https://ntrs.nasa.gov/search.jsp?R=19740017614 2020-03-23T08:13:42+00:00Z

NASA-CR-134283
N74-25727

HAL/S-360 Compiler System Specification

4 February 1977

IR-60-5

Approved: _____
Daniel J. Lickly
HAL Language/Compiler Dept.
Head

Approved: _____
Dr. Fred H. Martin
Shuttle Program Manager

Prepared by the staff of Intermetrics, Inc.
Transcript by V.L. Cripps.

1 CR134283

UPDATE SHEET

Enclosed with each update package you receive is a new update sheet listing the new Version number, the affected pages, and the date of the update.

As you receive each update, replace the old update sheet with the new one. It is important that you refer to the most recent Version number on the update sheet whenever you correspond with Intermetrics concerning this document.

VERSION	AFFECTED MATERIAL	DATE
IR-60-5	The following pages have been updated or added: 1) Title Page/Foreword 2) Table of Contents Page iii/iv 3) Page 1-1/1-2 4) Page 1-3/blank page 5) Page 2-3/2-4 6) Page 2-19/2-20 7) Pages 2-23 thru 2-48 8) Pages 2-53 thru 2-56 9) Pages 2-59 thru 2-64 10) Page 2-67/2-68 11) Pages 2-71 thru 2-89 12) Pages 3-1 thru 3-4 13) Page 3-7/3-8 14) Page 3-11/3-12 15) Page 4-1/4-2 16) Pages 4-5 thru 4-10 17) Pages 4-13 thru 4-18 18) Pages 4-21 thru 4-24 19) Pages 5-5 thru 5-8 20) Page 5-17/5-18 21) Page 5-25/5-26 22) Page 5-33/5-34 23) Page 5-39/5-40 24) Page 5-49/5-50 25) Page 5-57/5-58 26) Pages A-3 thru A-8 27) Page A-93/A-94 28) Page A-103/A-104 29) Pages A-109 thru A-119	2/4/77

FOREWORD

This document was prepared for the Johnson Space Center, Houston, Texas, under contract NAS 9-13864.

Table of Contents (Con't)

		Page
2.1.13	Real Time Statements	2-60
2.1.13.1	WAIT Statement	2-60
2.1.13.2	CANCEL, TERMINATE	2-60
2.1.13.3	SIGNAL, SET, RESET Statements	2-61
2.1.13.4	UPDATE PRIORITY Statement	2-61
2.1.13.5	SCHEDULE Statement	2-62
2.1.14	NAME Operations	2-63
2.1.14.1	NAME Comparisons	2-63
2.1.14.2	NAME Assignments	2-63
2.1.15	%MACRO Operations	2-64
2.1.15.1	%SVC	2-64
2.1.15.2	%NAMECOPY	2-64
2.2	Phase III - Simulation Data File Generation	2-65
2.2.1	SDF Generation	2-65
2.2.1.1	Overall SDF Design	2-65
2.2.2	Phase III Printed Data	2-67
2.2.3	Stand-Alone	2-68
2.3	360 Code Generation for the HAL/S Statement Processor	2-69
2.3.1	Statement Processor Linkage Instructions (Hooks)	2-69
2.3.1.1	Form of Linkage Instructions	2-70
2.3.1.2	Placement of Linkage Instructions	2-70
2.3.2	XMON Address Table	2-75
2.3.3	Statement Processor Data Table	2-78
2.4	Phase 1.5 - The Optimizer	2-79
2.4.1	General Description	2-79
2.4.2	Design Comments	2-79
2.4.3	Optimizations Attempted	2-80
2.4.3.1	Common Subexpression Eliminations	2-80
2.4.3.2	Matrix Transpose Eliminations	2-84

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

Table of Contents (Cont'd.)

		Page
2.4.3.3	Constant Folding	2-84
2.4.3.4	Division Eliminations	2-84
2.4.3.5	Inline Vector/Matrix Computations	2-84
2.4.3.6	Loop Simplification	2-85
2.4.3.7	Loop Combining	2-87
2.4.4	Scope of Optimization	2-88
3.0	SYSTEM CONCEPTS AND INTERFACES	3-1
3.1	HAL Object Module Layout	3-1
3.2	HALLINK	3-5
3.2.1	Stack Size Computation	3-5
3.2.1.1	The Algorithm	3-6
3.2.2	Template Checking	3-7
3.2.3	HALMAP	3-8
3.3	HAL/S Load Module and Operating Environment	3-9
3.4	Processes and the Stack Mechanism	3-9
3.5	Procedures and the Procedure Caller	3-10
3.5.1	Calling	3-10
3.5.2	Exiting	3-15
3.6	Intrinsics	3-15
3.7	User Written Assembly Language Subroutines	3-17
3.7.1	HMAIN	3-17
3.7.2	HENTRY	3-18
3.7.3	HCALL	3-18
3.7.4	HEXIT	3-18
3.7.5	HERROR	3-19
3.8	Include System	3-23
3.9	ACCESS Rights Implementation	3-24
3.10	Error File	3-27
4.0	USER INTERFACE	4-1
4.1	The Compile Step	4-1
4.1.1	Job Control Language	4-1
4.1.2	Inputs	4-6
4.1.2.1	Compiler Options	4-6
4.1.2.2	Source Input	4-11
4.1.3	Outputs	4-12
4.1.3.1	Source and Object Listings	4-12
4.1.3.2	Other Compiler Outputs	4-13

1.0 INTRODUCTION

1.1 Scope of Document

This document specifies the informational interfaces within the HAL/S-360 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-360 Compiler Functional Specification[1]. Familiarization with the Functional Specification is presumed throughout this document.

This Compiler System Specification is for the HAL/S-360 compiler and its associated run time facilities (including real-time) which implements the full HAL/S language[2]. The HAL/S-360 compiler is designed to operate "stand-alone" on any compatible IBM 360/370 computer and within the Software Development Laboratory (SDL) at NASA/JSC, Houston, Texas.

1.2 Outline of the Document

The HAL/S-360 compiler system consists of:

1) a five phase language processor (compiler) which produces IBM 360/370-compatible object modules and a set of simulation tables to aid in runtime verification.

2) a link edit step which augments the standard OS Linkage Editor.

3) a comprehensive run-time system and library which provides the HAL/S operating environment, error handling, a pseudo-real-time executive, and an extensive set of mathematical, conversion, I/O, and diagnostic routines.

The specifications of the information flow and content for this system are contained in this document and arranged as follows:

---

[1] HAL/S-360 Compiler System Functional Specification, 13 July 1973, PDRL #IM004.

[2] HAL/S Language Specification, 14 November 1975, IR #61-7.

Sec. 2 Compiler Information Content specifies the 360 code generation sequences produced by Phase II, and the simulation tables provided for run-time diagnostic purposes.

Sec. 3 System Concepts and Interfaces describes the informational links between the compiler as an entity, and its external system environment. Included are the object module layout, the "HALLINK-step" required during link-edit, the stack mechanism, the HAL/S operating environment (and relationships to OS/360), the interface to assembly language routines and the INCLUDE and Access Rights systems.

Sec. 4 User Interfaces describes the JCL, input and outputs for the compiler, link and execution steps. The compiler's output to the user (i.e. the HAL/S listing) is specified.

Sec. 5 Run-Time Library establishes all the interfaces to the run-time routines viz. matrix-vector arithmetic, mathematical functions, character string manipulations, I/O and conversions, pseudo real-time, error and diagnostic routines. In addition, interfaces to an external monitor are described. Such a monitor could coordinate environment simulation and diagnostics.

Appendix A specifies the intermediate code (HALMAT) emitted by Phase I of the compiler.

Appendix B specifies the internal tables for compilation.

Appendix C specifies the code generation process carried out during Phase II of the compiler.

1.3 Status of Document

This publication represents an updated Compiler System Specification. This document, plus the Compiler System Functional Specification, comprise the complete HAL/S-360 Compiler Specification.

Many features of the HAL/S-360 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-360 compiler system:

- HAL/S-360 Compiler System Functional Specification, 13 July 1973, PRDL #IM004, by Intermetrics, Inc.
- Interface Control Document: HAL/FCOS, Revision 5, Published by IBM Federal Systems Division, Houston, Texas.
- Interface Control Document: HAL/SDL, Revision 6, Published by IBM Federal Systems Division, Houston, Texas.
- HAL/S Language Specification, IR #61-8, Published by Intermetrics, Inc.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Structure templates are internally ordered such that the minimum boundary 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. Note that the storage addresses assigned refer to the actual data beginning, but the base-displacement address includes the negative OFFSET value.

Note that all formal parameters and all AUTOMATIC variables in a REENTRANT PROCEDURE or FUNCTION are based off the stack register (13).

For arrays, the offset is computed as follows for the number of array dimensions: (N_i is the ith array dimension).

# Dim	Offset
0	0
1	-1
2	$(-1\ \ N_2)-1$
3	$((-1\ N_2)-1)N_3-1$

The array OFFSET is then multiplied by the total width of the data type specified. For integers, scalars, bits, and characters, this is the width in bytes to contain one item of data. For vector and matrix types, this is the width in bytes for one item times the total number of items in the vector or matrix.

For structures, the OFFSET is 0 if the structure has no copies. If the structure has copies, the offset is -W, where W is the aligned width of one copy of the structure template.

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;
```

Allocation for the above HAL declarations are as follows (all addresses are in hex):

Alignment	Name	Location	Base	Displacement	(in decimal) Offset
Byte	C	000000	A	000	0
Halfword	B	00000A	A	00A	0
Fullword	A	00000C	A	00C	0
Doubleword	H	000010	A	010	0
Halfword	I	000018	A	00C	-12
Fullword	E	00004C	A	048	- 4
Fullword	G	000060	A	05C	- 4
Fullword	F	000084	A	050	-52
Doubleword	D	0000F0	A	0E8	- 8

2.1.1.3 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 twelve bit displacement. 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 2.2.2 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.

2.1.3.6 Partitioned Bit Assignments. The following sequences assume that R_X has already had the required conversions performed as described in Section 2.1.3.3 or 2.1.3.4. Definitions of I, N_Y, and N_r are as described in Section 2.1.3.3.

Operation	Length of Bit String Y	Code	
$Y_{subscript}$=X;	$N_y \leq 8$ (see note)	SR	R_X, R_X
		IC	R_X, X
		SLL	R_X, N_Y-I
		XR	R_X, R_Y
		N	R_X,=F'mask*'
		XR	R_X, R_Y
		STC	R_X, Y
	$9 \leq N_y \leq 16$ (see note)	LH	R_X, X
		SLL	R_X, N_Y-I
		XR	R_X, R_Y
		N	R_X, F'mask*'
		XR	R_X, R_Y
		STH	R_X, 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.

Operation	Length of Bit String Y	Code	
$Y_{subscript}$=X; (Cont'd)	$17 \leq N_y \leq 32$ (see note)	L	R_x, X
		SLL	R_x, N_y-I
		XR	R_x, R_y
		N	R_x,=F'mask'
		XR	R_x, R_y
		ST	R_x, Y

Note: If the right hand side of the assignment (X) is a BIT literal containing either BIN'0' or BIN(N_R)'1' then the following code is generated:

$Y_{3\ TO\ 5}$ = BIN'0';	N_y = 8	NI	Y, B'11000111'
$Y_{2\ TO\ 4}$ = BIN'111';	N_y = 8	OI	Y, B'01110000'
$Y_{10\ TO\ 12}$=BIN'0';	N_y = 16	LH	R_x, Y
		N	R_x, X'FFFFFF8F'
		STH	R_x, Y
$Y_{11\ TO\ 13}$=BIN'111';	N_y = 16	LH	R_x, Y
		O	R_x,=X'38'
		STH	R_x, Y
$Y_{29\ TO\ 31}$ = BIN'0';	N_y = 32	L	R_x, Y
		N	R_x,=X'FFFFFFF1'
		ST	R_x, Y
$Y_{28\ TO\ 30}$ = BIN'111';	N_y = 32	L	R_x, Y
		O	R_x,=X'1C'
		ST	R_x, Y

Operation	Code
CHAR TO I, I_2	LA 2, X HCALL CTOI*
CHAR TO S	LA 2, X HCALL CTOE**
CHAR TO S_2	LA 2, X HCALL CTOD**
CHAR TO CHAR	No code generated if unsubscripted. If subscripted, code same as component subscripting (see Section 2.1.4.3).
CHAR TO BIT	LA 2, X HCALL CTOB*
CHAR TO BIT$_{@<radix>}$	Same code as CHAR TO BIT, except call to CTOB is replaced as follows:

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

2.1.4.5 Character String Assignments. The following sequences assume that X has been converted as per Section 2.1.4.4 if it is not a character string. 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

* LIBRARY routine leaving result in register 1.
** LIBRARY routine leaving result in register F0.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Operation	Code
Y=X (Y, X are character strings)	LA 3, X LA 2, Y BAL 14, CAS
$Y_{Subscript}$-X;	LA 3, X LA 2, Y LA 1, N_i LA 0, N_f HCALL CPAS*
$Y_{Subscript}$=$X_{Subscript}$;	LA 3, X LA 2, Y LA 1, N_{ix} LA 0, N_{fx} L 4, =F'N_{fy}', N_{iy}' HCALL CPASP*

2.1.5 Vector Matrix Operations

2.1.5.1 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 loop:	LA R_L, 1 LR R_I, R_L SLA R_I, 2 LE F_R, V1(R_I) AE F_R, V2(R_I) STE F_R, temp(R_I) LA R_L, 1($0,R_L$) CH R_L, =H'n' BC 12, loop	same as for "n-code" with n=3.
V1+V2	double loop:	LA R_L, 1 LR R_I, R_L SLA R_I, 3 LD F_R, V1(R_I) AD F_R, V2(R_I) STD F_R, temp(R_I) LA R_L, 1($0,R_L$) CH R_L, =H'n' BC 12, loop	same as for "n-code" with n=3.
V1-V2	Same as for V1+V2 except that an "SE" instruction is used in place of the "AE" instruction ("SD" instead of "AD" for double precision).		

*HCALL is a standard HAL/S calling sequence - see Section 3.7.3.

Operation	Type	n-code	3-code
-V1	single	LA R_L, 1 loop: LR R_I, R_L SLA R_I, 2 LE F_R, V1(R_I) LCER F_R, F_R STE F_R, temp(R_I) LA R_L, 1(0,R_L) CH R_L, =H'n' BC 12, loop	same as for "n-code" with n=3
-V1	double	LA R_L, 1 loop: LR R_I, R_L SLA R_I, 3 LD F_R, V1(R_I) LCDR F_R, F_R STD F_R, temp(R_I) LA R_L, 1(0,R_L) CH R_L, =H'n' BC 12, loop	same as for "n-code" with n=3
V1 ⌀ V2 (yielding nxm matrix)	single	LA 3, V1 loop: LA 4, V2 LA 2, temp-storage-area LA 0, m LA 1, n HCALL VO6SN	LA 3, V1 LA 4, V2 LA 2, temp-storage-area HCALL VO6S3
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 3, V1 LA 4, V2 LA 2, temp-storage-area BAL 14, VX6S3
V1 · V2	single	LA 3, V1 LA 4, V2 LA 0, n HCALL VV6SN	LA 3, V1 LA 4, V2 BAL 14, VV6S3*
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 + M2, or M1 - M2 - M1		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 ⌀ M1 V1: length n M1: n x m	single	LA 3, V1 LA 4, M1 LA 2, temp-storage area LA 0, n LA 1, m HCALL VM6SN	LA 3, V1 LA 4, M1 LA 2, temp-storage area BAL 14, VM6S3

* HCALL is a standard HAL/S calling sequence - see Section 3.7.3.

Operation	Type	n-code	3-code
V1 ⊄ M1	double	Same as for single precision, except that the routines branched to are VM6DN and VM6D3 for the general case and the size 3 case respectively.	
M1 ⊄ V1 M1: n x m V1: length m	single	LA 3, M1 LA 4, V1 LA 2, temp LA 0, n LA 1, m HCALL MV6SN	LA 3, M1 LA 4, V1 LA 2, temp BAL 14, MV6S3
M1 ⊄ V1	double	Same as for single precision, except that the routines branched to are MV6DN for n-code and MV6D3 for 3-code.	
V1 ⊄ I*, V1 ⊄ I2*, V1 ⊄ S	single loop:	LA R_L, 1 LR R_I, R_L SLA R_I, 2 LE F_R, V1(R_I) ME F_R, S STE F_R, temp(R_I) LA R_L, 1(0, R_L) CH R_L, =H'n' BC 12, loop	Same as for "n-code" with n=3.
V1 ⊄ S2	double loop:	LA R_L, 1 LR R_I, R_L SLA R_I, 3 LD F_R, V1(R_I) MD F_R, S2 STD F_R, temp(R_I) LA R_L, 1(0, R_L) CH R_L, =H'n' BC 12, loop	Same as for "n-code" with n=3.
V1/I, V1/I2, V1/S, V1/S2	Same as for V1 ⊄ I, etc., except that a 'DE' instruction is used instead of 'ME' ('DD' instead of 'MD' 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 loop maximum, 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 loop maximum, n, is the product of the row size and the column size of M1.		

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

Operation	Type	n-code	3-code
I ∅ M1, I2 ∅ M1, S ∅ M1, S3 ∅ M1		Same as for M1 ∅ I, etc.	
M1**i (where i is either a literal, or a constant integer)	single	LA 1, i LA 3, M1 LA 4, temp-storage-area LA 2, temp-storage-area LA 0, n HCALL 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 3, M1 LA 2, temp-storage-area LA 0, n HCALL MM15SN	
M1**0	double	Same as for single precision, except branches to MM15DN.	
M1**T M1: m x n	single	LA 3, M1 LA 2, temp-storage-area LA 0, n LA 1, m HCALL MM11SN	LA 3, M1 LA 2, temp-storage-area BAL 14,MM11S3
M1**T	double	Same as for single precision, except the routine branched to is either MM11DN or MM11D3 for m x n matrices and 3 x 3 matrices respectively.	
M1 ∅ M2 M1: l x m M2: m x n	single	LA 3, M1 LA 4, M2 LA 2, temp-storage-area LA 0, 1 LA 1,=F'm,n' HCALL MM6SN	LA 3, M1 LA 4, M2 LA 2, temp-storage-area HCALL MM6S3
M1 ∅ M2	double	Same as for single precision, except that the routines branched to are MM6DN and MM6D3 for the general case and the 3x3 case respectively.	

2.1.5.2 Conditional Operators. The only comparison operators allowed for comparing vectors 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 3, V1 LA 4, V2 LA 0, n BAL 14, VV8SN BC COND, not-true-label	LA 3, V1 LA 4, V2 BAL 14, VV8S3 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 M1,M2: mxn	single	LA 3, M1 LA 4, M2 LA 0, mxn BAL 14,VV8SN BC COND, not-true-label	LA 3, M1 LA 4, M2 LA 0, 9 BAL 14, VV8SN BC COND, not-true-label
M1 <OP> M2	double	Same as for single precision, except that the routine branched to is VV8DN.	

2.1.5.3 Component Subscripting. Possible components of matrices include submatrices, 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 8 instead of 4. Register R4, when used, contains skip values between elements in partitioned matrices (see Section 2.3.1.3).

NTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Operation	n-code	3-code
$S = V_{X_i}$ i: integer literal	LE R_S, V_X + 4 *i STE R_S, S	
$S = V_{X_I}$ I: integer variable	LH R_I, I SLL R_I, 2 LE R_S, V_X (R_I) STE R_S, S	
$V_Y = V_{X\text{subscript}}$ Where subscript defines a vector of size n.	LA 3, V_X LA 2, V_Y LA 0, n BAL 14, VV1SN	LA 3, V_X LA 2, V_Y BAL 14,VV1S3
$M_Y = M_{X\text{subscript}}$ Where subscript defines a matrix of size nxm	LA 3, M_X LA 2, M_Y LA 4, skip-value LA 0, n LA 1, m HCALL 14, MM1SNP	
$V_Y = M_{X\text{subscript}}$ Where subscript defines a vector of length n	LA 3, M_X LA 2, V_Y LA 4, skip-value LA 0, n BAL 14,VV1SNP	LA 3, M_X LA 2, V_Y LA 4, skip-value BAL 14, VV1S3P
$S = M_{X_{i,i}}$	Same as for vectors, except that the displacement off R_S is computed to include the use of two subscripts for matrices.	

2.1.5.4 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 in unchanged.

Operation	n-code	
VECTOR(M_X) Produces vector of size equal to produce of dimensions of matrix: nxm.	LA 3, M_X LA 2, temp-storage-area LA 0, n*m BAL 14, VV1SN*	
MATRIX(V_X,V_Y,V_Z)**	LA 3, V_X LA 2, temp-storage-area BAL 14,VV1S3* LA 3, V_Y LA 2, temp-storage-area+DELTA1 BAL 14,VV1S3* LA 3, V_Z LA 2, temp-storage-area+DELTA2 BAL 14,VV1S3*	address of 4th matrix element address of 7th matrix element

* The pointer to the results of vector-matrix operations are left in register 3. Thus, the instruction to set up register 3 may be inhibited, if it is the same as the result of the previous operation.

** This is an example using several vectors to illustrate the multiple calling of the VV1S3 (or VV1SN) routine. It applies to the VECTOR shaping function.

2.1.5.5 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 $\bar{V} = 0;$.

Operation	Type		n-code	3-code
$V_x = V_y$	single	LA	R_y, Y	Same as for "n-code"
		LA	R_x, X	with n=3.
		MVC	$0(n*4,R_x),0(R_y)$	
$V_x = V_y$	double	LA	R_y, Y	Same as for "n-code"
		LA	R_x, X	with n=3.
		MVC	$0(n*8,R_x),0(R_y)$	
$V_x = 0$	single	LA	R_L, 1	Same as for "n-code"
	loop:	LR	R_I, R_L	with n=3.
		SLA	R_I, 2	
		SER	F_R, F_R	
		STE	F_R, $V_x(R_I)$	
		LA	R_L, $1(0,R_L)$	
		CH	R_L, H'n'	
		BC	12, loop	
$V_x = 0$	double	LA	R_L, 1	Same as for "n-code"
	loop:	LR	R_I, R_L	with n=3.
		SLA	R_I, 3	
		SDR	F_R, F_R	
		STD	F_R, $V_x(R_I)$	
		LA	R_L, $1(0,R_L)$	
		CH	R_L, H'n'	
		BC	12, loop	
$M_x = M_y$ and $M_x = 0$	Same as for vectors, except that the loop maximum, n, is the product of the row size and the column size of the matrix.			

For the following operations:

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

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. A-so, 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) a) 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.

   b) 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, both 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 only one of the variables in a formal parameter and the NEST level of the non-parameterized 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;
         Y(2 TO 3,*) = X(2 TO 3,*) + C(2 TO 3,*);
         B(2 TO 3,*) = P(2 TO 3,*) + Q(2 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 both 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,

$A_{1 \text{ TO } 2,*} = A_{3 \text{ TO } 4,*} + \ldots$ and

$A_{3 \text{ TO } 4,*} = A_{1 \text{ TO } 2,*} + \ldots$ are 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;
      A(1 TO 2,*) = B(3 TO 4,*) + C(3 TO 4,*);     Pairs A-B & A-C
                                                     disjoint
      A(3 TO 4,*) = B(1 TO 2,*) + C(3 TO 4,*);     Pair A-B not neces-
                                                    sarily disjoint
   CLOSE PROC;
   CALL PROC(A(3 TO 6,*),D(3 TO 6,*));     (B(1 TO 2,* is really
                                                   A(3 TO 4,*))
   A(3 TO 4,*) = D(3 TO 4,*) + E(1 TO 2,*);  A,D,E are, by default,
CLOSE EXAMPLE2;                              disjoint because they
                                             are nonidentical
```

If the operation has two operands, both receiver-operand pairs must be disjoint for this condition to be true. The non-identical 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.

