause object code differences generated by Backup Flight Software. Note, BIX loop combining could not be attributed to a particular CR or DR but was implemented before PASS 19v0.

CR13538 changed the object code for YCON to ZCON conversions so that the OHI instruction would not be emitted for BFS. This was necessary since the BFS Pilot does not set the Most Significant Bit (MSB) for ZCON data in Sector 0. The OHI was removed to prevent incorrect NAME compares with data converted using the YCON to ZCON routine in the compiler.

8.3.2 Functions Not Implemented In BFS Compiler

The macros %NAMEADD and %NAMEBIAS are not implemented in the BFS compiler and will generate an "XM1 - %NAMEADD (%NAMEBIAS) IS A NON-EXISTENT %MACRO" error message if the user tries to invoke them.

8.4 BFS/Pass Differences By Compiler Subsystem

MONITOR	PASS1	PASS2	PASS3	OPT
OLDTPL (BFS)/ MFID (PASS)-options	OLDTPL/MFID	Intial entry	CSECT Names	BIX Loop Combining
SCAL (BFS-options	%SVCI	Constant and Literal protection		
OUTPUT3 -Sequential (PASS) -PDS (BFS)	%NAMEADD	Real-time statement		
	%NAMEBIAS	%SVCI		
		Object code format		
		Process Directory Entry		
		UPDATE Blocks		
		EXCLUSIVE procedures		
		Alternate entry processing		
		Program and task names		
		BCR/BCRE instruction (CR8348)		
		ADCON offsets (CR8348)		
		Branch Condensing (CR8348)		
		OHI in YCON to ZCON conversions (CR 13538)		

Figure 8-1

8.5 Summary of PASS/BFS Differences

BFS	**PASS**
• BOS	• FCOS
• Pilot linker	• AP101 linker
• Pilot - formatted object code	• 360 - formatted object code
• SVCI implemented. Must be last statement before CLOSE	• SVCI not implemented. Automatic SVC generated. (see Table 7-1 for a list of SVC Options)
• Initial Entry processing - carry bit	• No Initial Entry support
• Constants and literals in data areas are protected and unprotected on a half word basis	• All data areas are unprotected and code areas protected on a CSECT basis
• SCHEDULE, TERMINATE, CANCEL, WAIT, UPDATE PRIORITY, SIGNAL, SET, RESET, SEND ERROR, RUNTIME, CLOCKTIME, DATE, PRIO, ERRGRP, ERRNUM are disallowed	• The mentioned functions are allowed
• No Process directory entry	• Process directory entry (#E - stacks and flags)
• Program and task names formed with '$' appended to front of a 7 character name	• Program and task names formed with 2 character($0 - $9) appended to front of a 6 character name

Figure 8-2

Summary of PASS/BFS Differences (continued)

BFS	PASS
• UPDATE blocks and EXCLUSIVE procedures or functions are disallowed	• Allowed
• Block definition of Program and Task	• Block Definition of Program and Task
• When program or task is invoked, stack address is already in R0	• The stack address has to be loaded into R0 at the beginning of program or task
• An SVC 15(3) instruction is always generated for alternate entry processing	• No SVC 15(3) instruction is generated
• SCAL inhibits use of SCAL and SRET instructions	•
• MICROCODE and NOSCAL results in BAL linkage use	•
• If NOMICROCODE specified neither SCAL nor NSCAL has any effect	•
•	• Allows specification of the Major Function IDs

Figure 8-3

Table 8-1 SVC Options

FFFF	TASK QUIT- TERMINATE CURRENT TASK
0000	TASK NEXT- GO TO NEXT TASK
000F	ERROR RESTART
0010	MSEC CARD 07- NONCRITICAL I/O REQUEST
0011	MSEC CARD 11- NONCRITICAL I/O REQUEST
0012	MSEC RESET
0013	MSEC CRITICAL I/O
0014	HAL/S RTL ERROR
0015	ASYNCHRONOUS I/O REQUEST
0016	TASK ATTACH REQUEST*
0017	TASK DETACH REQUEST*
0018	TASK STOP REQUEST*
0019	MAKE A GPC ERROR LOG ENTRY
001A	EIU1 MSTR RESET
001B	EIU1 STAT OVERRIDE
001C	EIU2 MSTR RESET
001D	EIU2 STAT OVERRIDE
001E	EIU3 MSTR RESET
001F	EIU3 STAT OVRD
*	The SVC code is the first halfword of a list of actions to be performed. The actual parameter passed by the SVC instruction is the address of the list that contains the SVC code and address and mask pairs.

This page intentionally left blank.

Appendix A Error Classifications

Note: "ƀ" denotes a blank.