2.1.6 Structure Operations

2.1.6.1 Structure Comparisons. Structure comparisons may only be = or ¬ =. The comparison is 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 2.1.1.3). No logical variables are created by comparisons. Instead, branching to the "not-true-label" occurs with the "not-true" condition.

Operation	Code
X <OP> Y	LA R_X, X LA R_Y, Y CLC x-terminal-1, y-terminal-1 BC COND, not-true-label CLC x-terminal-2, y-terminal-2 BC COND, not-true-label ⋮ CLC x-terminal-n, y-terminal-n BC COND, not-true-label ⋮

2.1.6.2 Structure Assignments. The assignment of both major and minor structures consists of loading registers R_X and R_Y with the address of the structure nodes being accessed, followed by a MVC (move characters) which moves the number of bytes specified by width from the locations specified by R_X to the location specified by R_Y.

Operation		Code
Y = X	width $\leq$ 256	LA R_X, X LA R_Y, Y MVC 0(width, R_Y),0(R_X)
	256 width $\leq$ 1024 (n is number of 256 byte blocks)	LA R_X, X LA R_Y, Y MVC 0(256, R_Y), 0(R_X) : : MVC (n-1)*256(256,R_Y),(n-1)*256(R_X) MVC n*256(width mod 256,R_Y),n*256(R_X)
	width > 1024	LA 0, n LA R_X, X LA R_Y, Y BALR R_L, 0 MVC 0(256,R_Y),0(R_X) LA R_X, 256(,R_X) LA R_Y, 256(,R_Y) BCTR 0, R_L MVC n*256(width mod 256,R_Y), n*256(R_X)

METRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

2.1.7 Indexing and Arrayed Statements

2.1.7.1 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. At initial entry to the array loop (see Section 2.1.7.4), 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 2.1.1.3). 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 2.1.7.4. 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 word or doubleword data, it must be moved to another register to perform this shift.

2.1.7.2 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 2.1.1.3). 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 2.1.7.4.

2.1.7.3 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.

2.1.7.4 Arrayness and Loop Generation. This section has two examples of the form of array loops, and how indexing is used within them. The first example uses linear indexing within the loop, while the second uses non-linear indexing. An array loop consists of the following sections: initialization of index values, use of index values to develop the DELTA values (see Section 2.1.1.3) of the operands, actual operation performed on array elements (i.e. assignment, comparison, etc.), and incrementing and testing index values. Note that non-linear and array indexing produce more than one loop. However, only R_a is used for indexing, thus requiring temporary storage of the values of the indices for the outer loops.

Operation

`[X] = [Y]3 AT 2`

Type

```
[X]: ARRAY(3) SCALAR
[Y]: ARRAY(5) SCALAR
     DOUBLE
```

Code

```
        LA   9, 1          } initialize index
loop:   LR   8, 9          }
        SLL  8, 3          } indexing of [Y]
        LR   7, 9          }
        SLL  7, 2          } indexing of [X]
        LD   2, Y+8(8)     }
        STE  2, X(7)       } assignment
        LA   9, 1(0,9)     }
        CH   9,=H'3'       } increment and
        BC   12, loop      } test index
```

`[I] = [V]1,2 TO 3,*:2`

Type

```
[I]: ARRAY(2,4)
     INTEGER
[V]: ARRAY(2,3,4)
     VECTOR
```

Code

```
        LA   9, 1                  } initialize &
outer-loop: ST 9, temp-storage-    } store first
            area                   } index value
        LA   9, 1                  } initialize 2nd index
inner-loop: L  8, temp-storage-    }
            area                   }
        SLA  8, 2                  } indexing of
        AR   8, 9                  } [V]
        MH   8,=H'12'              }
        L    7, temp-storage-      }
             area                  }
        SLA  7, 2                  } indexing of
        AR   7, 9                  } [I]
        AR   7, 7                  }
        LE   0, V+200(8)           }
        BAL  14, FLOATFX           } assignment
        STH  1, I(7)               }
        LA   9, 1(0,9)             } increment &
        CH   9,=H'4'               } test 2nd
        BC   12,inner-loop         } index value
        L    9, temp-storage-      }
             area                  } increment &
        LA   9, 1(0,9)             } test 1st
        CH   9,=H'2'               } index value
        BC   12, outer-loop        }
```

2.1.8 PROCEDURE/FUNCTION Calls

2.1.8.1 Calls to HAL PROCEDURES and FUNCTIONS. The PROCEDURE/FUNCTION calling process consists of two parts: argument set up, and the actual branching to the subroutine's code (see Section 2.1.9.2). Argument set up uses registers 0-4 (as needed) for passing integers or bit strings, or pointers to vectors. matrices, character strings, arrays, or structures; registers F0, F2, and F4 are used as needed to pass scalar arguments. The actual code generated sets up the arguments in these registers in the reverse 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, integer 2);
```

then the registers are loaded in the order: 2 (using LH or L), 1 (using LA to load address of the vector's pointer), F2 (using LE or LD), F0 (using LE if scalar 1 is single precision, or LD if double), and 0 (using LH if integer 1 is single precision, or L if double). Once all of these registers are used, remaining arguments are stored in the run stack for the procedure or function being called. The parameters passed via registers are stored in the stack at the time of invocation. Again, the code will use appropriate integer or floating point instructions depending on the type of the subsequent arguments. If the value of any of these subsequent arguments is in a register, then only a store instruction is generated. Otherwise, both load and store instructions are generated (as shown in the code sequences below). Once all parameters are set up, a BALR is generated to branch to the subroutine.

Operation	No. of ARGS	Code	Alternate Code
Argument Setup	≤ 5 non-scalar and ≤ 3 scalar	LH 4, arg 5	L 4, arg 5 or LA 4, arg 5
		LH 3, arg 4	L 3, arg 4 or LA 3, arg 4
		⋮	⋮
		LH 0, arg 1	L 0, arg 1 or LA 0, arg 1
		LE 4, scalar-arg 3	LD 4, scalar-arg 3
		LE 2, scalar-arg 2	LD 2, scalar-arg 2
		LE 0, scalar-arg 1	LD 4, scalar-arg 1
		BALR 14, 12	
Actual Call		DC AL4(Proc. name)	
Argument Setup	> 5 non-scalar and/or > 3 scalar	LH R, arg n	L R, arg n or L R, arg n
		STH R, temp-storage*	ST R, temp-storage*

* Temp-storage is an area in the stack of the routine to be called, addressed as the stack relative address augmented by the max-stack size of the calling routine.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

```
                    LE   R, scalar-arg n                    LD   R, scalar-
                                                                 arg n
                    STE  R, temp-storage*                   STD  R, temp-
                                                                 storage*
                          :
                    LE   4, scalar-arg 3                    LD   4, scalar-
                          :                                      arg 3
                    LE   0, scalar-arg 1                    LD   0, scalar-
                                                                 arg 1
                    LH   4, arg 5          LA  4, arg 5 or L    4, arg 5
                          :
                    LH   0, arg 1          LA  0, arg 1 or L    0, arg 1
Actual Call
                    BALR 14, 12
                    DC   AL4 (proc-func-name)
```

2.1.8.2 Calls to NONHAL(1) Procedures and Functions. NONHAL(1) is defined as FORTRAN compatible linkage. All arguments are passed as addresses. Unlike FORTRAN, however, addresses of literal values are not passed; instead, addresses of temporary locations containing the literal value are passed.

Example:

```
DECLARE FTSUB  PROCEDURE NONHAL(1)
  :
CALL FTSUB(A,B,C,1);
```

```
                          Code
Argument Setup      LA    R, A
                    ST    R, temp-storage-area
                    LA    R, B
                    ST    R, temp-storage-area+4
                    LA    R, C
                    ST    R, temp-storage-area+8
                    LA    R, 1
                    ST    R, work-area
                    LA    R, work-area
                    ST    R, temp-storage-area+12
                    MVI   temp-storage-area+12, X'80'
                    LA    1, temp-storage-area
Actual Call         ST    13, max-temp-area+4
                    LA    13, max-temp-area
                    L     15, =V(FTSUB)
                    BALR  14, 15
                    L     13, 4(13)
                    L     15, 0(13)
```

* Temp-storage is an area in the stack of the routine to be called, addressed as the stack relative address augmented by the max-stack size of the calling routine.

METRICS INCORPORATED - 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

2.1.9 Block Definition Statements

All of the forms of block definition statements in the following subsections are basically similar so that the following conventions apply. In the constant where the value "name" appears in the code, it refers to the name of the block being defined. If this constant is halfword aligned, the alternate code shown uses: ,X'0' to force alignment. The value of "ℓ" is the length in characters of "name" (or for unlabeled UPDATE blocks and INLINE functions, it is the length of "$NAME"). The value of "n" in the unconditional branch instructions is the relative address of the first executable instruction following the constants' declarations.

2.1.9.1 PROGRAM and TASK Definition. PROGRAM and TASK definitions (as well as external procedure and function definitions) are similar to those for procedures and functions except that the last two load instructions are needed to provide addressability.

Operation	Code	Alternate Code
PROGRAM or TASK Definition	BC 15, n(0, 15) DC AL1(0), AL3(FSIM CSECT #F) DC AL2(max-temp-size) DC AL1(ℓ), C'name'	DC AL1(ℓ), C'name',X'0'
	L 11, 4(0,15) LM 6, 10, 128(11) MVI 0(13),m**	

* Appropriate displacement into the run stack for the called procedure or function.

** m is the code for the ID of the block - see p. 3-12.

2.1.9.2 PROCEDURE and FUNCTION Definition. Both PROCEDURE and FUNCTION definitions are similar to PROGRAM and TASK definitions. However, because floating point registers are not automatically saved, STE instructions are used to save the values, in a temporary storage area, of any used registers (see Section 2.3.8 for information on how F0, F2, and F4 are used in parameter passing).

Operation	Code	Alternate Code
PROCEDURE/FUNCTION header	BC 15, n(0,15) DC AL1(0),AL3(FSIM CSECT address) DC AL2(max-temp-size) DC AL1(ℓ), C'name'	DC AL1(ℓ), C'name' , X'0' STE F0, arg 1 STE F2, arg 2 STE F4, arg 3

2.1.9.3 UPDATE and INLINE FUNCTION Definition. The UPDATE definition process is identical to that for INLINE FUNCTIONs, except that additional instructions are generated to establish the lock group numbers in use. Note that both variabled UPDATE blocks and INLINE functions are referred to with $ left-catenated to their names.

Operation	Code	Alternate Code
UPDATE Definition	BC 15, n(0,15) DC AL1(0), AL3 (FSIM CSECT address) DC AL2(max-temp-size) DC AL1(ℓ), C'$name' LH R0, lock-group-mask HCALL 14, LOCK . .	DC AL1(ℓ) C'$name',X'0'
INLINE FUNCTION Definition	BC 15, n(0,15) DC AL1(0), AL3 (FSIM CSECT address) DC AL2(max-temp-size) DC AL1(ℓ), C'$name' . . .	DC AL1(ℓ), C'$name',X'0'

2.1.10 Flow of Control Statements

2.1.10.1 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
	then label: {executable code for THEN clause}
	:
	BC 15, next-statement
	else-label {executable code for ELSE clause}
	:
	next-statement:
IF <cond exp.> THEN <>;	BC {cond, next-statement executable code for THEN clause}
	:
	next-statement:

2.1.10.2 DO FOR...Loops. The DO FOR loop has two forms: the iterative, and the discrete. They both may also allow termination of the loop by use of the clauses 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;*`	`LA 9, a`
	`test-label: STH 9, I`
	`CH 9,=H'b'`
	`BC 2, exit-label`
	`: } executable code within DO group`
	`repeat**: LA 9, c`
	`AH 9, I`
	`BC 15, test-label`
	`exit-label:`
	`: } code for statement following DO group`
`DO FOR I = a TO b BY c`	
`. UNTIL <cond exp>;`	`MVI temp-storage-are,0 } UNTIL code`
`:`	`LA 9, a`
`END;`	`test-label: STH 9, I`
	`CH 9,=H'b'`
	`BC 2, exit-label`
	`TS temp-storage-area`
	`BC 8, first-statement***`
	`: } code for conditional expression } UNTIL code`
	`BC cond, exit-label`
	`first-statement: . } executable code`
	`: } within DO group`

* Assumes a, b, and c are literal values.
** This is referenced by the REPEAT statement (see Section 2.3.10.5).
*** This is done to avoid testing the <cond exp> until after executing through the loop at least once.

RMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Operation		Code
	repeat*:	LA 9, c AH 9, I BC 15, test-label
	exit-label:	: } code for statement following DO group
DO FOR I = a TO b BY c . WHILE <cond exp>; : END;		LA 9, a
	test-label:	STH 9, I CH 9,=H'b' BC 2, exit-label
		: } code for conditional expression } WHILE code
		BC cond, exit-label
		: } executable code within DO group
	repeat*:	LA 9, c AH 9, I BC 15, test-label
	exit-label:	: } code for statement following DO group
DO FOR I = a_1, a_2,...,a_n; : END;	label 1:	LA 9, a_1 BAL 14, test-label
	label 2:	LA 9, a_2 BAL 14, test-label
		:
	label n:	LA 9, a_n LA 14,exit-label
	test-label:	ST 14, temp-storage-area STH 9, I
		: } executable code within DO group
	repeat*:	L 14, temp-storage-area BCR 15, 14
	exit-label:	: } code for statement following DO Group

* This is referenced by the REPEAT statement (see Section 2.3.10.5).

Operation		Code	
DO FOR I = I1 to I2 BY I3*;		LH	9, I2
.		STH	9, temp-test
.		LH	8, I3
.		STH	8, temp-increment
END;		LH	7, I1
	test-label:	STH	7, I
* (I1, I2, I3 variables)		LA	9, exit-label
		TM	temp-increment, X'80'
		BZ	positive-test
		CH	7, temp-test
		BCR	4, 9 negative increment
		B	loop-begin
	positive test:	CH	7, temp-test
		BCR	2, 9 positive increment
	loop-begin:		code for DO group
	repeat**	LH	7, temp-increment
		AH	7, I
		BC	15, test-label
	exit-label:		.
			.
			.

** Repeat label.

2.1.10.3 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 preceeding subsections.

Operation		Code	
DO WHILE <cond exp>	repeat:	. . .	code for conditional expression
		BC cond, exit-label	
		. . .	code for statements within DO group
		BC 15, repeat	
	exit-label:	. . .	code for statement following DO group
DO UNTIL <cond exp>		BC 15, first-statement	
	repeat:	. . .	code for conditional expression
		BC cond, exit-label	
	first-statement :	. . .	code for statements within DO group
		BC 15, repeat	
	exit-label:	. . .	code for statement following DO group

2.1.10.4 GO TO, REPEAT, EXIT. All of these statements consist of unconditional branches. The REPEAT and EXIT statements are used only in DO groups; REPEAT is restricted to DO FOR, DO WHILE or DO UNTIL groups, and branches to the code which tests whether looping is done or not. Refer to Sections 2.1.10.2 and 2.1.10.3 for the locations of "test-label" and "exit-label".

Operation	Code	
GO TO label	BC 15, label	
REPEAT	BC 15, repeat	repeat is the location of the code which determines whether iteration is finished or not.
EXIT	BC 15,exit-label	exit-label is the location of the code immediately following the end of the DO loop.

2.1.10.5 RETURN. This statement will branch back from the code for a function to the code immediately following the function's call. The value returned by the function is either an integer value, or a pointer in R1, or it may be a scalar value in F0.

Operation	Code	
RETURN <exp>	.	evaluation of exp with scalar results in F0, otherwise with integer result, or pointer in R1
	.	
	.	
	BC 15, 4(0, 12)	

METRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

2.1.10.6 ON ERROR/SEND ERROR,

Operation		Code
OFF ERROR$_{m:n}$	MVI	temp-storage-area*, 0
ON ERROR$_{m:n}$ <statement>	LA	14, ℓ1
	ST	14, temp-storage-area+4
	L	14, =XL4'mmnn0000'
	ST	14, temp-storage-area*
	BC	15, ℓ2
ℓ1:	code for <statement>	
ℓ2:	next statement	

ON ERROR$_{m:n}$ <action> [<event action>]

<action>	α
SYSTEM	1
IGNORE	2

<event action>	β
none	0
SIGNAL	1
SET	2
RESET	3

	LA**	14, e
	ST**	14, temp-storage-area+4
	L	14, =XL4'mmnnααββ'
	ST	14, temp-storage area*

SEND ERROR$_{m:n}$	BAL	14, ERRSND
	DC	X'mmnn'
	DC	AL4(0)

* Area in stack reserved for recording outstanding error conditions for current block.

** Omitted if <event action> = none.

2.1.11.2 Out of Line Functions. Out of line functions require branches to the run time library (see Section 5). Parameters are passed via any or all of registers R0 through R4, and/or registers F0 through F4. The actual registers needed, and the name of the library routine branched to, are specified in the tables of Section 5. Examples are given for representative argument types. Scalar results are returned in register F0; any other types of results are returned by R1.

Operation	Type	Code
COS(X)	scalar, single	LE 0, X HCALL COS
SQRT(X)	scalar, double	LD 0, X HCALL SQRT
ABVAL(X)	vector(n), single	LA 3, V1 HCALL VV9SN
	vector(3), double	LA 3, V1 HCALL VV9D3
TRANSPOSE(X)	matrix(m,n), double	LA 3, X LA 2, temp-storage-area LA 0, n LA 1, m HCALL MM11DN
	matrix(3,3), single	LA 3, X LA 2, temp-storage-area BAL 14, MM11S3
UNIT(X)	vector(n), single	LA 3, X LA 2, temp-storage-area LA 0, n HCALL VV10SN
	vector(3), single	LA 3, X LA 2, temp-storage-area HCALL VV10S3
RANDOMG		HCALL RANDOMG
TRIM(X)	character	LA 3, X LA 2, temp-storage-area HCALL CTRIMV
MAX(X)	array(n)	LA 3, X LA 0, n BAL 14, EMAX

2.1.11.3 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 2.1.2.3, 2.1.3.4, 2.1.4.4, and 2.1.5.4).

2.1.12 I/O Statements

2.1.12.1 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)...		LA 1, n SR 0, 0 HCALL IOINIT :
READALL(n)...		LA 1, n LA 0, 1 HCALL IOINIT :
WRITE(n)...		LA 1, n LA 0, 3 HCALL IOINIT :

2.1.12.2 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 2.1.1.3).

Operation	Type	Code
READ( )..., I, ...	integer, single	. . . initiation LA 2, I HCALL HIN
	integer, double	. . . initation LA 2, I HCALL IIN
READ( ),..., S, ...	scalar, single	. . . initiation LA 2, S HCALL EIN
	scalar, double	. . . initiation LA 2, S HCALL DIN
READ( )..., V, ...	vector(n);single	. . . initiation LA 3, V SR 4, 4 LA 0, 1 LA 1, n HCALL MM20SNP

Operation	Type	Code
READ( )..., V, ...	partitioned vector of length n whose first element is located at 'V+ displacement'	. . . initiation LA 3, V+displacement SR 4, 4 LA 0, 1 LA 1, n HCALL MM20SNP
	vector(n); double (partitioned or not partitioned)	same except branches to MM20DNP
READ( )..., M, ...	matrix(m,n); single	. . . initiation LA 3, M SR 4, 4 LA 0, m LA 1, n HCALL MM20SNP
READ( )..., M, ...	partitioned matrix whose resultant size is mxn, first element is M+displacement.	. . . initiation LA 3, M+displacement LA 4, DELTA LA 0, m LA 1, n HCALL MM20SNP
	matrix(m,n); double (partitioned or not partitioned)	Same except branches to MM20DNP
READ( )..., C, ... or READALL( )...,C,...	character string	. . . initiation LA 3, C HCALL CIN
READ( )..., $C_{m \text{ TO } n}$,... or READALL( )..., $C_{m \text{ TO } n}$,...	partitioned character string	. . . initiation LA 3, C LA 1, m LA 0, n HCALL CINP

Operation	Type	Code
	matrix(m,n); double (partitioned or not partitioned)	same except branches to MM21DNP
WRITE( )..., C,...	character string	. . . initiation LA 3, C HCALL COUT
WRITE( )..., $C_{m \text{ TO } n}$	partitioned character string	. . . initiation LA 3, C LA 1, m LA 0, n HCALL COUTP
WRITE( )..., C_n,...	single partitioned character string	. . . initiation LA 3, C LA 1, n LR 0, 1 HCALL COUTP
WRITE( )..., B,...	bit string (of length n)	. . . initiation SR 0, 0 IC 0, B* LA 1, n HCALL BOUT

Arrayed Output

The actual code generated depends on the type of array. Thus, the code will consist of an array loop (see Section 2.1.7.3) to cause iterative outputting of each array element using the code shown above (corresponding to the array element type).

* In the case shown $n \leq 8$ so that an IC is used; for $8 < n \leq 16$ a LH is used; and for $n > 16$, a L is used.

2.1.13 Real Time Statements

2.1.13.1 WAIT Statement. Except for any form of the WAIT FOR statement, WAIT routines generally require a double precision scalar in register F0 as input arguments. This is shown below using LD instructions, although other means of loading F0 may actually be used (for example WAIT UNTIL I where I is an integer will cause a branch to FXFLOAT which will return with the double precision value properly situated in F0). WAIT FOR <event exp> uses a pointer to the <event exp> in R0.

Operation	Type	Code
WAIT n	n: literal	LD 0, =XL8'floating-point-form-of-n' HCALL WAIT
WAIT X	scalar, double	LD 0, X HCALL WAIT
WAIT FOR DEPENDENT		HCALL WAITDEP
WAIT FOR X	event value	LA 0, event-expression-containing X HCALL WAITFOR
WAIT UNTIL X	scalar, double	LD 0, X HCALL WAITUNTL

2.1.13.2 CANCEL, TERMINATE. When CANCEL or TERMINATE contain the name of a task or program to be cancelled or terminated, then the entry point of the program or task (referred to as: <taskid> in the code below) must be loaded into register R0.

Operation	Type	Code
CANCEL		HCALL CANCEL
CANCEL <taskid>		L 0, taskid HCALL CANCELT*
TERMINATE		HCALL TERMIN
TERMINATE <taskid>		L 0, taskid HCALL TERMINT**

* For taskid list, calls CANCELTC for all but last list item.
** For taskid list, calls TERMINTC for all but last list item.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

2.1.13.3 SIGNAL, SET, RESET Statements.

Operation	Type	Code
SIGNAL <event var>	latched or unlatched event	LA 0, event-var HCALL SIGNAL
SET <event var>	latched	LA 0, event-var HCALL SET
RESET <event var>	latched	LA 0, event-var HCALL RESET

2.1.13.4 UPDATE PRIORITY Statement. UPDATE PRIORITY requires an integral value to be specified in register R0 for the resultant priority. If the statement specifies via <taskid> a program or task, then the entry point of this program or task is contained in register R1.

Operation	Type	Code
UPDATE PRIORITY TO i	integer	L 0, i LH 0, i HCALL UPPRIO
UPDATE PRIORITY <taskid> TO i		L 0, i LH 0, i L 1, taskid HCALL UPPRIOT

2.1.13.5 SCHEDULE Statement.

```
SCHEDULE<task id>

  -                          <no code>
  AT T1                      LD    F0, =D'T1'
  IN T1                      LD    F0, =D'T1'
  ON E1                      LA    2, event expression containing E1

  PRIORITY P                 L     3, =F'P'

  -                          <no code>
  ,REPEAT                    <no code>
  ,REPEAT EVERY T2           LD    F2, =D'T2'
  ,REPEAT AFTER T2           LD    F2, =D'T2'

  -                          <no code>
  UNTIL T3                   LD    F4, =D'T3'
  WHILE E3                   LA    4, event expression containing E3
  UNTIL E3                   LA    4, event expression containing E3

                             L     1, task id
                             LA    0, flags*
                             HCALL SCHEDULE
```

* For description of flags, see Section 5.6.4.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

2.1.14 NAME Operations

2.1.14.1 NAME Comparisons. NAME comparisons may only be = or ¬=.

Operation	Code
NAME(X) <OP> NAME(Y) X, Y NAME variables	L R_X, X L R_Y, Y CR R_X, R_Y BC COND, not-true-label
NAME(X) <OP> NAME(Y) X=declared variable Y=NAME variable	LA R_X, X L R_Y, Y CR R_X, R_Y BC COND, not-true-label

2.1.14.2 NAME Assignments. 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.

```
NAME(Y) = NAME(X);
```

where X is declared variable

```
LA    R_X, X
ST    R_X, Y
```

```
NAME(Y) = NAME(X);
```

where X is NAME variable

```
L     R_X, X
ST    R_X, Y
```

2.1.15 %MACRO Operations

2.1.15.1 %SVC.

Operation	Code
%SVC(X)	LA R_X, X ST R_X, temp-storage-area MVI temp-storage-area, X'80' LA 1, temp-storage-area ST 13, max-temp-area+4 LA 13, max-temp-area L 15, =V(SVC) BALR 14, 15 L 13, 4(13) L 15, 0(13)

2.1.15.2 %NAMECOPY. This operation works in the same manner as NAME assignments except that the operands must be structures, but not necessarily having identical properties.

Operation	Code
%NAMECOPY(Y,X) X is actual variable	LA R_X, X ST R_X, Y

2.1.15.3 %COPY. The code is identical to the code for structure assignments (see Section 2.1.6.2) when the length to move is known (literal specification or omitted third parameter). For expression third parameter, the code is as follows:

Operation	Code
%COPY(X,Y,I);	LH 4, I AR 4, 4 LA 3, Y LA 2, X HCALL PCCOPY

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 involed 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.

2.2.2 Phase III Printed Data

For each invocation of Phase III, 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.

2.2.3 Stand-Alone Entries

The HAL/SDL Interface Control Document (ICD) fully describes the SDF format. Strictly speaking this ICD controls the interfaces between the HAL/S compiler and the Software Development Laboratory. Several stand-alone items are included for convenience but are explicitly controlled by this Specification. These items are listed below:

ICD Para. Reference	Item	Comment
2.2.1.2.1.1 (Sim. Table Directory Header)	Field No. 1 Bit No. 3, 7	FC_FLAG, NOTRACE_FLAG
	Field No. 33 (2 bytes)	Total Free Cell Space
2.2.1.2.1.2.2 (HAL Block List)	Field No. 17	ISN of 1st executable statement after DECLARES

Type I

The following HAL statements are implemented with the linkage instructions occurring before the normal statement code:

CANCEL	SCHEDULE
CLOSE	SEND ERROR
EXIT	SET/RESET
GO TO	SIGNAL
IF*	TERMINATE
Null	UPDATE PRIORITY
ON ERROR	WAIT/WAIT FOR
REPEAT	
RETURN	

For example, a GO TO ALPHA statement might expand into the following sequence of code:

```
BALR        14,11
DC          XL2'ISN_#'
BC          15,DISP_ALPHA(RX,RB)
```

* The linkage instructions for the associated THEN or ELSE statements are placed according to their own type, as if they were separate statements.

Type II

The following HAL statements are implemented with the linkage instructions occurring after the normal statement code:

Assignment	FILE
CALL	READ
DO (simple)	READALL
END	WRITE

For example, the generated code for a CALL BETA statement is as follows:

```
BALR        14,12
DC          A'BETA'
BALR        14, 11
DC          XL2'ISN_#'
```

Type III

The following HAL statements are emplemented with the linkage instructions occurring at special points within the statement code:

DO FOR	DO WHILE
DO FOR UNTIL or WHILE*	DO UNTIL
DO CASE	
DO END	

*Implementation of this is same as for DO FOR.

The in-line code sequences for the type III linkages are as follows:

DO FOR

HAL Statement

```
ISN_1            label:  DO FOR I = 1 TO 10 BY 2;
ISN_2→ISN_k              stmts;
ISN_k+1                  END;
```

Machine Code Generated

```
                 LA      R2,1
LBL_#1           ST      R2,I
                 BALR    14,11      } XMON link for DO FOR
                 DC      XL2'ISN-1' } Δt includes times for
                                      compare logic
                 C       R2, =F'10'
                 BH      LBL_#2
                 ---------------    } Machine language for
                 ---------------    } "stmts" with XMON links
                 ---------------    }
                 LA      R2,2
                 A       R2,I
                 B       LBL_#1
LBL_#2           BALR    14,11      } XMON link for END
                 DC      XL2'ISN_k+1'}
```

DO WHILE

HAL Statement

```
ISN_1           label:  DO WHILE (cond);
ISN_2→ISN_k             stmts;
ISN_k+1                 END;
```

Machine Code Generated

```
LBL_#1      BALR    14,11       } XMON link for DO
            DC      XL2'ISN_1'  } Includes execution time
                                  evaluation for "cond"

            ____________        } Machine language for
            ____________        } evaluation of "cond"
            ____________

            BNC     ST_#2       Branch on "cond" not met

            ____________        } Machine language for
            ____________        } "stmts" with XMON links
            ____________

            B       LBL_#1
LBL_#2      BALR    14,11       } XMON link for END
            DC      XL2'ISN_k+1'
```

DO UNTIL

HAL Statement

```
ISN_1           label:  DO UNTIL (cond);
ISN_2→ISN_k             stmts;
ISN_k+1                 END;
```

Machine Code Generated

```
            B       LBL_#1
LBL_#2
            ____________        } Machine instructions for
            ____________        } evaluation of "cond"
            ____________

            BC      LBL_#3      Branch on condition met
LBL_#1      BALR    14,11       XMON link for DO
            DC      XL2'ISN_1'

            ____________        } Machine instruction for
            ____________        } "stmts"
            ____________
```

```
                B       LBL_#2
LBL_#3          BALR    14,11          XMON link for END
                DC      XL2'ISN_k+1'
```

DO CASE ELSE

HAL Statement

```
ISN_1          label:  DO CASE J;
ISN_2                  ELSE S;
ISN_3          S1:,    ______________;
ISN_4          S2:     ______________;
ISN_5                  END;
```

Machine Code Generated

```
                ________________
                ________________   Evaluation of J
                ________________
                BALR    14,11      XMON link for evaluaton
                DC      XL2'ISN_1'  of J
                BR                 BR to S1, S2 or S
S:              ________________   Machine language for S
                ________________
                BALR    14,11      XMON link for S
                DC      XL2'ISN_2'
S1:             ________________   Machine language for S1
                ________________
                BALR    14,11      XMON link for S1
                DC      XL2'ISN_3'
                B       LBL#1
S2:             ________________   Machine language for S2
                ________________
                BALR    14,11      XMON link for S2
                DC      SL2'ISN_4'
                B       ST_#1
LBL#1           BALR    14,11      XMON link for END
                DC      XL2'ISN_5'
```

Depend on Statement Type

Note: Statements S, S1, and S2 are assumed to be type II statements.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

2.3.2 XMON Address Table

The XMON Address Table is a CSECT that contains the Statement Processor branch instruction mentioned previously plus a table of addresses that is required by the Statement Processor and the External Monitor.

CSECT name: #FCCCCCC, where CCCCCC is the first six characters of the compilation unit name, underscores removed and padded with blanks, if necessary.

Byte	Length	Contents
0	4	Branch Instruction — transfer to Statement Processor. See below for further information
4	4	Address of 1st byte of DECLARE data CSECT.
8	4	Address of Statement Processor Data Table CSECT @TCCCCCC).
12	4	Same as above minus 4*F (F=statement number of 1st executable statement - either the PROGRAM or PROCEDURE statement).
16	4	Entry address of compilation unit ($0CCCCCC or #CCCCCCC).
20	2	Statement # of 1st executable statement. (DC H'F')
22	2	Statement # of last executable statement. (DC H'L'
24	16	Reserved for external or HAL/System 360 use (zeros).
40	20	Compiler's data addressing register conten (R6-R10).

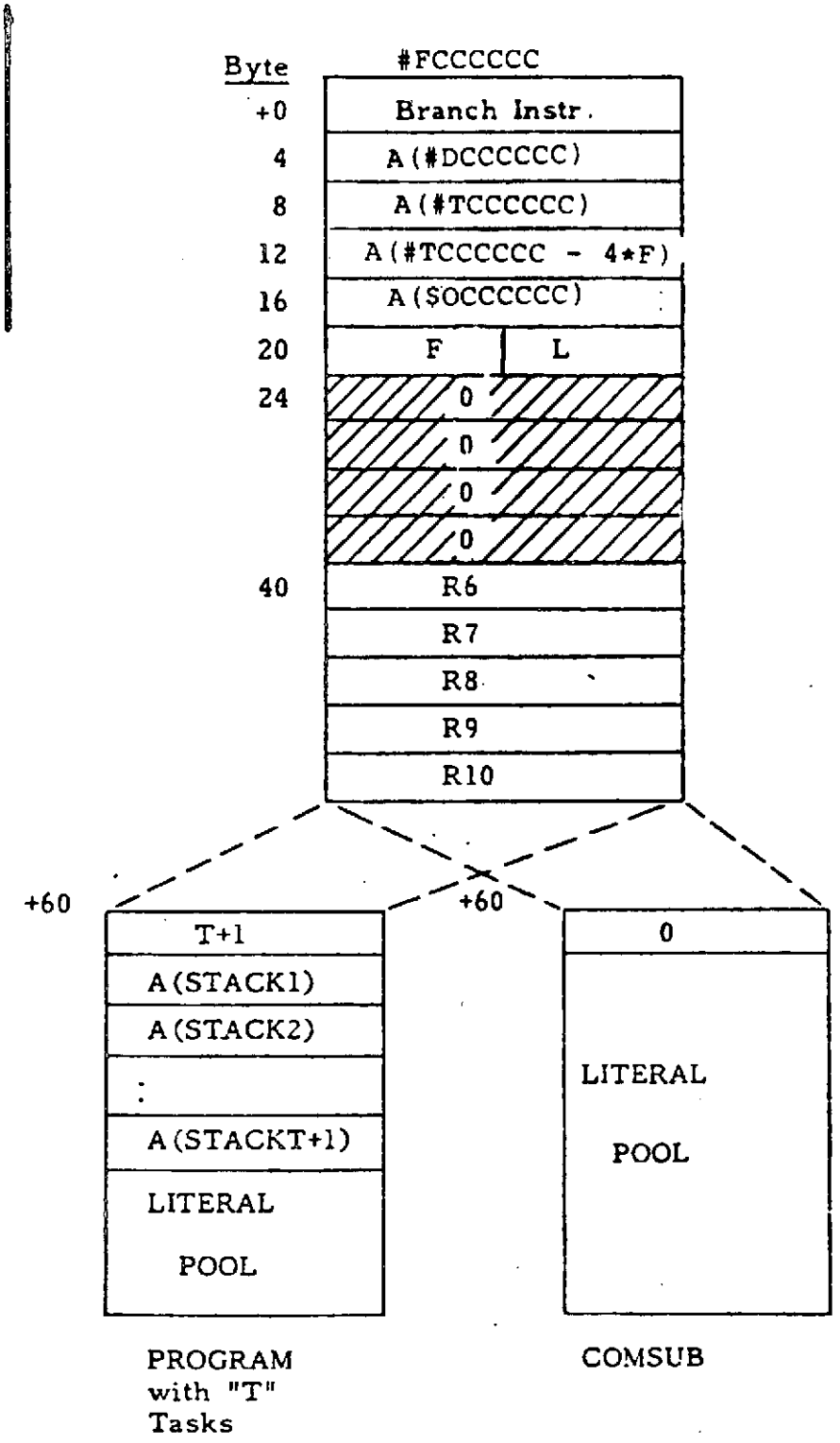


Figure 2.3-1: XMON Address Table CSECT

For programs with "T" tasks:

60	4	Total number of stacks (T+1).
64	4	Address of program stack (@0CCCCCC).
68	4	Address of task #1 stack (@1CCCCCC).
:		
64 + 4T	4	Address of task #T stack (@TCCCCCC).
68 + 4T	-	Compiler's literal area begins here.

For COMSUBs:

60	4	4 bytes of zeros
64	-	Compiler's literal area begins here.

The first entry in the CSECT is the Statement Processor branch instruction. This is the instruction pointed to by register 11 in the linkage instructions described in Section 2.2.1.3.1. It is simply an unconditional branch to the Statement Processor contained in the HALSYS object module. The form of the instruction is as follows:

```
BC    15,DISP(0,12)
```

Register 12 contains the address (loaded by HALSTART at initialization) of HALSYS and DISP is the displacement of the Statement Processor into HALSYS. This instruction does not destroy the contents of register 14, which still points to the compiler's internal statement number for the particular statement being executed.

A four-byte pointer to the location of the XMON Address Table CSECT and, hence, to the branch instruction, is contained at a displacement of four bytes from the start of the CSECT which contains the executable code for the unit of compilation.

2.3.3 Statement Processor Data Table

The Statement Processor Data Table for a unit of compilation is a CSECT that contains the basic information necessary for the Statement Processor to process each statement in that unit of compilation. There is an entry in the table for each compiled HAL source statement and, hence, for each internal compiler statement number, with one major exception. There are no entries in the table for any COMPOOL declaration statements which precede the primary unit of compilation. Since the first portion of almost every unit of compiler consists of these non-executable statements, a significant space-saving is achieved in this manner. The remainder of the table is dense with every statement having one entry.

Each entry in the table contains a two-byte statement execution time (in machine cycles), a statement action flag bit, and a two-byte utility field for miscellaneous use in any manner desired by the external environment. The statement action flag bit occupies the leftmost bit position of the utility field. The format of an entry is shown in the figure below.

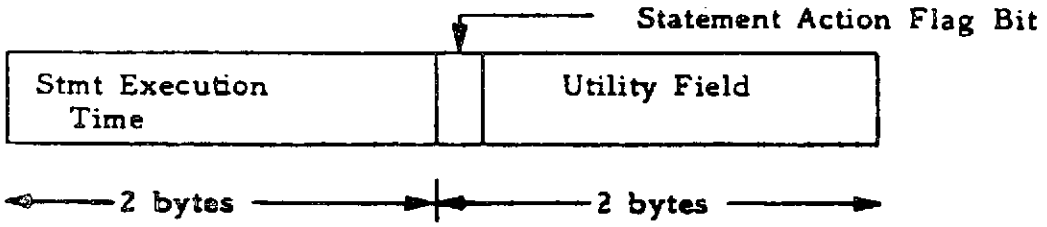


Figure 2.3-2: Format of Statement Processor Data Table Entry

The name of the Statement Processor Data Table CSECT is of the form #TCCCCCC, where CCCCCC is the first six characters of the compilation unit name with any underscores removed and the name padded with blanks, if necessary.

A pointer to the start of this table is provided in the XMON Address Table described in Section 2.3.2.

2.4 Phase 1.5 - The Optimizer

2.4.1 General Description

The HAL/S Optimizer takes HALMAT produced by Phase I and performs the following functions:

- Common subexpressions (CSE's) are recognized.
- Additional constant folding is carried out.
- Unneeded divisions are replaced by multiplications.
- Superfluous matrix transpose operations are eliminated.
- Inline code is generated to replace certain VECTOR/MATRIX LIBRARY CALLS.
- LOOP invariant HALMAT is pulled outside of DO and Array loops.
- Adjacent and nested array loops are combined when possible.

Altered HALMAT is then passed to Phase II for object code generation.

2.4.2 Design Comments

The most important design consideration is that the Optimizer does nothing to most HAL/S statements! Thus, the sooner this is recognized, the less time wasted on a statement and the more efficient is the Optimizer. More concretely, the following features are of note:

1. The CSE_TAB doubly linked list drastically reduces the number of Nodes searched for CSE's. This might be compared with FORTRAN H where the previous ten statements are searched for CSE's, even though they may contain no common variable with the present statement.
2. If a Node does not have enough eligible operands for a CSE, no search is made (SEARCHABLE=FALSE).
3. The Optimizer is quite conservative. For example, all user procedure and function calls cause ZAP_TABLES to be invoked.

2.4.3 Optimizations Attempted

This section describes those optimizations presently implemented in the HAL/S OPTIMIZER and corresponding Phase II, and gives appropriate user information.

2.4.3.1 Common Subexpression Eliminations.

a. "Cummutative" Operations

For bits: &, |

For scalars: +, -, <>, ÷

For integers: +, -, <>

For vectors and matrices: +, -

Example 1:

```
F = A - D + B - C;
G = D - C - B + A;
```

becomes*:

```
CSE1 = A - C;
CSE2 = B - D;
   F = CSE1 + CSE2;
   G = CSE1 - CSE2;
```

Example 2:

```
F = (A/B) (C/D);
G = C(B/D) A;
```

becomes:

```
CSE1 = C/D;
CSE2 = A/B;
   F = CSE1 CSE2;
   G = CSE1/CSE2;
```

---

* Often the CSE's are merely retained in registers with no temporaries created.

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

Example 3:

```
F = A + B + (C D) + E + (B C A);

G = D + (D C) + E + A + (A B);
```

becomes:

```
CSE1 = A + E + (C D);
CSE2 = (A B);
   F = CSE1 + B + (CSE2 C);
   G = CSE1 + D + CSE2;
```

b. Noncommutative Operations

1. For bits: ||, ¬.

   Built-in functions: XOR.

2. For scalars and integers: **, negation, conversion to integer or scalar from integer or scalar.

   Built-in functions: ABS, CEILING, FLOOR, ODD, ROUND, SIGN, SIGNUM, TRUNCATE, ARCCOS, ARCCOSH, ARCSIN, ARCSINH, ARCTAN, ARCTANH, COS, COSH, EXP, LOG, SIN, SINH, SQRT, TAN, TANH, DIV, MOD, SHL, SHR, INDEX, LENGTH, MIDVAL, ARCTAN2, REMAINDER.

3. For vectors and matrices*: negation, m v, v m, v*v, v x, x v, v/x, m m, v v, m x, x m, m/x, m**i.

   Built-in functions: ABVAL, DET, INVERSE, TRACE, TRANSPOSE, UNIT.

---

* i = non-negative integer literal,
x = scalar or integer,
m = matrix, and
v = vector.

Example 4:

```
X_NEW = X COS(THETA) + Y SIN(THETA);
Y_NEW = Y COS(THETA) - X SIN(THETA);
```

becomes:

```
CSE1 = COS(THETA);
CSE2 = SIN(THETA);
X_NEW = X CSE1 + Y CSE2;
Y_NEW = Y CSE1 - X CSE2;
```

Example 5:

```
R1 = (-B + SORT(B**2 - 4 A C))/2A;
R2 = (-B - SQRT(B**2 - 4 A C))/2A;
```

becomes:

```
CSE1 = -B;
CSE2 = SQRT(B**2 - 4 A C);
CSE3 = 2 A;
  R1 = (CSE1 + CSE2)/CSE3;
  R2 = (CSE1 - CSE2)/CSE3;
```

Subscript Common Expressions (CSE)

Subscripting of arrayed data requires the calculation of a displacement. SCE's are recognized in these computations with these exceptions:

Character and Bit Types

No SCE's are recognized which only involve terminal subscripts. (Character type is not handled at all.)

Structure Subscripts

Only the entire TSUB operator is eligible for CSE's; no partial computation.

Example 6:

Suppose A is a three-dimensional array of matrices. Then to reference:

$$A_{3,J,4:\ K+1,\ L}$$

code must be generated computing:

(J constant_1) + (K constant_2) + L.

Consider:

$$F = A_{3,J,4:\ K+1,\ L} + A_{3,J,5:\ K,L}$$

The subscript computation will be done only once.

Subscript common expressions will also be recognized between:

$$A_{3,J,4:\ K+L,\ M}$$

and:

	Term	SCE
1.	$A_{1,J+2,1:\ 1,M}$	(J constant_1) + M
2.	F = K+L	K+L
3.	F = (K+L) constant 2	(K+L) constant_2
4.	$B_{K+L,2}$	K+L or (K+L) constant_2 if B is of the right dimension

Only machine independent information will appear as SCE's. Phase 2 must still perform shifting for data type and alignment.

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

2.4.3.2 Matrix Transpose Eliminations. M^T V is changed to V M and V M^T is changed to M V, saving a transpose operation.

Example 7:

```
M = M^T((M1 + M2)^T V);
```

becomes:

```
M = (V (M1 + M2)) M;
```

2.4.3.3 Constant Folding. Some constant folding not done by Phase I involving integer and scalar +, -, <>, and ÷ is performed.

Example 8:

```
F = (2A)/(4 B C);          (all scalars)
```

becomes:

```
F = (.5A)/(B C);
```

CSE's involving folding constants are found.

2.4.3.4 Division Eliminations. Terms are rearranged to eliminate unneeded divisions.

Example 9:

```
F = (A/B) (C/D) (E/F);
```

becomes:

```
F = (A C E)/(B D F);
```

2.4.3.5 Inline Vector/Matrix Computations. The following vector/matrix operations were originally handled by calls to library routines, each containing an iterative loop. Optimization generates HALMAT to perform these operations inline.

Vector Assign
Vector Negate
Vector Add
Vector Subtract
Vector-Scalar Product
Vector-Scalar Divide
Matrix Assign
Matrix Negation

Matrix Add
Matrix Subtract
Matrix-Scalar Product
Matrix-Scalar Divide

Example 10:

```
V1 = V2 + V3 + V4;        /* ALL VECTORS */
```

This statement, which would require three calls to library routines, is accomplished in a single inline loop. Without optimization, the statement would be executed as follows:

```
VTEMP = V2 + V3;
V1 = VTEMP + V4;
```

2.4.3.6 Loop Simplification. Expressions within iterative loops, which contain only variables that are loop invariant; i.e., do not have their values changed within the loop, are evaluated once before the loop is entered.

Example 11:

IF A and THETA are loop invariant then:

```
DO FOR I = 1 TO Y;
        F = A + B + COS(THETA);
        . . .
END;
```

would become:

```
CSE = A + COS(THETA)
DO FOR I = 1 TO Y;
        F = B + CSE;
        . . .
END;
```

saving needless recomputations of A + COS(THETA).

Exceptions:

It is necessary to guard against run time errors being generated by this process when loops are executed 0 times or when statements such as:

```
IF B ≠ 0 THEN A = 1/B;
```

appear within the loop.

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

To prevent most (but not quite all) such errors, no computation will be pulled involving a variable referenced in a conditional test.

An assign to a name variable, a call to a user function, or a real time statement would prevent any variable in the loop from being pulled. No name variables can be pulled.

Example 12:

Subscript computations are prime candidates for this optimization, especially since the programmer cannot do this. Thus, if J is loop invariant, then for:

$$A_{J,3}$$

the product:

J constant_1

will be pulled outside the loop.

Example 13:

Array loops will be handled like other loops for example:

```
[A] = [B] + C + [D] + E;
```

will become:

```
TEMP = C + E;

[A] = [B] + [D] + TEMP;
```

Example 14:

Array Subscript Loops

For array subscripts only, presense of "AT", "TO", or "*" subscripts produce loops. Loop invariant computations can be beneficially pulled before loops. Thus:

```
\T J,K: L,4
```

will have J const_1 + K const_2 + L const_3 removed from the array loop.

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661 1840

Example 15:

Common Subexpressions

CSE's will continue to be found within loops. If the invariant expression pulled from a loop is a CSE outside the loop, it will be combined. E.g.

```
F = A + B + D;
DO FOR I = 1 TO Y;
        G = A + B + C;
        . . .
END;
```

If A and B are loop invariant, we will get:

```
CSE = A + B;
  F = CSE + D;
  DO FOR I = 1 TO Y;
          G = CSE + C;
          . . .
  END;
```

2.4.3.7 Loop Combining. After all other processing is complete, a pass is made to combine adjacent vector/matrix or array loops of the same dimension.

Two loops can be combined if all of the following hold:

1. Neither contain function or procedure calls.
2. Neither contain assignments into name variables, assign parameters, or DSUBS.
3. If the loops are in different statements, neither statement contains a TSUB.

Loops are not combined if:

1. Both an assignment and a DSUB reference occur anywhere within either of the loops.
2. The first loop contains an assignment into a vector/matrix type variable, and that same variable appears in a non-assignment context in the second loop, and the second loop is arrayed or invokes a vector/matrix routine that cannot be done inline.

Example 16:

Given vectors V1, V2, and V3, AV ARRAY(3) VECTOR, and AM ARRAY(3) MATRIX, the following loop pairs would not be combined:

```
V1 = V2 + V3;          V1 = V2 + V3;
AV = AM V1;            AV = AV + V1;
```

De-Nesting Array Loops

Consider the arrayed statement:

```
F = A + B;      (F, A, B ARRAY(3,5))
```

Two nested loops are generated.

If no DSUB's, TSUB's, or terminals of structures with copies are present, then one loop replaces the nested loops. This also saves on multiplications.

2.4.4 Scope of Optimization

Common subexpressions are recognized over approximately basic blocks of code. No CSE's are recognized across:

- labels
- user procedure or function calls
- assignments into name variables
- HALMAT blocks
- inline functions
- GO TO's
- END's of DO FOR's, DO WHILE's, DO UNTIL's
- END's for simple DO END if there is a corresonding EXIT
- Major or Minor Structure Assignments
- READ, READALL, and FILE I/O instructions
- program organization operators (e.g. PROCEDURE, CLOSE)
- WAIT statements
- ERROR statements
- IF statement conditionals containing more than one boolean comparison
- ends of the true parts in IF THEN's or IF THEN ELSE's
- end of IF THEN ELSE's

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

The presence of any of the following causes the entire statement to be skipped.

user procedure or function calls
inline functions
I/O instructions
shaping functions
character operations
bit or character conversion to integer or scalar
real time statements

3.0 SYSTEM CONCEPTS AND INTERFACES

3.1 HAL Object Module Layout

Each successful HAL compilation produces a number of named control sections (CSECTS).

The CSECT name corresponding to the executable code is derived from the label of the PROGRAM, PROCEDURE, FUNCTION or COMPOOL being compiled, according to the following rules:

1) Eliminate all underscores from label.

2) Pad or truncate to 6 characters where necessary. All CSECT names are based on this 6 character name.

3) Prefix 6 character name with "$0" if PROGRAM, "#C" if PROCEDURE or FUNCTION, or "#P" if COMPOOL.

All TASKS also result in a separate control section.

The name of the CSECT corresponding to a TASK is derived from the PROGRAM in which it is defined, not from the label on the TASK declaration itself.

The names are similar to the PROGRAM name, except that the first task name is prefixed with "$1", the second task with "$2", etc. The letters "A-Z" follow the digit "9", thus allowing up to 35 TASKs in a compilation unit.

Other CSECTs are:

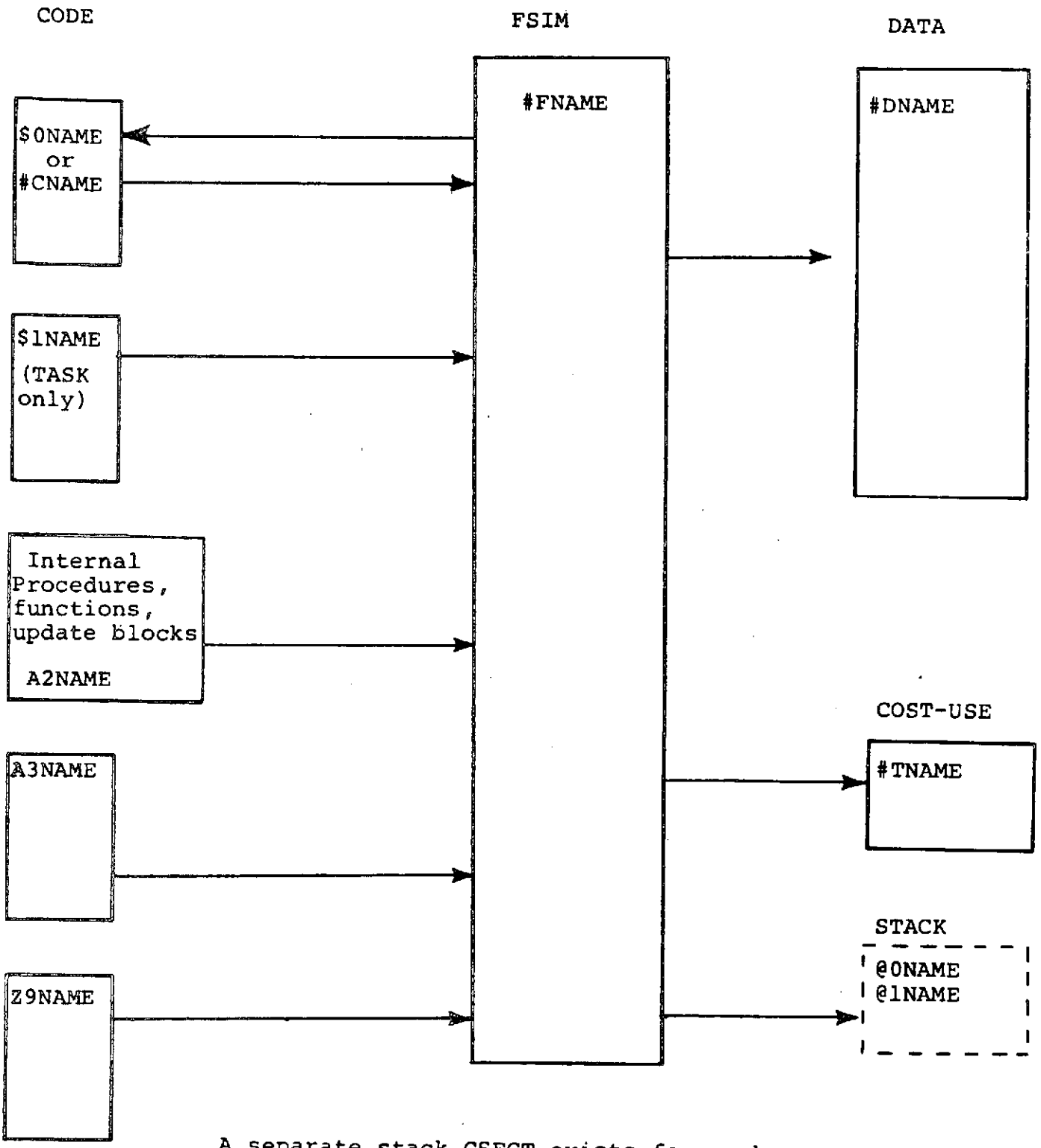
1) Data are for HAL variables. Name derived by prefixing the 6 character name with "#D".

2) FSIM containing FSIM address tables and literals used in HAL programs. Name derived by prefixing 6 character name by "#F".

3) Cost-use array. Cost and use arrays for statement timing, hot bits, and SDL/SLS/Stand-Alone specific information. Name derived by prefixing 6 character name by "#T". This is present only if TRACE was specified during compilation.

4) Internally defined PROCEDURES, FUNCTIONS, UPDATE Blocks, and Inline FUNCTIONS. Name derived by prefixing 6 character name as follows: "A2" - "A9", "B0" - "B9"., ...,"Z0" - "Z9". The name is derived from an internally generated block number. Note that blocks are named with a different convention.

A stack CSECT for each PROGRAM and TASK is generated by HALLINK. The name of the stack CSECT is obtained by replacing the $ in the code CSECT name by an at sign (@).



A separate stack CSECT exists for each PROGRAM and TASK. The stack CSECTS are generated by the HALLINK program.

Figure 3-1

A typical example:

```
                ( PARM.HAL = 'TRACE'    specified)
ABC: PROGRAM;
     X: TASK; ...CLOSE;
     Y: PROCEDURE; ...CLOSE;
     Z: UPDATE ...CLOSE;
CLOSE ABC;
```

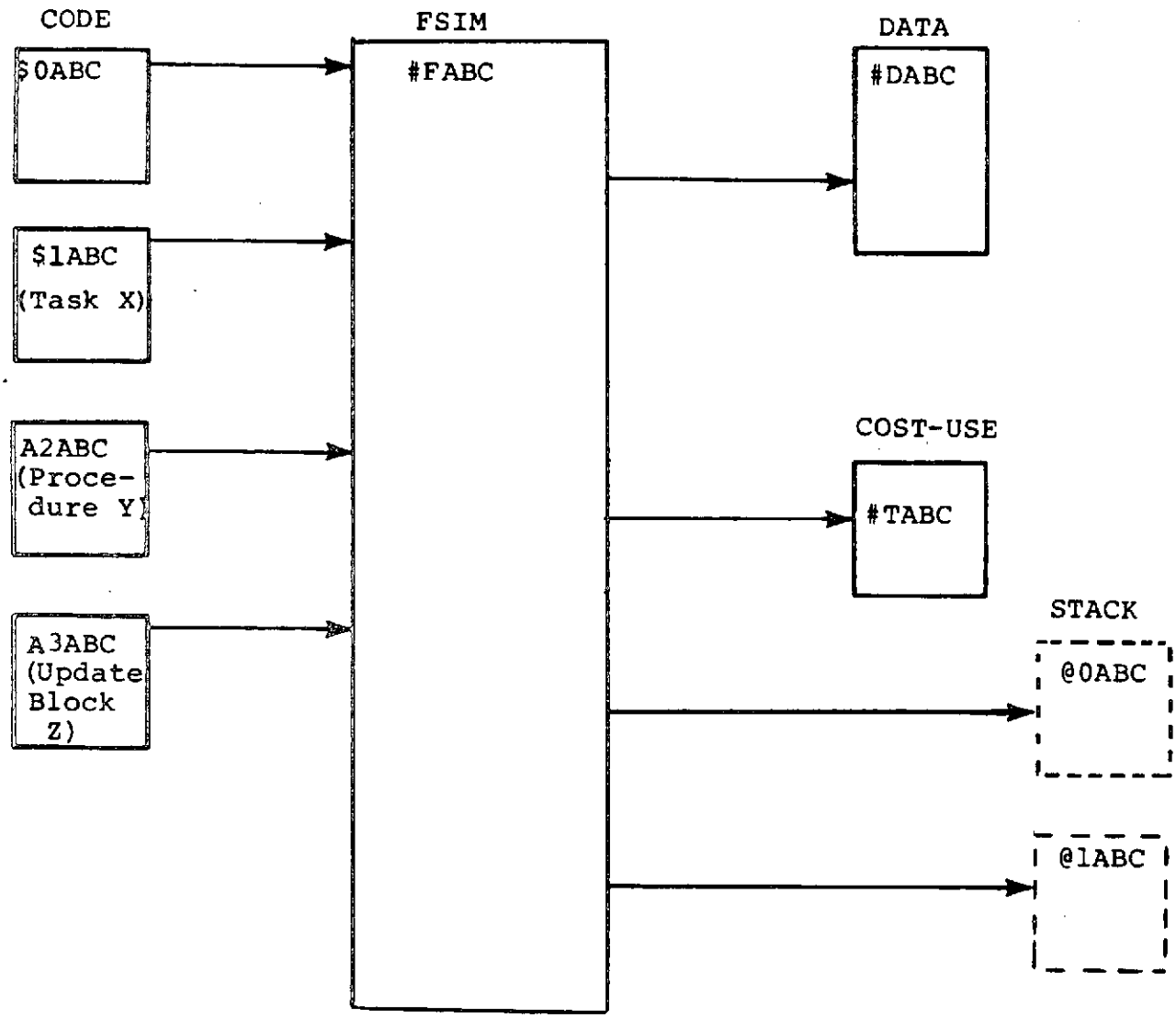


Figure 3-2

3.2.2 Template Checking

Template checking is a technique used to ensure that definitions of PROGRAMS, COMPOOLS, and COMSUBS (external procedures and functions) are consistent throughout a load module at run-time.

Each compilation unit has a template and a version number associated with it. The version number is contained within the template. Its range is 1 to 255 inclusive.

The template consists of standardized card images containing information derived from the compilation unit. It is laid out as follows:

1) Block header of compilation unit modified by the keyword EXTERNAL.

2) All declarations, if compilation unit is COMPOOL; otherwise, if the compilation unit has any parameters, all declarations, REPLACE statements, and structure templates up to, and including, the last declaration pertaining to those parameters.

3) CLOSE;

4) D VERSION XX

   where XX is one byte version number.

The first time a compilation unit is compiled, a template is generated with the version number 1. If each subsequent compilation differs from the existing one, a new template is generated. Upon each regeneration, the version number is incremented by one, unless it is 255, in which case it is reset to 1.

Template checking is provided via the "sym" cards of the linkage editor.

If a compilation unit references other external compilation units, the compiler determines the version numbers of their templates (which must be included in the compilation), and passes them to HALLINK. The version number of the compilation unit being compiled is also passed to HALLINK.

When a load module is produced by HALLINK, a check is made for each external compilation unit referred to by the compilation unit being compiled. An error is signalled if the version number of the template of the external compilation unit is not the same as the version number of the object module of that external compilation unit.

3.2.3 HALMAP

The control section named HALMAP is inserted into all load modules by HALLINK. Its length varies according to the number of compilation units included in the load module. For each compilation unit, the following information is included:

1) Type of compilation unit; i.e. PROGRAM, COMSUB, or COMPOOL;

2) Address of first byte;

3) Name of member in SDF files with pertinent information.

Length of HALMAP is 4+12n bytes, n = number of compilation units.

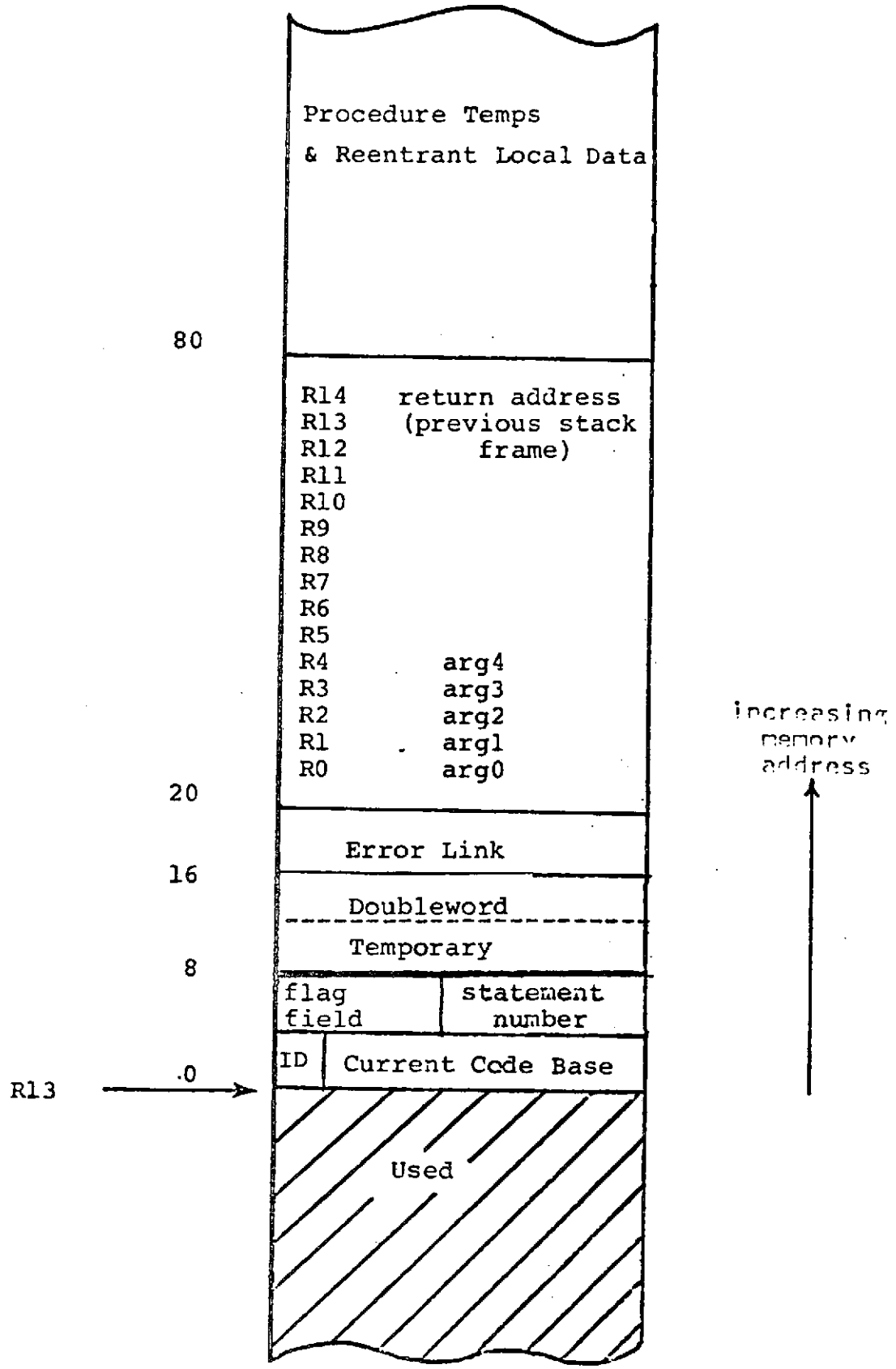
Halmap Layout

Location	Length (bytes)	Description
0	2	# of compilation units (n).
2	2	# of PROGRAMS and TASKS.
4	12n	One 12 byte entry for each compilation set up as outlined below.
Compilation Entry		
0	1	Type: X'00' = COMPOOL X'01' = PROGRAM X'03' = COMSUB
1	3	Address of first byte of control section.
4	8	SDF Member Name.

NTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Figure 3-3

STACK LAYOUT



NOTE: High order bit of flag field is "1" if exclusive procedure; The other 15 bits represent the lock groups for an update block.

Figure 3-4

PROCEDURE AND FUNCTION CALLS