CLASS A:	ASSIGNMENT STATEMENTS
A	ARRAY ASSIGNMENT399
V	COMPLEX VARIABLE ASSIGNMENT
ƀ	MISCELLANEOUS ASSIGNMENT
CLASS B:	COMPILER TERMINATION
B	HALMAT BLOCK SIZE
I	INTERNAL ERRORS
N	NAME SCOPE NESTING
S	STACK SIZE LIMITATIONS
T	TABLE SIZE LIMITATIONS
X	COMPILER ERRORS
ƀ	MISCELLANEOUS
CLASS C:	COMPARISONS
ƀ	GENERAL COMPARISONS
CLASS D:	DECLARATION ERRORS
A	ATTRIBUTE LIST
C	STORAGE CLASS ATTRIBUTE
D	DIMENSION
I	INITIALIZATION
L	LOCKING ATTRIBUTE
N	NAME
Q	STRUCTURE TEMPLATE TREE ORGANIZATION
S	FACTORED/UNFACTORED SPECIFICATION
T	TYPE SPECIFICATION
U	UNDECLARED DATA
ƀ	MISCELLANEOUS
CLASS E:	EXPRESSIONS
A	ARRAYNESS
B	BIT STRING EXPRESSIONS
C	CROSS PRODUCT
D	DOT PRODUCT
L	LIST EXPRESSIONS
M	MATRIX EXPRESSIONS
N	NAME
O	OUTER PRODUCT
V	VECTOR EXPRESSIONS
ƀ	MISCELLANEOUS EXPRESSIONS

CLASS F:	FORMAL PARAMETERS & ARGUMENTS
D	DIMENSION AGREEMENT
N	NUMBER OF ARGUMENTS
S	SUBBIT ARGUMENTS
T	TYPE AGREEMENT
~~b~~	MISCELLANEOUS

CLASS G:	STATEMENT GROUPINGS (DO GROUPS)
B	BIT TYPE CONTROL EXPRESSION
C	CONTROL EXPRESSION
E	EXIT/REPEAT STATEMENTS
L	END LABEL
V	CONTROL VARIABLE

CLASS I:	IDENTIFIERS
L	LENGTH
R	REPLACED IDENTIFIERS
S	QUALIFIED STRUCTURE NAMES

CLASS L:	LITERALS
B	BIT STRING
C	CONVERSION TO INTERNAL FORMS
F	FORMAT OF ARITHMETIC LITERALS
S	CHARACTER STRING

CLASS M:	MULTILINE FORMAT
C	OVERPUNCH CONTEXT
E	E-LINE
O	OVERPUNCH USE
S	S-LINE
~~b~~	COMMENTS

CLASS P:	PROGRAM CONTROL & INTERNAL CONSISTENCE
A	ACCESS CONTROL
C	COMPOOL BLOCKS
E	EXTERNAL TEMPLATES
F	FUNCTION RETURN EXPRESSIONS
L	LABELS
M	MULTIPLE DEFINITIONS
P	BLOCK DEFINITION
R	ON ERROR/SVCI MACRO (ONLY EMITTED BY BFS COMPILER)
S	PROCEDURE/FUNCTION TEMPLATES
T	TASK DEFINITIONS
U	CALLS FROM UPDATE BLOCKS
~~b~~	MISCELLANEOUS

```
CLASS Q:   SHAPING FUNCTIONS
A          ARRAYNESS
D          DIMENSION INFORMATION
S          SUBSCRIPTS
X          ARGUMENT TYPE

CLASS R:   REAL TIME STATEMENTS
E          ON/SEND ERROR STATEMENTS
T          TIMING EXPRESSIONS
U          UPDATE BLOCKS

CLASS S:   SUBSCRIPT USAGE
C          SUBSCRIPT COUNT
P          PUNCTUATION
Q          PRECISION QUALIFIER
R          RANGE OF SUBSCRIPT VALUES
S          USAGE OF ASTERISKS
T          SUBSCRIPT TYPE
V          VALIDITY OF USAGE

CLASS T:   I/O STATEMENTS
C          CONTROL
D          DEVICE NUMBER
ƀ          MISCELLANEOUS

CLASS U:   UPDATE BLOCKS
I          IDENTIFIER USAGE
P          PROGRAM BLOCKS
T          I/O

CLASS V:   COMPILE-TIME EVALUATIONS
A          ARITHMETIC OPERATIONS
C          CATENATION OPERATIONS
E          UNCOMPUTABLE EXPRESSIONS
F          FUNCTION EVALUATION

CLASS X:   IMPLEMENTATION DEPENDENT FEATURES
A          PROGRAM ID DIRECTIVE
D          DEVICE DIRECTIVE
I          INCLUDE DIRECTIVE
M          %MACRO
Q          INDIRECTION
R          DATA_REMOTE DIRECTIVE
S          LANGUAGE (E.G. $$$SUBSET)
U          UNKNOWN OR INVALID DIRECTIVE
```