```
Load        R0,     Argument 0
Load        R1,     Argument 1
.
Load        R4,     Argument 4
Load        F0,     Floating Arg. 0

BALR        R14,    R12     GO TO PROCEDURE CALLER

DC          AL1(ID), AL3(Entry-Adcon)
```

PROCEDURE AND FUNCTION EXITS

```
B           4(R12)          GO TO PROCEDURE EXITER
```

Notes:

a) The ID-Adcon is a four-byte field aligned on the halfword immediately following the BALR instruction.

b) The one-byte ID is the unique block ID of the called procedure or function, as follows:

ID	Block Type
N	Programs
0	COMSUBS, and library routines
$\geq$ N	Nested procedures

where N is a compiler-assigned internal block number.

c) The three-byte Adcon is the entry point address of the called procedure or function.

the compiler will attempt to generate in-line code sequences, including as many operations within a single loop as possible. In many cases, the stores into temp-area's, 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

```
DECLARE VECTOR,V,W,X,Y,Z;

 V = V+(W+X)*Y-Z;
```

Code

```
        LA      R_L, 1
        LR      R_I, R_L
L1      SLA     R_I, 2
        LE      FR, W(R_I)
        AE      FR, W(R_I)
        STE     FR, templ(R_I)
        LA      R_L, 1(0, R_L)
        CH      R_L, =H'3'
        BC      12, L1
        LA      4, Y
        LA      3, templ
        LA      2, temp2
        HCALL   VX6S3
        LA      R_L, 1
        LR      R_I, R_L
L2      SLA     R_I, 2
        LE      FR, V(R_I)
        AE      FR, temp2(R_I)
        SE      FR, V(R_I)
        LA      R_L, 1(0,R_L)
        CH      R_L, =H'3'
        BC      12, L2
```

4.0 USER INTERFACE

User interfaces are those which are directly related to the actions which must be taken by a user to communicate with the HAL/S-360 compiler system. User interfaces do not include many actions taken automatically by the compiler as a result of System Interfaces (see Section 3). The interfaces presented here are primarily related to control card and source card input and to printed outputs.

The HAL/S-360 compiler system is designed to run in both a Stand-Alone mode and within the Software Development Laboratory (SDL). This presentation of user interfaces describes operation in the Stand-Alone mode. Special sections are included, as needed, to describe areas in which SDL operation differs from that in Stand-Alone.

In order to avoid duplication of some information already contained in other sections, some references to such sections are made where appropriate.

4.1 The Compile Step

4.1.1 Job Control Language

The JCL necessary to execute the HAL/S-360 compiler is described below. A list of typical JCL statements to which the comments apply is shown in Figure 4.1.

- The EXEC card invoking the HAL/S-360 compiler must specify program name MONITOR. MONITOR handles all compiler/OS interfaces and also performs the actual loading and overlaying of the phases of the compiler. The compiler requires a 350K region. A larger region may be specified. The compiler will always use all the memory it is given. A larger region will generally result in smaller compilation times. A default time limit of 1 minute is shown. This is sufficient for most average size HAL/S programs (approx. 300 HAL/S statements).

  The PARM field contains the compile-time options as described in Section 4.1.2.

- The STEPLIB DD card specifies the location of the load module library which contains the module MONITOR needed to run the compiler. This card may define any direct access library which contains the proper module or may be omitted at installations where the module has been made part of the system library (SYS1.LINKLIB).

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

Figure 4.1

```
//HAL       EXEC PGM=MONITOR,REGION=350K,TIME=1,PARM=<compile time options>'
//STEPLIB   DD DISP=SHR,DSN=HALS360.MONITOR
//PROGRAM   DD DISP=SHR,DSN=HALS360.COMPILER
//SYSPRINT  DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=3458)
//LISTING2  DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=3458)
//OUTPUT3   DD UNIT=SYSDA,DISP=(MOD,PASS),SPACE=(CYL,(1,1)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=400),
//             DSN=&&HALOBJ
//OUTPUT5   DD DISP=(MOD,PASS),DSN=&&HALSDF,SPACE=(TRK,(2,2,1)),
//             DCB=(RECFM=FB,LRECL=1680,BLKSIZE=1680),UNIT=SYSDA
//ERROR     DD DISP=SHR,DSN=HALS360.ERRORLIB
//FILE1     DD UNIT=SYSDA,SPACE=(CYL,(3))
//FILE2     DD UNIT=SYSDA,SPACE=(CYL,(3))
//FILE3     DD UNIT=SYSDA,SPACE=(CYL,(3))
//FILE4     DD UNIT=SYSDA,SPACE=(CYL,(3))
//FILE5     DD UNIT=SYSDA,SPACE=(CYL,(3))
//FILE6     DD UNIT=SYSDA,SPACE=(CYL,(3))
//INCLUDE   DD DISP=OLD,DSN=INCLIB
//OUTPUT6   DD DISP=OLD,DSN=TEMPLATE.LIB
//ACCESS    DD DISP=OLD,DSN=ACCESS
//SYSIN     DD * or <dsn pointers>
//OUTPUT4   DD SYSOUT=B,DCB=(RECFM=FB,LRECL=80,BLKSIZE=400)
```

The following JCL defines data which are needed only if certain of the compiler's features are used.

- The INCLUDE DD card identifies the dataset(s) which will be searched to resolve requests made on INCLUDE compiler directives (see Section 3.8 above) and to perform block template verification (see Section 3.2 above). Several DD cards may be concatenated to define the INCLUDE JCL. Each dataset referenced must be of partitioned organization and must have RECFM=F or FB.

- The OUTPUT6 DD card defines the dataset onto which block templates will be written (see Section 3.2.2 above). The dataset specified must have partitioned organization and must specify fixed length records.

- The ACCESS DD card specifies the partitioned dataset from which the compiler will obtain ACCESS control information as described in Section 3.1.2 above. The DSORG must be PO and the RECFM must be F or FB with LRECL = 80.

- The SYSIN DD card specifies the location of the primary source input to the compiler. This file must have sequential organization and must have the following DCB attributes for Stand-Alone operation:

  RECFM = F or FB

  LRECL = 80

  When operating in an SDL environment, the SYSIN dataset must still have RECFM=F or FB but may have

  $80 \leq LRECL \leq 132$

- The OUTPUT4 DD card specifies the destination of an object deck output. The object deck will be identical to that produced by the OUTPUT3 DD card. The OUTPUT4 DD card is generally used to obtain punched card output of an object deck. Its DCB requirements are identical to OUTPUT3. This DD card is used if the duplicate object deck is requested via the DECK compiler option.

4.1.2 Inputs

User-defined inputs to the compiler step consist of both compiler options and HAL/S language source statements.

4.1.2.1 Compiler Options. The following is a list of options which may be coded in the PARM field of the EXEC card which invokes the HAL/S-360 compiler. In all cases, options are separated in the PARM field by commas. If an option is referenced more than once in a PARM field, the last reference (scanning left to right) will be used to determine the option's setting.

There are two general classes of options recognized by the compiler: Type 1 options having a binary value of "on" or "off", and Type 2 options having a numeric or string value.

Type 1 Options

Type 1 options are controlled by keywords in the PARM field. The appearance of the keyword indicates that the option is to be "on" during the compilation unless the keyword is preceeded by the characters "NO" in which case the option is "off". Some Type 1 options have alternate, shorter spellings which may be used interchangably with the standard keywords.

When a Type 1 option has an altnerate form, the negative or "off" value (equivalent to adding 'NO' to the standard keyword) is specified by preceeding the alternate form with the charanter 'N'. The 'NO' and 'N' notations may only be used with the standard and alternate forms respectively. For example, the LIST option has the alternate form L. If the negative is to be specified, it may be done as NOLIST or NL; NLIST or NOL will not be recognized.

The following Type 1 options are recognized. The default settings shown are used in the absence of overriding PARM field specifications.

Keyword	Alternate	Default	Function
LISTING2	L2	off	Causes unformatted source listing to be generated.
DUMP	DP	off	Requests the compiler to produce a memory dump if certain internal compiler errors occur.
LIST	L	off	Produces an assembly listing from Phase II of the compiler.
TRACE	TR	on	Causes the generation of a link to the HSS end-of-statement routine in the object module. Enables Real Time execution and debugging.
DECK	D	off	Controls production of an additional object deck on the OUTPUT4 DD card.
TABLST	TL	off	Causes Phase III of the compiler to produce formatted dump of the simulation data file (SDF).
SRN	none	off	Causes the compiler to omit the last eight columns or characters from the source scanning. These columns are then used to print information on the listing.
TABLES	TBL	on	Controls generation of Simulation Data Files.
ADDRS	A	off	Indicates the presence of statement address information in the Simulation Data Files.
TABDMP	TBD	off	Causes Phase III of the compiler to produce a hexadecimal dump of the simulation data file.

Keyword	Alternate	Default	Function
SDL	none	off	Informs the compiler that it is operating within the SDL. ACTIONS specific to SDL operations are keyed to this option such as inclusion of SRN, Change Authorization Field and Source record revision indicator on primary listing.
FCDATA	FD	off	Causes HAL/S-360 data to be allocated using a halfword as the basic memory unit. This causes data area allocation which maps directly into HAL/S-FC data allocation.
SREF	SR	off	Causes special processing of user-defined symbols which appear within an EXTERNAL COMPOOL template which is included in another compilation. Any items in such a COMPOOL which are not referenced by the primary compilation unit are not printed in the symbol table listing.
QUASI	Q	off	Causes alternate generation of double precision multiply and divide instructions. The pseudo-instructions cause program exceptions which can be trapped by the HAL/S runtime system as a means of simulating alternate floating point precision. See Section 6.4.

Type 2 Options

Type 2 options have "values" which may be altered by the user. The values are specified by including the pseudo-assignment statement:

...,<type 2 opt>=<value>,...

in the PARM field where <type 2 opt> is one of the legal type 2 options, and <value> is the value to be used during compiler execution. The form of <value> is determined by the specific options. Some Type 2 options have alternate, shorter spellings which may be used interchangably with the standard forms.

The following Type 2 options are recognized. The default values shown are used in the absence of overriding PARM field specifications.

Standard	Alternate	Default	Function
PAGES=	P=	250	Sets the maximum page number to be allowed in generation of the primary compilation listing.
LINECT=	LC=	59	Sets the maximum number of lines which will be printed on any one page of either the primary or secondary source listing.
TITLE=	T=	null	Specifies 1 to 60 characters used by the compiler when printing header information at the top of each page of the listing.
SYMBOLS=	SYM=	200	Specifies the size of the compiler's symbol table.
MACROSIZE=	MS=	500	Specifies the maximum number of characters allowed in text of macro definitions.
LITSTRINGS=	LITS=	2000	Specifies the maximum total number of characters allowed in character literals in a table.

Standard	Alternate	Default	Function
COMPUNIT=	CU=	0	Specifies a compilation unit number to identify the unit of compilation. The number is made available in the SDF and in the Block Data Areas for code blocks in a HAL/S-FC compilation.
XREFSIZE=	XS=	2000	Specifies the number of cross reference table entries allocated by the compiler. Each entry uses 4 bytes of storage.
CARDTYPE=	CT=	null	Specifies pairs of characters which define a mapping of arbitrary input record types (column 1 of the record) into the standard types (E,M,S,C, D, and blank). E.g. CT=XCYM would cause any 'X' records to be compiled as comments and any 'Y' records to be compiled as 'M' records.
LABELSIZE=	LBLS=	1200	Specifies the maximum number of internal label points which will be maintained by the code generator.
BLOCKSUM=	BS=	400	Specifies the maximum number of entries in the table used to accumulate data for the "BLOCK SUMMARY" printouts at the ends of blocks.

A listing of the object code produced by the compiler may be requested. The result of such a request is a pseudo-assembler listing which identifies the code produced for the compilation in both a pure hexadecimal and in a mnemonic op-code format. References to HAL/S variables are indicated by appropriate comments on each instruction. Additionally, the time penalty for each HAL/S statement is indicated. This value is an approximate Shuttle GPC execution time and is used in the FSIM real time executive simulation to advance the pseudo timer.

When operating in the SDL, additional information is provided on the primary source listing. The Record Sequence Number, Record Revision Indicator, and Change Authorization fields (see Section 4.1.2.2) are printed on the primary source listing next to the statements to which they apply.

Under control of the TABLST and TABDMP compiler options, a formatted and/or hexadecimal dump of the contents of an SDF may be requested.

4.1.3.2 Other Compiler Outputs. In addition to the listings described in Section 4.1.3.1, other non-printed outputs are generated by the compiler. These outputs are generally considered systems interfaces and are, therefore, dealt with in Section 3. Their general nature is mentioned here only for completeness.

- object decks
- block templates
- simulation data tables

4.2 The Link Step

The HAL/S-360 compiler system employs a mechanism for the handling of temporary work areas at execution time which requires special processing at the time all pieces of a run are linked together. This processing is achieved by substituting a HAL/S-360 compiler system routine for the standard OS/360 link editor in the LKED step in the program generation process. This program is known as HALLINK.

Temporary work areas and general registers save areas used by a running HAL/S program are obtained from an area called the STACK. The STACK is really a CSECT of sufficient size to allow all routines with temporary data requirements to obtain memory from the STACK CSECT. One STACK CSECT exists for each PROGRAM or TASK in a program complex. It is not until link-edit time that all of the individual routines' requirements for temporary space are known. The HALLINK program determines the requirements and creates the STACK for each PROGRAM and/or TASK. In performing this function, HALLINK makes use of the standard OS/360 linkage editor. The HALLINK program has been designed to be essentially transparent to the user (i.e. it performs functionally the same task as the standard link-editor).

In addition to producing explanatory material, HALLINK also performs a template matching function. Whenever a compilation unit is compiled, a template for the unit is created. This template contains a version number which is maintained by the compiler. Upon recompilation of any unit, the compiler checks the compatibility of the new unit with the old by matching the old template with one created during the recompilation. If an incompatibility is found (such as a difference in arguments between two compilations of a COMSUB), the old template is replaced by the new one. The new template has a different version included in it. If no incompatabilities are found, the old template and version remain intact. The resulting version number is incorporated into the object module for the compilation unit and uniquely ties the compiled code to the specific interface requirements of the corresponding templates. This is all performed at compilation time.

Whenever compilation units reference each other, templates for the referenced units are included by the user. From the included template, the compiler extracts the version information placed there during generation of the template. This version information is incorporated in the object code for the referencing unit, thus linking any references to the external modules to code of the proper version.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

HALLINK uses this version information at link edit time to verify that all inter-module references are proper. This checking guarantees that compile time interface checks involving templates are valid for the object modules actually being linked together.

If version mis-matches are found, HALLINK prints a message indicating both the version number expected by a "caller" and the one presented by the "called" routine.

The processing done in HALLINK is generally broken down into three phases:

1) Invoke the standard linkage editor thus performing all library searches and producing a load module with references to the STACK csects unresolved. This load module is written to the TEMPLOAD DD card.

2) Analyze the load module which was put on the TEMPLOAD DD card and create the necessary control sections as object files on the STACKOBJ DD card.

3) Re-invoke the standard linkage editor to incorporate the STACK CSECTS into a final load module which is placed on the SYSLMOD DD card.

4.2.1 Job Control Language

The JCL necessary to execute the HALLINK program is described below. A list of typical JCL statements to which the comments apply is shown in Figure 4.2.

Since the majority of the JCL for HALLINK is identical to that required for the IBM OS/360 Linkage Editor program, only those JCL statements which have specific HALLINK implications are discussed here.

- The EXEC card invoking the link step must specify "HALLINK" as the program to be executed.

- The STEPLIB DD card specifies the location of the load module library which contains the module HALLINK. This card may define any load module library which contains the proper module. The STEPLIB DD card may not be necessary at installations where the module has been made part of the system library (SYS1.LINKLIB).

- The SYSPRINT DD card must specify a BLKSIZE value.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Figure 4.2

```
//LKED     EXEC PGM=HALLINK,REGION=100K
//STEPLIB  DD DISP=SHR,DSN=HALS360.MONITOR
//SYSPRINT DD SYSOUT=A,DCB=BLKSIZE=1210
//SYSLIB   DD DISP=SHR,DSN=HALS360.RUNLIB
//SYSLIN   DD DISP=OLD,DSN=HALOBJ
//           DD DDNAME=SYSIN
//SYSLMOD  DD DSN=HALMOD(GO),DISP=OLD,UNIT=SYSDA,
//              SPACE=(CYL,(1,1,1))
//SYSUT1   DD SPACE=(CYL,(1,1)),UNIT=SYSDA
//STACKOBJ DD SPACE=(TRK,(5,10)),UNIT=SYSDA
//TEMPLOAD DD SPACE=(CYL,(1,1,1)),UNIT=SYSDA
```

- The SYSLIB DD card should designate the HAL/S-360 run time library (HALS360.RUNLIB) as the primary source of unresolved external references. Additional libraries may be concatenated to HALS360.RUNLIB if desired.

- The STACKOBJ DD card specifies a sequential data set onto which generated object decks are placed by HALLINK. The JCL should only specify a device and space allocation. Other parameters are internally determined.

- The TEMPLOAD DD card defines a temporary work partitioned dataset. This DD card should only specify SPACE and UNIT parameters.

Some special considerations may arise when attempting to use features of the OS/360 linkage editor in the HALLINK step. A few comments on certain of these features follow:

a) Provision has been made to pass load module name information to the second link edit step if a NAME card was sent by the user to the first link edit. If the member name on the TEMPLOAD load module is not TEMPNAME, the second link edit step is passed the record:

```
NAME XXXXXXX(R)
```

as part of the generated object decks. The TEMPLOAD member name is determined by the first name found in the directory of that PDS. If the member name was TEMPNAME, no such card will be passed to the second link edit, and it is the user's responsibility to ensure that a name is specified on the SYSLMOD DD card, otherwise the link editor will attempt to store the load module as TEMPNAME.

The user should be fully aware of the consequences of supplying a NAME card without overriding the member name on the catalogued SYSLMOD DD card. This situation will lead to JCL errors if the GO step attempts to use refer-back (PGM=*.LKED.SYSLMOD) to identify the module to be executed.

b) The overlay capabilities of the Linkage Editor should not be used.

c) Any input defined by the SYSLIN DD card, whether object modules or OS link editor control cards are available to only the first invocation of the OS link editor. If it is necessary to make certain link editor input available to the second link edit, that input must be defined via a LINKIN DD card. Any data defined by a LINKIN DD card will be copied unchanged to the STACKOBJ DD card before the HALLINK object output (see below) is placed there.

4.2.2 Inputs

Inputs to the link step are specified directly by object module and link edit control cards or options indicated via the PARM field.

4.2.2.1 Options. Parameters may be passed to HALLINK. The JCL for this is PARM.LKED = 'link parms/HALLINK parms'. The "PARM.LKED" field may be changed to "LINKOPT" when using the standard catalogued procedures.

The slash is optional if no HALLINK parameters are passed.

HALLINK parameters are coded as shown, separated by commas.

Option	Significance
TREE	Causes list of control sections, stack sizes, and immediate sons to be printed. If omitted, no tree will be printed.
BOTH	Pass the LINKPARM to both link edits. If omitted, only second will receive parameters passed by user.
OSLOAD	(or NOGO) causes: a) only link #1 b) output to SYSLMOD c) PARM= 'NCAL,TEST' passed These options are used to construct a user's load library. If either is specified, then all other parameters will be ignored.
XREF	Lists out control section names and the names the programmer actually used.
SDL	Turns off template version checking. If omitted, template versions are checked for consistency.
MSG	Allows printing of HALLINK messages.
DDLIST	Causes printing of any alternate DD list passed to HALLINK.

4.2.3.5 The HALMAP CSECT. The HALMAP CSECT contains the following information about the load module:

1) Number of Process Control Blocks (PCBs) required by the Real Time Executive to handle all potential processes in the module.

2) Address of each PROGRAM, COMSUB, and COMPOOL, and an indicator as to which type each pointer is referencing.

3) Simulation Data File (SDF) file member name containing information about symbols in corresponding compilation units.

Layout of HALMAP

Loc	Length	Description
0	2	Number of entries in pointer table.
2	2	Number of PCBs.
4	-	Pointer table. 12 byte entries as outlined below.
Pointer Table		
0	1	Type (X'00'=COMPOOL, X'01'=PROGRAM, X'03'=COMSUB)
1	3	Address of CSECT.
4	8	SDF member name.

Note that the last entry is not indicated by high order bit being set in type field. Use halfword at location 0 in HALMAP to determine number of entries.

4.2.3.6 HALLINK Return Codes.

Return Codes	Description
0,4,8,12,16	As defined by Link Editor.
1,5,9,13,17	Return code of n correspond to Link Editor return code of (n-1) with recursive calls detected in load module and HALLINK option NOREC not specified.
100	Recursive calls and NOREC not specified.
104	Insufficient space for tables. Rerun in larger partition.
108	Unable to open STACKOBJ or TEMPLOAD.
120+n	Corresponds to return code of n from FIND macro. If 124, most likely caused by allocating TEMPLOAD to a PDS member.
140	Two version definitions of the same control section name encountered. Caused by: a) Re-link editing a HAL load module. b) Two compilation units have first 6 characters of name identical.
144	A control section had a different version number than the number when compiled. Most likely cause by recompiling a compilation unit without re-compiling those which reference it.
148	No version definition received for one or more control sections for which references were made with version information.

4.3 Execution Step

The Execution step is one in which a properly compiled and linked HAL/S program (or series of programs) is executed.

4.3.1 Job Control Language

Execution of a HAL/S-360 program may occur in two different ways: 1) in a stand-alone manner, and 2) under control of a monitoring program. The JCL necessary to operate in these modes is shown in figures 4.3 and 4.4. Descriptions of additional JCL lines follows. The details of operation under the HAL/S-360 Diagnostic System are located in Section 5.9.

The JCL for the execution step contains two distinct groups: 1) that minimal JCL needed to load and begin execution of the HAL/S program, and 2) that JCL which defines data needed by the running HAL/S program due to internal HAL/S I/O requests. Figure 4.3 shows some typical stand-alone JCL to which the comments apply.

- The EXEC card defines the program to be executed. It should contain a REGION parameter of sufficient size to allow execution. The required region is dependent upon program size. A PARM field may be included to pass any requests to the HAL/S runtime facility.

- The STEPLIB card is needed to define the location of the program to be executed, and to define the HAL/S-360 system library in which the "DUMPALL" routine is located. The "DUMPALL" program is used to obtain a formatted HAL/S variable dump at the end of execution under the Stand-Alone execution system. This feature, under control of DUMPALL runtime option, is the only type of dump available under stand-alone operation. Complete dump and trace features are available under the Execution Monitoring System.

Figure 4.3

Execution JCL (no dumps/traces)

```
//GO          EXEC PGM=HALPROG
//STEPLIB     DD DISP=OLD,DSN=HALMOD
//            DD DISP=SHR,DSN=HALS360.MONITOR
//STPLIB      DD DISP=SHR,DSN=HALS360.STPLIB
//HALSDF      DD DISP=OLD,DSN=HALSDF
//CHANNELn    DD <parameters>
              .
              . <other user JCL>
              .
```

Figure 4.4

Execution JCL (for dump and trace capability)

```
//            EXEC PGM=RUNMON
//STEPLIB     DD DISP=SHR, DSN=HALS360.MONITOR
//STPLIB      DD DISP=SHR, DSN=HALS360.STPLIB
//PROGRAM     DD DISP=SHR,DSN=HALS360.DIAGPROC
//SYSPRINT    SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=3458)
//HALLIB      DD DISP=OLD,DSN=HALMOD
//CHANNEL6    SYSOUT=A
//HALSDF      DD DISP=OLD,DSN=HALSDF
//REQUESTS    DD  Request data set
//    <other user JCL>
```

5.1.5.3.2 Direct Access.

A) OPEN

B) READ DI, DK

C) WRITE DI, DK, DKF, DA

D) CHECK

E) SYNADAF, SYNADRIS

F) GETMAIN, FREEMAIN

G) DCB DSORG = DA, MACRF = (RKIC, WAKIC)

5.1.5.4 Miscellaneous.

A) TIME - provides the time of day and date.

B) For real time executive queue elements GETMAIN

5.1.6 List of Names of the Run Time Library

(A indicates alias.)

ALLOW	A	CTOX		EVENTENQ		MM14D3	
ARCCOS		CTRIMV		EVENTPRO	A	MM14SN	
ARCCOSH		DARCCOS		EXCLUDE		MM14S3	
ARCSIN	A	DARCCOSH		EXECTRCE		MM15DN	
ARCSINH		DARCSIN	A	EXP		MM15SN	
ARCTAN	A	DARCSINH		FILEIN		MM16DNP	
ARCTANH		DARCTAN	A	FILEOUT	A	MM16SNP	
BAKTRACE		DARCTANH		FLUSH	A	MM17DN	
BIN		DATAN2		FORMATDA		MM17SN	
BOUT		DATE		GETSEED	A	MM20DNP	
BTOC	A	DCOS		HALEODAD	A	MM20SNP	
CANCEL		DCOSH		HALPRINT	A	MM21DNP	
CANCELT	A	DEXP		HALSIM		MM21SNP	
CANCELTC	A	DIN	A	HALSTART		MM6DN	
CIN	A	DISPACHS	A	HALSYNAD	A	MM6D3	
CINDEX		DISPACHT	A	HALSYS		MM6SN	
CINP		DISPACHW	A	HIN	A	MM6S3	
CLJUSTV		DISPATCH		IIN	A	MOMSTACK	
CLOKTIME		DLOG		INPUT		MSGIOINT	A
CLOSEHAL		DMIDVAL		IOINIT		MV6DN	
COLUMN		DOUT	A	IOUT		MV6SN	
COS	A	DSIN	A	ITOC	A	OTOC	A
COSH		DSINCOS	A	ITOTHEI		OUTPUT	
COUT	A	DSINH	A	KTOC	A	PAGE	
COUTP		DSLD		LINE		PCCOPY	
CPAS		DSQRT		LOCK		PROGINT	
CPASP		DSST	A	LOG		QSHAPQ	
CPRC		DTAN		MIDVAL		RANDOM	
CPSLD		DTANH		MM1DNP		RANDOMG	A
CPSLDP	A	DTOC	A	MM1DSNP		RESET	A
CPSST	A	DTOTHED		MM1SDNP		SCHEDULE	
CPSSTP	A	DTOTHEI	A	MM1SNP		SDATRAP	A
CRJUSTV		DUMPARM	A	MM11DN		SDCTRAP	A
CSHAPQ		DUMPHAL		MM11SN		SDETRAP	A
CSLD		EATAN2		MM12DN		SDFTRAP	A
CSLDP	A	EIN	A	MM12D3	A	SDINIT	A
CSST		ENTERTQE		MM12SN		SDLDUMMY	
CSSTP	A	EOUT		MM12S3	A	SDLSTACK	
CTOB	A	ERRGRP		MM13DN		SDNTRAP	A
CTOD		ERRNUM	A	MM13D3		SDSTRAP	A
CTOE	A	ERRORMON		MM13SN		SDTTRAP	A
CTOI		ERRORSUM	A	MM13S3		SDWTRAP	A
CTOK	A	ETOC	A	MM14DN		SET	
CTOO	A	ETOTHEE					
		ETOTHEI	A				

SETSEED	A
SIGNAL	
SIN	A
SINCOS	
SINH	A
SKIP	
SKIPIN	A
SKIPOUT	A
SQRT	
SVBLOCK	
SVBTOC	
SVDTOC	
SVETOC	
SVITOC	
SVPMSG	
SVSIGNL	
SVSTOP	
SVTDEQ	
SVTENQ	
SVTIME	
SVVSTP	
TAB	A
TAN	
TANH	
TENSTBL	
TERMIN	
TERMINT	
TERMINTC	A
TERMPCB	
TIMECANC	
TIMEINT	
TIMENQ	
UNLOCK	A
UPPRIO	
UPPRIOT	A
VM6DN	
VM6SN	
VO6DN	
VO6D3	A
VO6SN	
VO6S3	A
VV10DN	A
VV10D3	A
VV10SN	A
VV10S3	
VV16DNP	
VV16SNP	
VV6DN	
VV6SN	
VV9DN	A
VV9D3	
VV9SN	A
VV9S3	
WAIT	
WAITDEP	
WAITFOR	
WAITUNTL	A
WHERE	
XTOC	

5.2 Vector-Matrix Operations for HAL/S-360

Conventions Specific to Vector-Matrix Routines

1. General registers 2, 3, and 4 are used for argument pointers.

   a. Result is put into area pointed at by R2.

   b. R3 is pointer to left-hand input argument.

   c. R4 is pointer to right-hand input argument if more than one is required.

2. General registers 0, 1, and 4 are used for size parameters.

   a. R0 is length of vector arguments.

   b. R0 is number of rows for matrix arguments.

   c. R1 is number of columns for matrix arguments.

   d. R4 is used for skip values between elements in partitioned matrices.

3. Scalar inputs and results use F0.

4. System routines (intrinsics) are incorporated into the HAL System package and addressed directly by HALSYS (R12).

   a. Registers 0 through 4, and F0, F2, and F4 are available for usage.

   b. Return to register 14 (R14).

   c. Local branching via HALSYS (R12).

5. Library routines are called via the normal user procedure caller.

   a. Registers are saved automatically.

   b. F0, F2, and F4 available.

   c. R15 for local addressing.

   d. Temporary area off R13 available.

   e. Not loaded unless used.

If C ¬= B...	CPR
WRITE( ) C;	COUT
WRITE( ) $C_{I \text{ TO } J}$;	COUTP
READ C;	CIN
READ $C_{I \text{ TO } J}$;	CINP
READALL C;	CIN
READALL $C_{I \text{ TO } J}$;	CINP

Readall calls the same routines; it is differentiated by the IOINIT input call.

Name	Routine	Intrinsic/ Library	Destination	Source	Arguments
* CAS	Character Assign	I	data	data, vac, literal	Ptr's (R2,R3)
* CASV	Character Assign	I	vac	d,v,l	Ptr's (R2,R3)
* CASP	Partitioned Assign	I	data	partition	Ptr's, 1st char(R1), last char(R0)
* CASVP	Partitioned Assign	I	vac	partition	Ptr's, 1st char(R1), last char(R0)
* CPAS	Character Assign into Partition	L	partition	d,v,l	Ptr's, 1st char(R1), last char(R0)
* CPASP	Partition into Partition	L	partition	partition	Ptr's, 1st char-in(R1), last char-in(R0), last and 1st char-out(R4)
* CAT	Concatenate	I	data	d,v,l	Ptr's (R2,R3,R4)
* CATV	Concatenate	I	vac	d,v,l	Ptr's (R2,R3,R4)
CPR	Character Compare	I	d,v,l	d,v,l	Ptr's (R3,R4) Result in CC & R1
CPRC	Character Compare (based on collating seq)	L	d,v,l	d,v,l	Ptr's (R3,R4) Result in CC
COUT	Character Output	L	output	d,v,l	Ptr (R3)
COUTP	Partitioned Char Output	L	output	d,v,l	Ptr (R3), 1st char(R1), last char(R0)
CIN	Character Input	L	d,v,l	input	Ptr (R3)
CINP	Partitioned Char Input	L	d,v,l	input	Ptr (R3), 1st char(R1), last char(R0)
*CLJUSTV		L	vac	d,v,l	Ptr's (R2,R3), size (R4)
*CRJUSTV		L	vac	d,v,l	Ptr's (R2,R3), size (R4)
*CTRIMV		L	vac	d,v,l	Ptr's (R2,R3)
CINDEX		L	Accum (R1)	d,v,l	Ptr's (R3,R4), result(R1

NOTE: R3 is the source pointer; R2 is the destination pointer. Routines marked with "*" set R3 on exit to value R2 had on input to aid in chained operations.

B) TERMINATE STATEMENT

I. TERMINATE; EXTERNAL NAME: TERMIN no parameter
II. TERMINATE <taskid>; EXTERNAL NAME: TERMINT
III. TERMINATE <taskid>, ...; EXTERNAL NAME: TERMINTC
R0: Parameter: entry point of program or task.

C) CANCEL STATEMENT

I. CANCEL; EXTERNAL NAME: CANCEL no parameter
II. CANCEL <taskid>; EXTERNAL NAME: CANCELT
III. CANCEL <taskid>, ...; EXTERNAL NAME: CANCELTC.
R0: Parameter: entry point of program or task.

D) WAIT <arith exp>; EXTERNAL NAME: WAIT

F0: parameter: # seconds, double precision .

E) WAIT UNTIL <arith exp> ; EXTERNAL NAME: WAITUNTL

F0: parameter: # seconds, double precision .

F) WAIT FOR <event exp>; EXTERNAL NAME: WAITFOR
R0: parameter: point to event expression (see event expression).

G) WAIT FOR DEPENDENT; EXTERNAL NAME WAITDEP
no parameters .

H) SIGNAL <event var>; EXTERNAL NAME: SIGNAL
R0: ptr to event variable (latched or unlatched).

I) SET <event variable>; EXTERNAL NAME: SET
R0: pointer to event variable (must be latched).

J) RESET <event variable>; EXTERNAL NAME: RESET
R0: pointer to event variable (must be latched).

K) UPDATE PRIORITY TO <arith exp> ; EXTERNAL NAME UPPRIO
R0: Priority.

L) UPDATE PRIORITY <taskid> to <arith exp>; EXTERNAL NAME UPPRIOT
R0: Priority.
R1: entry point of program or task.

M) RUNTIME function; intrinsic in HALSYS
F0: current time in seconds returned.

N) NEXTIME<task id>; EXTERNAL NAME NEXTIME
R0: entry point of PROGRAM or TASK.
F0: function value on return.

5.9 Simulation Data File (SDF) Access Package

SDFPKG is an IBM-360 assembly language program comprised of five CSECTS: SDFPKG, LOCATE, PAGMOD, NDX2PTR, and SELECT. Its function is to provide a demand paging form of access to data contained within SDFs. SDFPKG can be separately link edited and employed as a loadable and deletable service module, or it may be linked directly with other software. The latter is the case with the HAL/S-360 stand-alone diagnostic system.

5.9.1 General Considerations

The following is a brief summary of the more important aspects of SDFPKG:

1) SDFPKG is a modular access method for SDFs built upon a demand paging virtual memory foundation. It can be separately linked, loaded, and deleted.

2) All calls to SDFPKG are made through a single ENTRY point by supplying a mode number. Eighteen different mode calls are currently provided.

3) SDFPKG employs a paging area of from 1 to 250 1680-byte pages in size that may be dynamically expanded or contracted as the core memory situation alters.

4) SDFPKG can support simultaneous access to an unlimited number of SDFs. The area needed for FCBs (File Control Blocks) can be automatically provided by SDFPKG or be under the control of the user.

5) SDFPKG is serially reusable. Following a TERMINATE call a new INITIALIZE call my be made.

6) SDFPKG allows SDFs to be modified or merely read.

7) SDFPKG provides built-in binary search algorithms to allow high-level access to data that must be searched.

8) SDFPKG FREEMAINS all storage at the TERMINATE call that it may have GETMAIN'ed since the INITIALIZE call.

9) SDFPKG performs one OPEN (for the HALSDF DD) at INITIALIZE and one CLOSE (same DD) at the TERMINATE call.

10) SDFPKG uses only the following OS services: GETMAIN, FREEMAIN, FIND, BLDL, POINT, READ, WRITE, CHECK, OPEN, CLOSE.

11) SDFPKG can be configured at INITIALIZE so that it will perform no GETMAINs.

12) SDFPKG performs complete error checking and will force an ABEND in case of a legitimate user error or I/O error. A complete set of return codes is used to signal off-nominal conditions that are not reflections of serious user error.

Mode #	Function
17	Locate a Statement Node given the statement number.

Note 1) Definitions and layouts for the various data blocks contained within an SDF can be found in the HAL/SDL Interface Control Document. Different terminology is employed, however, so that the following correspondence may be helpful:

Compiler Spec. Terminology	ICD Terminology	ICD Figure No.
Directory Root Cell	Simulation Table Directory Header	Fig. 2.2.1.2.1.1
Block Data Cell	HAL Block List Member	Fig. 2.2.1.2.1.2.2
Symbol Data Cell	Symbol Data Entry	Fig. 2.2.1.2.2.2
Statement Data Cell	Statement Data Entry	Fig. 2.2.1.2.3.2
Block Node	Block Index Table	Fig. 2.2.1.2.1.2.1
Symbol Node	Symbol Names and Pointers Table	Fig. 2.2.1.2.2.1
Statement Node	Statement Names and Pointers Table	Fig. 2.2.1.2.3.1

Note 2: DSECTs for the pertinent data blocks can be found as members of the HALS.DIAGNSTC.MACLIB dataset:

	Member Name
Directory Root Cell	DROOTCEL
Block Data Cell	BLKCELL
Symbol Data Cell	SYMBDC
Statement Data Cell	STMTDC
Block Node	BLCKNODE
Symbol Node	SYMBNODE
Statement Node (no SRNs)	STMTNOD0
Statement Node (SRNs)	STMTNOD1

For convenience, listings of these DSECTs follow:

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

```
'HALS.DIAGNSTC.MACLIB(DROOTCEL)'
00100            MACRO
00200            DROOTCEL
00300 DROOTCEL   DSECT                    DIRECTORY ROOT CELL
00400 SDFFLAGS   DS     2C
00500 LASTPAGE   DS     H                 # OF LAST PAGE IN SDF FILE
00600 SDFDATE    DS     F                 DATE OF CREATION
00700 SDFTIME    DS     F                 TIME OF CREATION
00800 LASTDPGE   DS     H                 # OF LAST DIRECTORY PAGE
00900 COMPOOLS   DS     H                 # OF INCLUDED COMPOOLS
01000 BLKNODES   DS     H                 # OF BLOCK NODES
01100 SYMNODES   DS     H                 # OF SYMBOL NODES
01200 FBNPTR     DS     A                 POINTER TO FIRST BLOCK NODE
01300 LBNPTR     DS     A                 POINTER TO LAST BLOCK NODE
01400 INSTRCNT   DS     H                 NO. OF EMITTED MACHINE INSTRUCTIONS
01410 FREEBYTE   DS     H                 TOTAL AMT OF FREE SPACE IN SDF
01500 DLSTHEAD   DS     H                 LIST HEAD FOR DECLARED VARS (BY ADDR)
01502 RLSTHEAD   DS     H                 LIST HEAD FOR REMOTE VARS (BY ADDR)
01600 FSNPTR     DS     A                 POINTER TO FIRST SYMBOL NODE
01700 LSNPTR     DS     A                 POINTER TO LAST SYMBOL NODE
01800 CUBTCPTR   DS     A                 PTR TO COMP. UNIT BLOCK DATA CELL
01900 BTREEPTR   DS     A                 POINTER TO ROOT OF BLOCK TREE
02000 FSTMTNUM   DS     H                 FIRST STATEMENT NUMBER
02100 LSTMTNUM   DS     H                 LAST STATEMENT NUMBER
02200 EXECSTMT   DS     H                 # OF EXECUTABLE STATEMENTS
02300 STMTNODE   DS     H                 # OF STATEMENT NODES
02400 FSTNPTR    DS     A                 POINTER TO FIRST STATEMENT NODE
02500 LSTNPTR    DS     A                 POINTER TO LAST STATEMENT NODE
02600 SNELPTR    DS     A                 POINTER TO STATEMENT NODE EXTENT LIST
02700 FIRSTSRN   DS     CL8
02800 LASTSRN    DS     CL8
02900 CUBTCNUM   DS     H                 BLOCK NUMBER OF UNIT BLOCK
02910 COMPUNIT   DS     H                 COMPILATION UNIT ID CODE
02920 TITLEPTR   DS     F                 VIRTUAL MEMORY POINTER TO TITLE INFOR
03100 USERDATA   DS     CL8               FREE FOR USER DATA
03102 SYMBCNT    DS     F                 ACTUAL NUMBER OF SYMBOLS IN COMP.
03104 MACROCNT   DS     F                 TOTAL SIZE OF MACRO TEXT (BYTES)
03106 LITSCNT    DS     F                 TOTAL NUMBER OF LITERAL STRINGS
03108 XREFCNT    DS     F                 ACTUAL NUMBER OF XREF ENTRIES
03200 DRCLEN     EQU    *-DROOTCEL
03300            MEND
```


INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

5.9.7.4 Mode 3 - Rescind Paging Area Augments.

A.	Input:	MODE	3
B.	Output:	APGAREA, AFCBAREA, NBYTES	0
		NPAGES	Number of pages which can yet be added to the Paging Area.

5.9.7.5 Mode 4 - Select an SDF (Explicitly).

A.	Input:	DISP	0 (Auto-Select parameter should not be specified)
		MODE	4
		SDFNAM	8 character SDF name, e.g. ##NAVIGA
B.	Output:	R15, CRETURN	0 → Select successful 8 → BLDL unsuccessful (member not found) 12 → FCB Area is exhausted (only if user is supplying FCB Areas)

5.9.7.6 Mode 5 - Locate Pointer.

A.	Input:	DISP	{SELECT, MODF, RESV, RELS}
		MODE	5
		PNTR	Virtual memory pointer to be located
B.	Output:	R1, ADDR	Core address corresponding to "located" pointer

5.9.7.7 Mode 6 - Set Disposition Parameters.

A.	Input:	DISP	{MODF, RESV, RELS}
		MODE	6
B.	Output	None	

5.9.7.8 Mode 7 - Locate Directory Root Cell.

A.	Input:	DISP	{SELECT, MODF, RESV, RELS}
		MODE	7
B.	Output:	R1 ADDR	Core address of Directory Root Cell
		PNTR	Virtual memory pointer to Directory Root Cell

5.9.7.9 Mode 8 - Locate Block Data Cell given Block Number.

A.	Input:	DISP	{SELECT, MODF, RESV, RELS}
		MODE	8
		BLKNO	Block Number
B.	Output:	R1 ADDR	Core address of Block Data Cell
		PNTR	Pointer to Block Data Cell
		BLKNLEN	Number of characters in block name
		CSECTNAM	Name of code CSECT of block
		BLKNAM	Block Name (up to 32 EBCDIC characters)

RESERVES — Total reserve count (sum of reserve counts of all active core slots)

The TERMINATE call (mode 1) zeros out this data area so that the values must be extracted prior to the call. These parameters are maintained dynamically and may be accessed at any time between the INITIALIZE and TERMINATE calls.

5.9.9 HAL Variable Dump Module (DUMPALL)

DUMPALL is an IBM-360 assembly language program comprised of seven CSECTs (DUMPALL, DUMPUNIT, HALCALL, DSTRUCT, DUMPVAR, FIXSTACK, and RESOLVE) in addition to the five CSECTS of SDFPKG, which are all linked together to form a single load module of approximately 21,000 bytes in size.

DUMPALL is constructed from basic elements of the HAL/S-360 stand-alone diagnostic system and is intended for use in applications where the full machinery of the diagnostic system is either not desired or is not available.

The purpose of DUMPALL is to print the names, attributes, and current values of all HAL variables contained within the HAL load module on the Message Channel (normally CHANNEL6 which is assigned to the line printer). In the current implementation, DUMPALL handles all HAL variables with the exception of stack variables. Although the intent of DUMPALL is to provide post-mortem dumps in the absence of the stand-alone diagnostic system, DUMPALL may be invoked at any time after HALSTART is called and before it exits.

Since DUMPALL is a self-contained load module, it may be dynamically loaded and invoked. As mentioned previously, the main prerequisite for its use is that the Message Channel still be open. DUMPALL submits character data for printing by making use of Field 18 (Print Service) of the Simulation Vector Table (see Figure 2.4.1 of the HAL/SDL ICD). In addition, DUMPALL makes use of the four character conversion routines (SVBTOC, SVITOC, SVETOC, SVDTOC) that are also accessible via the Simulation Vector Table.

The output of DUMPALL is identical to that obtained when the following command is serviced by the stand-alone diagnostic system:

```
AT END: DUMPALL;
```

DUMPALL is called via standard OS-360 linkage conventions with register 1 pointing to a 12 byte data area of the following form:

Field		
1	Address of HALSTART	4
2	DDNAME of PDS containing SDFs (EBCDIC characters)	8

Field 2 normally would contain the characters HALSDFbb.

Since DUMPALL employs the services of SDFPKG there is a possibility that some of the SDFPKG Abend codes could result from a DUMPALL invocation.

If the SDF DCB cannot be opened successfully, then DUMPALL will return with a return code of 4 in register 15.

DUMPALL performs no GETMAINs and it CLOSEs the SDF DCB prior to returning control.

Fields:

D = Operand Field: 16 bits of data with significance depending upon Q (See table below).

T1, T2 = General purpose tag fields: of 8 bits and 3 bits respectively. The significance of these fields depends upon the type of Operator Word preceding the Operand Word.

Q = Operand Qualifier Tag: 4 bits which determine the significance of the Operand Field, according to the following table:

Qualifier Q (hex)	Mnemonics	Value of Operand Field
0	-	either empty or reserved for a special purpose.
1	SYT or SYL	a symbol table pointer (either mnemonic is used depending upon context).
2	GLI or INL	an internal flow number reference (either mnemonic is used, depending upon context).
3	VAC	"virtual accumulator," a back pointer to the result of a previous HALMAT instruction.
4	XPT	an extended pointer.
5	LIT	a pointer into the literal table.
6	IMD	an actual numerical value used by the operator.
7	AST	an asterisk pointer.
8	CSZ	component size.
9	ASZ	array or copy size.
A	OFF	an offset value.

This document also uses the following generic mnemonics for the Q field of operand words:

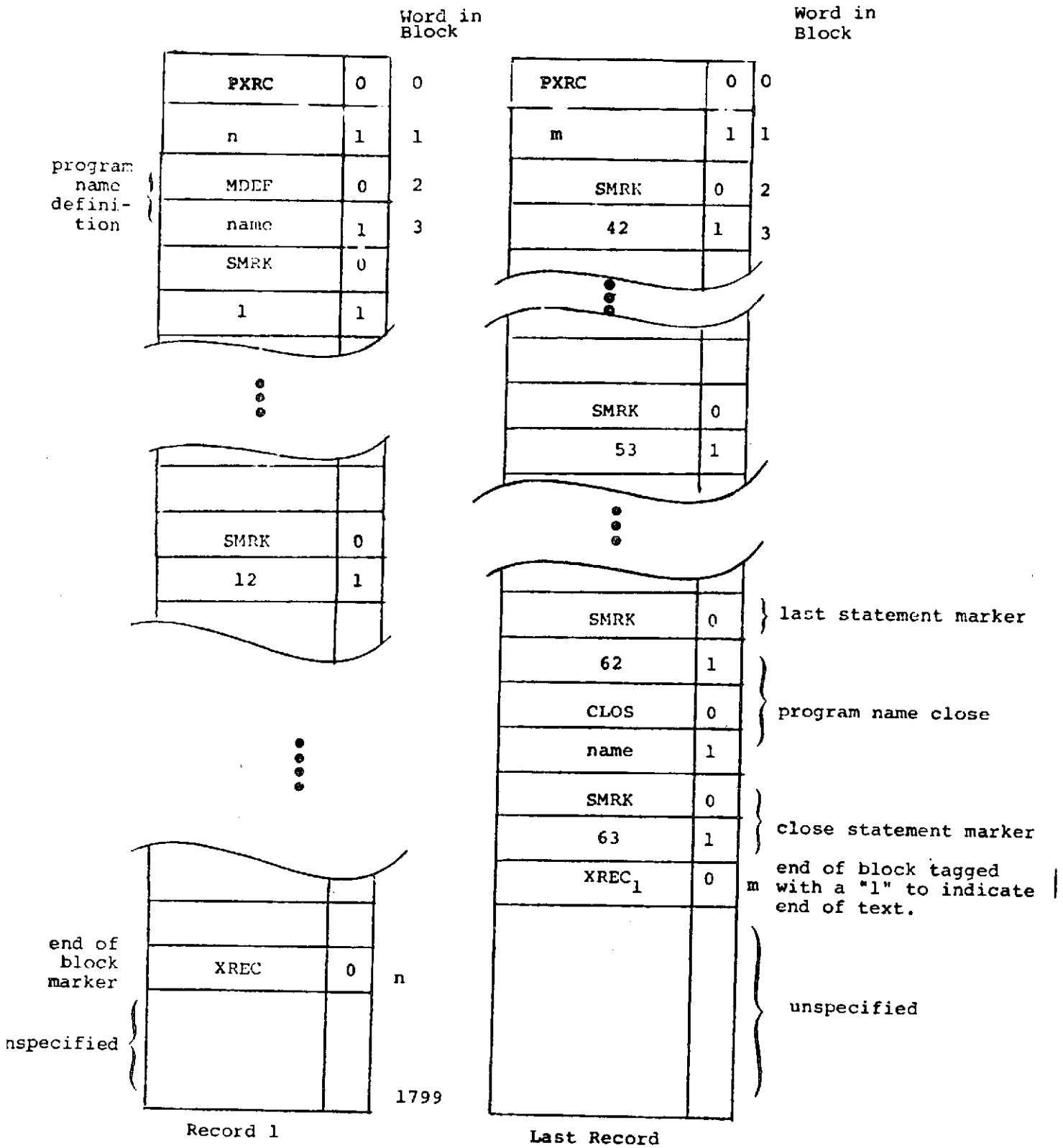
EXV	external variable: may be any of SYT, LIT, VAC, XPT.
EEV	extended external variable: may be any of SYT, LIT, VAC, XPT, IMD.
ESV	either SYT or XPT.
EVV	either SYT, XPT, or VAC.

The HAL/S source text for an entire program consists of possible template information and the body of the primary block of the compilation. Phase 1 of the compiler converts each statement to a "PARAGRAPH" of HALMAT text. Each PARAGRAPH of HALMAT text consists of a sequence of HALMAT instructions derived from the source text followed by a SMRK instruction. If no HALMAT text is generated, then the derived PARAGRAPH consists only of the SMRK instruction.

As many whole PARAGRAPHS of HALMAT text as possible are stored on each "RECORD", which contains at most 1800 OPERAND and OPERATOR WORDS (in each block of a disk file[1]). A PXRC operator is always the first operator in a RECORD; an XREC operator follows the last PARAGRAPH on each RECORD. The final XREC in a compilation contains a tag of 1 in the general purpose tag field. This situation is presented diagrammatically in the Figure on the next page, which shows the macroscopic structure of the HALMAT text for a HAL/S program with no COMPOOL declarations. The first significant instruction is the program definition head: MDEF. This is matched by a closing CLOS instruction at the end of the text.

[1] Both Phase I and Phase II of the HAL/S compiler use a disk file block size of 7200 bytes.

INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840



Layout of HALMAT Files

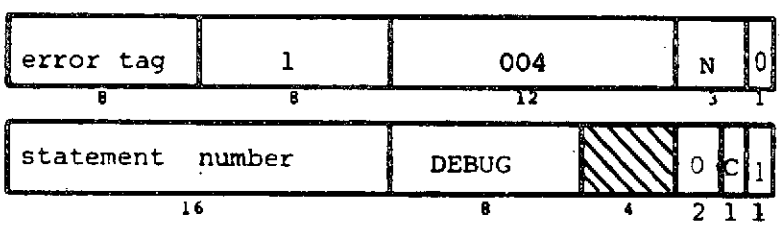
A.1.1 Formatting Operators

- NOP No Operation

- XREC End of HALMAT Record

tag = 0 for all HALMAT blocks except last.
= 1 for last HALMAT block only.

- SMRK Statement Marker (General)



SMRK follows the generated code of each HAL/S source statement not contained in an **Inline** Function Block.

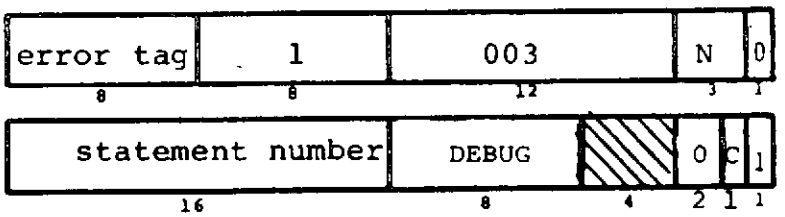
"Error Tag" = maximum statement error severity, 0 if no errors.

"Statement Number" = source statement number generated in Phase 1.

"C" = 0 for statements with no HALMAT code, 1 otherwise.

"Debug" = number used for compiler testing; normally 0.

- IMRK Statement Marker (Inline Functions)



IMRK follows the generated code of each HAL/S source statement within an Inline Function Block.

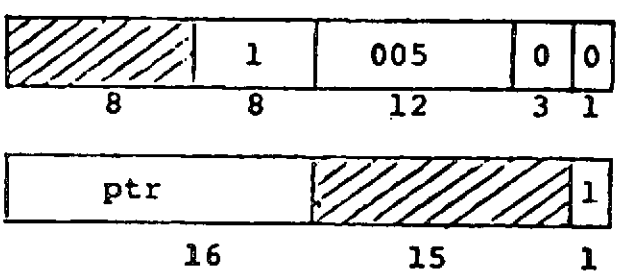
"Error Tag" = maximum error severity, 0 if no errors.

"Statement Number" = source statement number generated in Phase 1.

"C" = 0 for statements with no HALMAT code, 1 otherwise.

"Debug" = number used for compiler testing; normally 0.

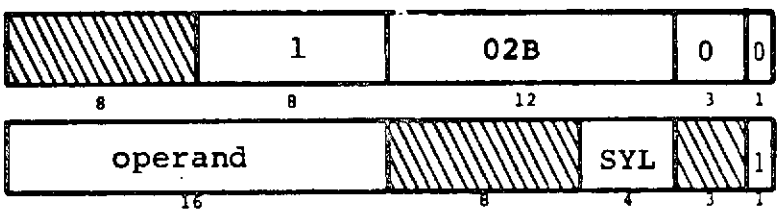
- PXRC Pointer to XREC



PXRC is the first operator in each HALMAT block. PTR is the index of the XREC for that block.

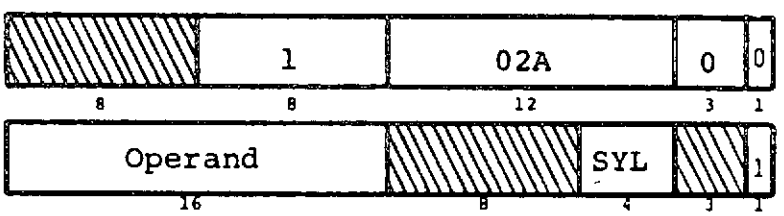
A.1.2 Program Organization Operators

- MDEF Program Definition Header



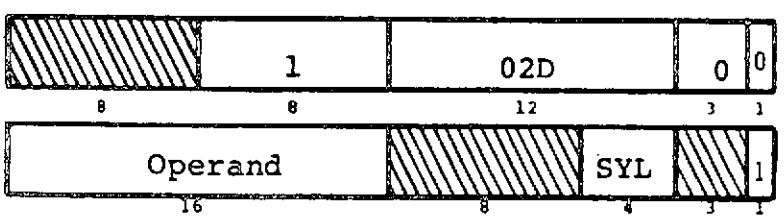
"Operand" points to the program name in the symbol table.

- TDEF Task Definition Header



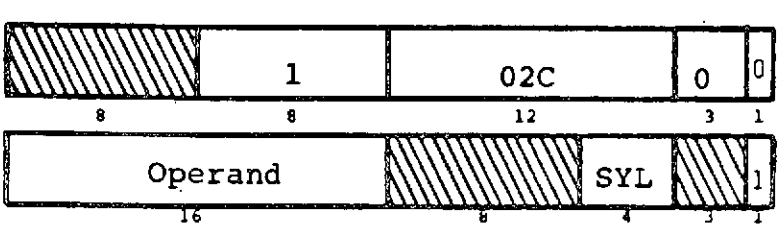
"Operand" points to the task name in the symbol table.

- PDEF Procedure Definition Header



"Operand" points to the procedure name in the symbol table.

- FDEF Function Definition Header

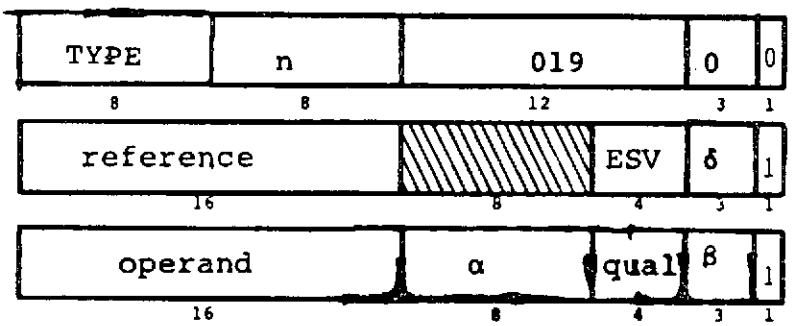


"Operand" points to the function name in the symbol table.

Kind of Subscript	Number of Operands		α	β	qual
*	1		8	-	AST
index	1		9	-	EEV
to-partition [① to ②]	2	①	A	1	IMD
		②	A	0	IMD
at-partition [① AT ②]	2	①	B	1	IMD
		②	B	0	EEV

The following HALMAT operator specifies both array and component subscripting.

- DSUB - regular subscript specifier



number of operands
n ≥ 2

δ = 1 for assign context.

"reference" is a direct or indirect reference to the data item referenced.

"type" is the result type of the data item after possible modification by component subscripts.

Following the first operand those are between one and five groups of operands, each group specifying one subscript from the list of subscripts. Each group may have between one and four operands - depending on the kind of subscript. The subscript list is represented in left-to-right order, array subscripting list.

The following table shows the possible forms of an operand group.

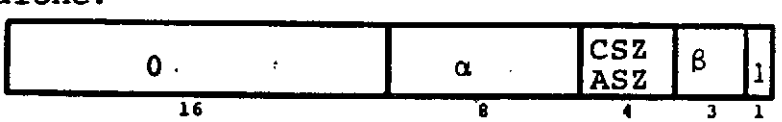
Kind of Subscript	Number of Operands (See Note)	α		β	qual		
		array	component		array	character component	other component
*	1	4	0	-	AST	AST	AST
index	1	5	1	-	EEV,ASZ	EEV, CSZ	EEV
to-partition	2 ①	6	2	1	IMD	EEV, CSZ	IMD
[① to ②]	②	6	2	0	IMD	EEV, CSZ	IMD
at-partition	2 ①	7	3	1	IMD	EEV	IMD
[① at ②]	②	7	3	0	EEV,ASZ	EEV, CSZ	EEV

Note: If an operand has a "qual" of CSZ or ASZ then it may be immediately followed by an extra subsidiary operand. CSZ/ASZ operands correspond to specification of # expressions in character subscripting when the size is not known at compile time:

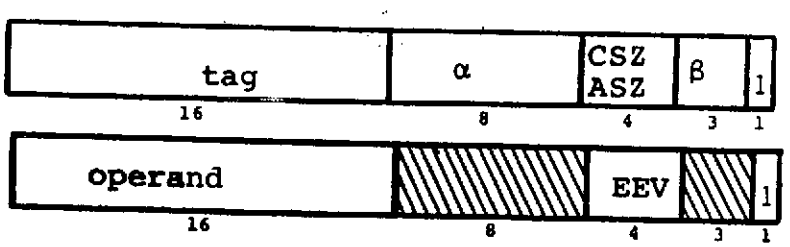
ASZ - * array size

CSZ - character strings

a) # alone:



b) # ± expression



tag = 1 + expression
2 - expression

extra subsidiary operand specifying expression reference

TERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

A.2 Index of HALMAT Operators

Operator	Mnemonic	Page
Class 0:		
0	NOP	6
1	EXTN	91
2	XREC	6
3	IMRK	7
4	SMRK	6
5	PXRC	7
7	IFHD	49
8	LBL	49
9	BRA	50
A	FBRA	50
B	DCAS	51
C	ECAS	52
D	CLBL	51
E	DTST	52
F	ETST	53
10	DFOR	54-55
11	EFOR	57
12	CFOR	56
13	DSMP	57
14	ESMP	57
15	AFOR	56
16	CTST	53
17	ADLP	89
18	DLPE	90
19	DSUB	93/101
1A	IDLP	90
1B	TSUB	92/100
1D	PCAL	61
1E	FCAL	61
1F	READ	62
20	RDAL	62

Operator	Mnemonic	Page
Class 0 (Con't.)		
21	WRIT	63
22	FILE	63
25	XXST	58
26	XXND	59
27	XXAR	58/100
2A	TDEF	8
2B	MDEF	8
2C	FDEF	8
2D	PDEF	8
2E	UDEF	9
2F	CDEF	9
30	CLOS	9
31	EDCL	9
32	RTRN	11
33	TDCL	10
34	WAIT	81
35	SGNL	81
36	CANC	82
37	TERM	82
38	PRIO	83
39	SCHD	83/84
3C	ERON	80
3D	ERSE	80
40	MSHP	76
41	VSHP	76
42	SSHP	74
43	ISHP	75
45	SFST	59
46	SFND	60
47	SFAR	60
4A	BFNC	64

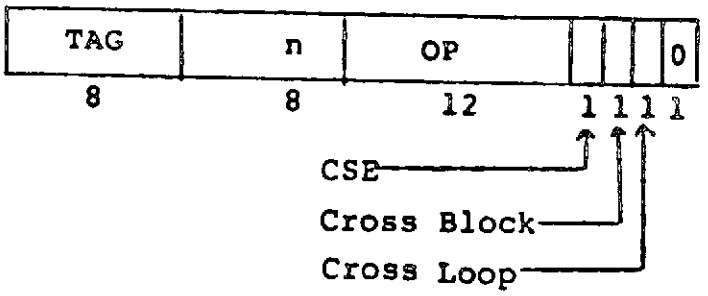
INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

Operator	Mnemonic	Page
Class 8: Initialization		
01	STRI	85
02	SLRI	86
03	ELRI	86
04	ETRI	85
21	BINT	87
41	CINT	87
61	MINT	87
81	VINT	87
A1	SINT	87
C1	IINT	87
E2	TINT	87
E1	NINT	98/99
E3	EINT	87

A.3 Optimizer HALMAT

The HALMAT produced by the Optimizer differs in many respects from the HALMAT as originally produced by Phase 1. The augmented formats described in this section are those which are subsequently utilized by the code generation phase. Code generators which are not equipped to handle these augmented formats should inhibit optimization from being performed.

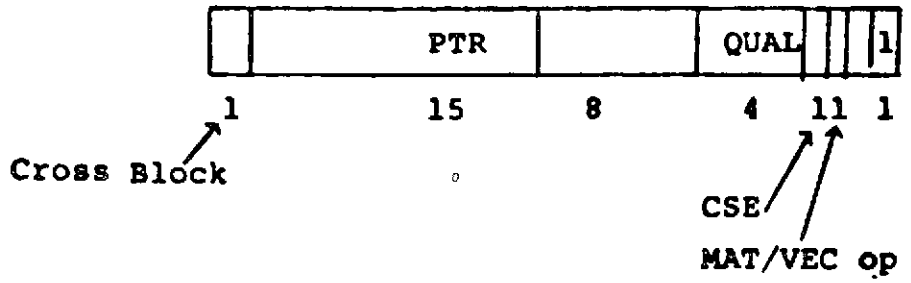
A.3.1 Changes in Operator Format



- CSE = 1 if the result is referenced more than once. (If OP=XREC, indicates next HALMAT block is expansion of current block, and potential cross block references exist.)
- Tag = 1 (for vector/matrix operations inside vector/matrix loops) indicates the result of the operation is referenced outside the vector/matrix loop.
- Cross Block = 1 if the result is referenced in the next HALMAT block (needed for subscript common expressions also).
- Cross Loop = 1 inside array loops if the result is referenced from within a different array loop.

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

A.3.2 Changes in Operand Format



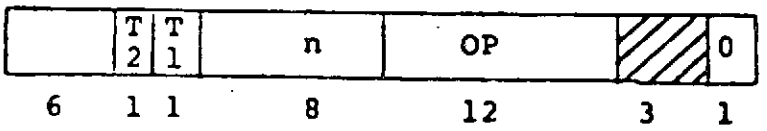
- CSE = 1 if Qual is VAC and operator is referenced by Inter VAC operands.

- Cross Block = 1 if PTR refers to the last previous HALMAT block (needed for subscript common expressions also).

- MAT/VEC op = 1 if operand inside a vector/matrix loop possesses the vector/matrix arrayness.

A.3.3 Register Tags in Conditionals

The comparison operators are class 7 HALMAT operators from "725" = BNEQ to "7CA" = ILT.

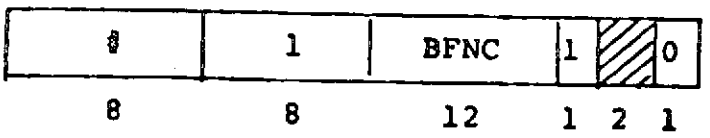


T1 = 1 if only comparison operator in the statement.

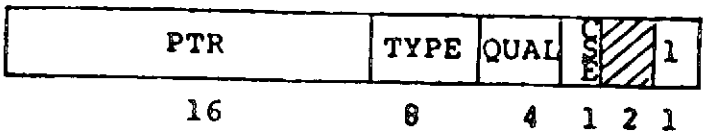
T2 = 1 if register environment can be preserved to the next comparison operator. This is true if the next comparison operator can only be executed following the current operation.

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

A.3.4 SINCOS Function



operator



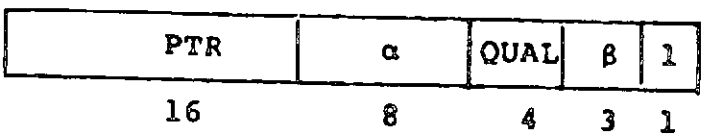
operand

If θ = "39" then VAC's point to the operator word for the SIN and the operand word for the COS.

If θ = "3A" then VAC's point to the operator word for the COS and the operand word for the SIN.

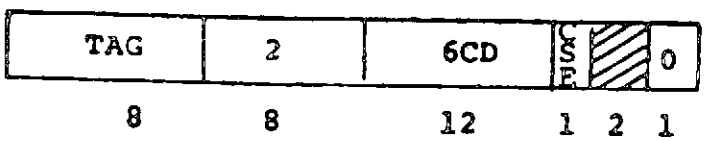
Common subexpressions for the SIN and COS are indicated (separately) by VAC's in the normal manner.

A.3.5 Subscript Common Expressions



A final operand for the DSUB operator may be added having α = 5 and β = 1. This operand is a quantity to be added to the subscript computation before shifting for type, alignment, etc. takes place.

The Integer Integer Product operator is changed to:



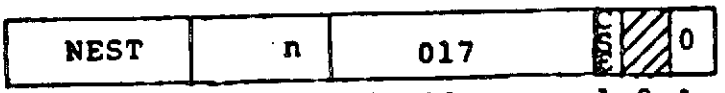
TAG = 1 if the IIPR is generated by the optimizer in a subscript computation.

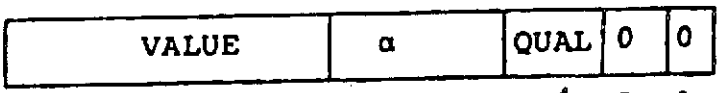
A.3.6 Inline Vector/Matrix Loops and Loop Combining

Inline vector/matrix loops may be generated for the following operators:

MASN	VASN	IASN (e.g. VEC=0)
MNEG	VNEG	
MADD	VADD	
MSUB	VSUB	
MSPR	VSPR	
MSDV	VSDV	

The ADLP arrayness specifier is changed:





CSE = 1 if the loop refers to vector/matrix arrayness only (i.e. the number of elements in the vector or matrix).

α = 0 except for the last operand.

For the last operand:

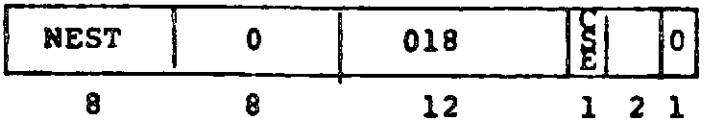
α = 1 if vector/matrix arrayness present (only one VDLP operand present).

α = 2 if a vector/matrix arrayed operation is referenced outside the loop or a CSE is referenced from within a subsequent loop.

α = 4 if the ADLP from Phase 1 has been denested or vector/matrix arrayness has been denested in with regular arrayness.

These conditions may be OR'ed.

The DLPE end arrayness specifier is changed:



CSE = 1 if the loop refers to vector/matrix arrayness only.

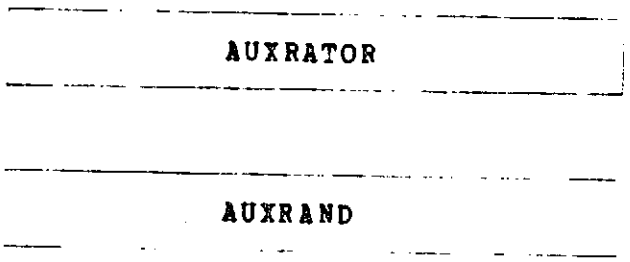
A.4 AUXILIARY HALMAT

A.4.1 Introduction

This section serves as a specification of AUXILIARY HALMAT (AUXMAT) used in the HAL compiler system releases 360-17 and PC-12. AUXMAT is produced by the AUXILIARY HALMAT GENERATOR (AUXMATER), a phase of the HAL compilers which follows the OPTIMIZER and precedes the code generation phase (Phase 2). AUXMAT is specifically designed to convey certain types of machine independent information useful for machine dependent optimizations which might be performed during code generation.

A.4.2.0 General Description

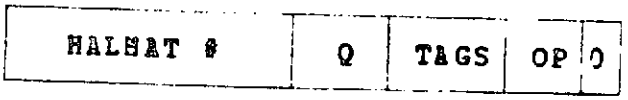
AUXMAT conveys information to Phase 2 of the HAL compiler via 32 bit operator words (AUXRATORs) and 32 bit operand words (AUXRANDs). Each piece of AUXMAT information is conveyed via an ordered pair:



The fields and general field descriptions within the AUXRATOR and AUXRAND words are described in this section. Section A.4.3 describes the specific information conveyed by AUXMAT and the actual values associated with the various fields.

A.4.2.1 AUXRATOR Format

The AUXRATOR has the following format:



INTERMETRICS INCORPORATED • 701 CONCORD AVENUE • CAMBRIDGE, MASSACHUSETTS 02138 • (617) 661-1840

The HALMAT @ is a 16 bit pointer to the HALMAT operator or operand for which auxiliary information is being supplied.

Q is a 5 bit field which, together with the PTR field supplied in the AUXRAND (see Section A.4.2.2), describe the piece of data for which information is supplied. In general, these correspond to the Q and D fields, respectively, found in a HALMAT operand.

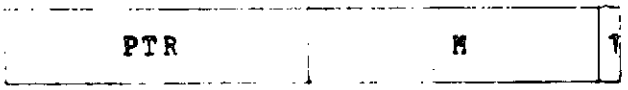
TAGS is a 6 bit general purpose tag field used to further supplement the information provided by the AUXMAT ordered pair.

OP is a 4 bit field which specifies the type of information being conveyed by the AUXMAT pair in question.

The low order bit of an AUXRATOR is always 0 to distinguish it from an AUXRAND.

A.4.2.2 AUXRAND Format

The AUXRAND has the following format:



The PTR field (in conjunction with the Q field) is a general purpose 16 bit field used to specify a piece of data for which AUXMAT information is being supplied.

The M field is a general purpose 15 bit field used to convey the AUXMAT information being supplied.

The lower order bit of an AUXRAND is always 1 to distinguish it from an AUXRATOR.

A.4.2.3 Passing of AUXMAT Between Phases

AUXMAT, like HALMAT, is passed on a disk file. Each record in the file contains 900 AUXMAT ordered pairs. The last ordered pair is always an END OF AUXMAT. (see Section A.4.3) While AUXMAT is designed to be read in parallel with HALMAT, it is not divided into "PARAGRAPHS". Instead, XREC synchronization ordered pairs are provided at appropriate points within the AUXMAT records. Only unused portions of the particular record containing the END OF AUXMAT remain

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

unspecified. No XREC synchronization operator is supplied for the last HALMAT block. In general, fewer than 900 AUXMAT ordered pairs are generated per HALMAT block. The resulting parallel structure between AUXMAT and HALMAT is illustrated in Figure 2.1.

A.4.3 AUXMAT Operations

In its current design, AUXMAT conveys 6 different types of information. These pieces of information are:

1) NEXT USE of a HAL variable or VAC

2) Burn all registers

3) HALMAT XREC synchronization

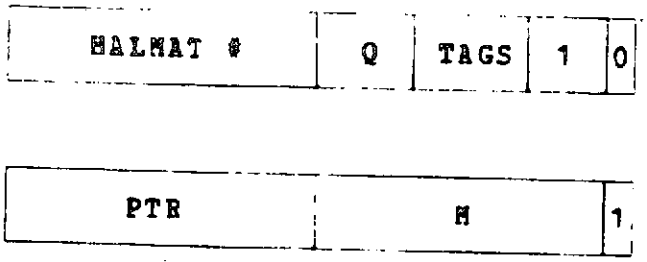
4) VAC targeted for a HAL built-in function argument

5) Loop invariant HAL variables

6) END OF AUXMAT

Information layout in the AUXMAT ordered pair for each of the above is supplied below. Any fields which have a fixed value for that particular ordered pair contain a decimal number denoting that fixed value. A short description of each of the fields without fixed values and the particular meaning of that field is also provided. The HALMAT # field is always as described in Section A.4.2.1.

NEXT USE



Q and PTR describe the HAL variable or VAC for which NEXT USE information is being passed.

INTERMETRICS INCORPORATED · 701 CONCORD AVENUE · CAMBRIDGE, MASSACHUSETTS 02138 · (617) 661-1840

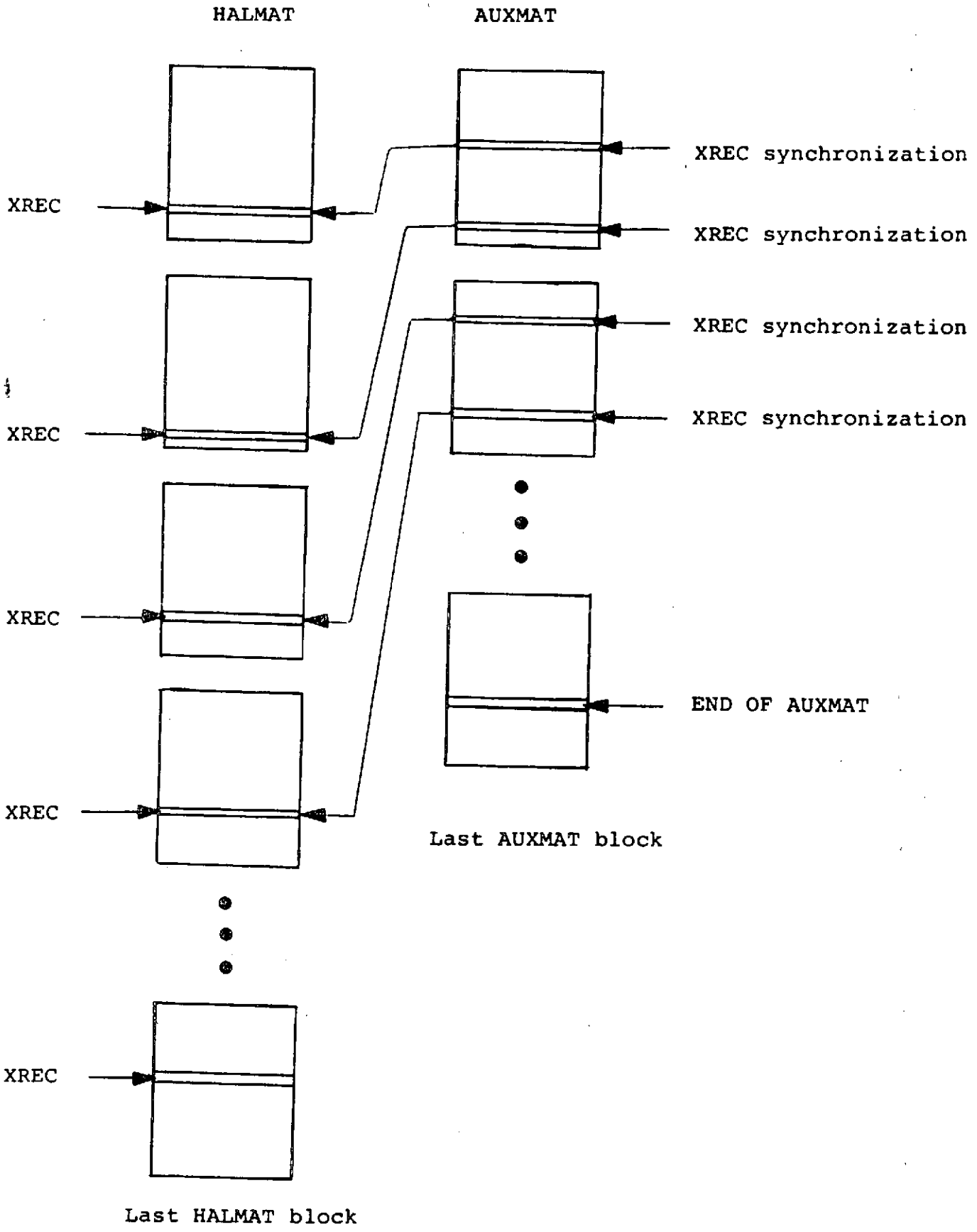


Figure 2-1: Parallel Layout of HALMAT and AUXMAT

TAGS may have a non zero value only if Q and PTR describe a CSE. In these cases, the value of TAGS is:

1 if the HAL data item referred to is a CSE and it is next used only in the true part of an IF THEN ELSE statement.

2 if the HAL data item referred to is a CSE and a reference to it exists beyond an IF THEN ELSE statement.

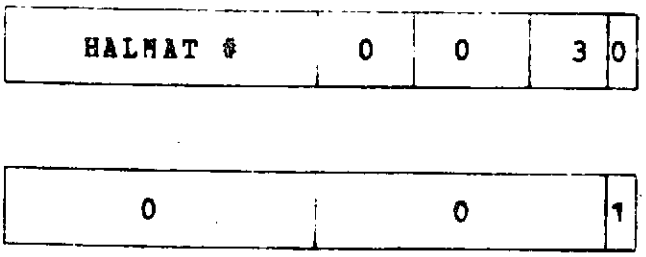
3 if both of the above apply.

The value of M is the NEXT USE value of the HAL variable or VAC.

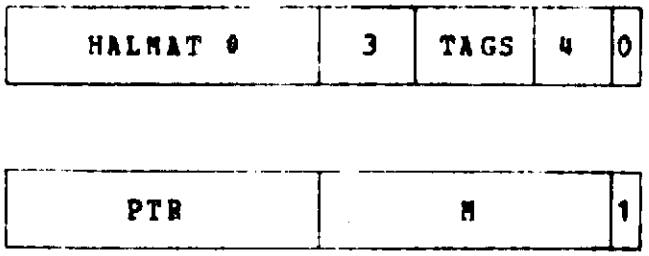
Burn All Registers

Burning of registers and register environments is currently handled in the code generation phase of the compiler. The AUXMATER does not generate this type of ordered pair.

HALMAT XREC Synchronization



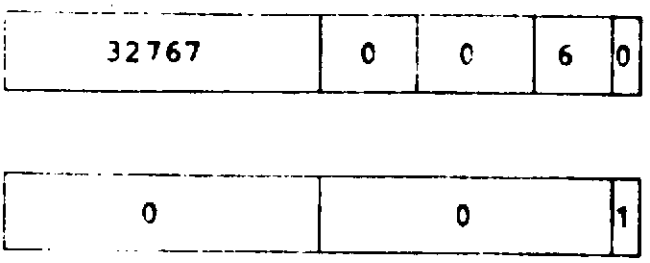
Target Information



TAGS contains the position number of the HAL built-in function argument for which the VAC will eventually be used.

M contains the number of the built-in function for which the VAC in question is an argument.

Loop Invariant HAL Variables

The AUXMATER does not generate loop invariant information.

END OF AUXMAT