V	VERSION DIRECTIVE
CLASS Y:	ADVISORY MESSAGES
A	ASSIGNMENTS
C	COMPARISONS
E	EXPRESSIONS
F	FORMAL PARAMETERS AND ARGUMENTS
CLASS Z:	PRODUCE ‘TRAP’ MESSAGES FROM THE COMPILER
B	BIT INSTRUCTION
C	%COPY
O	OPTIMIZER
P	REGISTER PRESSURE
S	SUBBIT EXPRESSION

Appendix B Revision History

Revision	Release	Date	CR/DR Number	Sections Changed	
03	7.0	03/16/75			
04	-	Unknown			
05	11.0	03/01/76			
06	13.0	01/15/77			
07	21.7	08/12/88			
08	23.1	02/04/91		Total Reprint	
09	24.0	03/30/92			
10	BFS7.0	11/12/92			
11	BFS8.0	03/15/93			
12	24.1, 25.0	09/03/93			
13	24.2	10/22/93			
14	25.1/9.1	01/11/94			
15	21.B	02/15/94			
16	26.0/10.0	09/02/94			
17	27.0/11.0	12/01/95		Total Reprint	
18	27.1/11.1	07/01/96		pp. vii, 3-19, 3-19A, 3-22, 3-23, 3-36, 3-77, 3-78, 5-1, 5-1A, 5-3, 5-9, 5-10, 5-17, 5-18 thru 5-33 deleted, 5-43 thru 5-55, 5-57 thru 5-60, 5-73, 5-114, 5-116 thru 5-125, 5-141, 5-152 thru 5-159, 5-345 thru 5-355, 5-430 thru 5-440, 5-543, 5-544, 7-3, C-1	
19	28.0/12.0	08/19/1997		Total Reprint to Bring to HAL Documentation Standards and HTML Compatibility	
			CR11148	5.2.7	- pp. 5-14
			CR12709	2.6.1	- pp. 2-6, 2-7, 2-8
				3.1.1.3	- p. 3-5
				3.1.1.5	- p. 3-8
				3.1.7.4	- p. 3-35
				3.1.9.2	- p. 3-37
				3.1.13.	- pp. 3-49, 3-51
				3.1.13.	- pp. 3-51, 3-52
				3.1.14.	- p. 3-53
				3.1.15.	- p. 3-58
				5.2.5.2	- p. 5-9
				5.3.1	- pp. 5-36, 5-37, 5-48
				5.3.2	- p. 5-90
				5.3.7	- p. 5-364
				7.2.1	- p. 7-2
				7.2.3	- p. 7-2
				App. A	- p. A-5

			CR12713	2.1 2.6.1	- p. 2-2 - pp. 2-6, 2-7, 2-8
			CR12432A	3.1.15.3 2.3	- p 3-55 - p. 2-4 - p. 5-10, 5-11, 5-13, 5-23, 5-24, 5-25, 5-26, 5-34, 5-36, 5-38, 5-44, 5-45, 5-45, 5-46, 5-47, 5-49, 5-50, 5-53, 5-55, 5-58, 5-91, 5-211

This is the Last Page of the Document.

DRD NUMBER: 1.4.3.8-b DOCUMENT NUMBER: USA003089

TITLE: HAL/S-FC Compiler System Specification

NASA-JSC

*BV	N. Moses
MS4	D. Stamper
EV111	EV Library (D. Wall)

USA-Houston

*USH-121G	SFOC Technical Library
USH-634G	Abel Puente
USH-64A6X	L.W. Wingo
USH-633L	Anita Senviel
USH-633L	Benjamin L. Peterson
USH-633L	Cory L. Driskill
USH-633L	Judy M. Hardin
USH-633L	Mark E. Lading
USH-633L	Quinn L. Larson
USH-633L	James T. Tidwell
USH-633L	Vicente Aguilar
USH-633L	Betty A. Pages
USH-633L	Jeremy C. Battan
USH-633L	George H. Ashworth
USH-634L	Mark Caronna
USH-634L	Burk J. Royer
*USH-635L	Joy C. King
USH-635L	Ling J. Kuo
USH-635L	Trang K. Nguyen
USH-635L	Billy L. Pate
USH-635L	Karen H. Pham
*USH-635L	Dan A. Strauss
USH-635L	Pete Koester
USH-632L	Renne Siewers
*USH-635L	Barbara Whitfield (2)

Boeing

HS1-40	B. Frere

blake.a.frere@boeing.com

* Denotes hard copy

Submit NASA distribution changes, including initiator's name and phone number, to JSC Data Management/BV or call 281-244-8506. Submit USA distribution changes to USA Data Management/USH-121E or via e-mail to *usadm@usa-spaceops.com*. Most documents are available electronically via USA Intranet Web (*usa1.unitedspacealliance.com*), Space Flight Operations Contract (SFOC), SFOC Data and Deliverables.

Indicates hardcopy

11/23/2005 7:03 AM