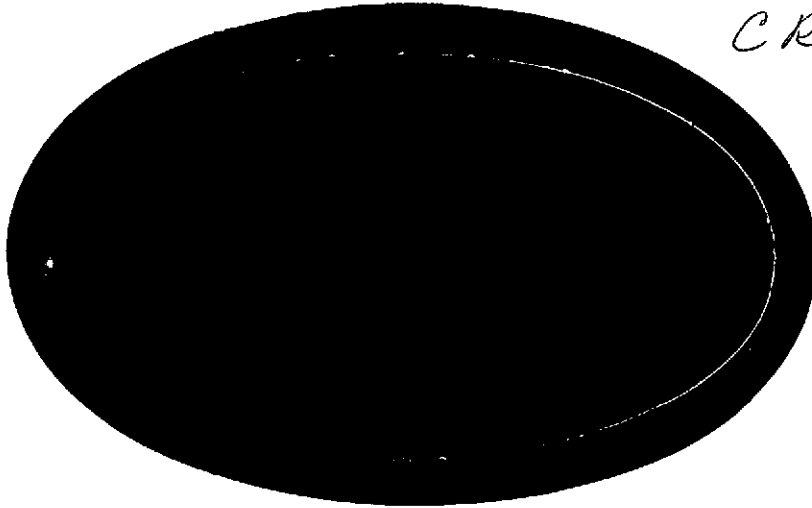


nif

CR 134284

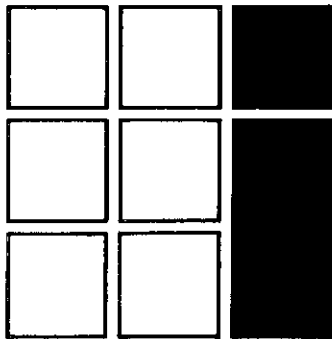


(NASA-CR-134284) HAL/S-360-USER'S MANUAL
(Intermetrics, Inc.) -202-p HC
206

CSCL 09B

N74-25728

G3/08 Unclas
39737



INTERMETRICS

Reproduced by
NATIONAL TECHNICAL
INFORMATION SERVICE
US Department of Commerce
Springfield, VA. 22151

PRICES SUBJECT TO CHANGE

HAL/S-360
USER'S MANUAL
IR-58-5

April 7, 1974

Prepared by:

R. E. Kole
P. H. Helmers
R. L. Hotz

Approved by:



Daniel J. Lickly
HAL Language/Compiler Dept. Head

Approved by:



Dr. F. H. Martin
Shuttle Program Manager

FOREWORD

This document was prepared for the Johnson Space Center, Houston, Texas, under contract NAS9-13864.

This version of the User's Manual corresponds to Release 7 of the HAL/S-360 compiler system.

The black lines in the margins of the text indicate new changes since the previous version of the HAL/S-360 User's Manual, issued on February 18, 1974




TABLE OF CONTENTS

1.	INTRODUCTION	
1.1	Purpose of This Manual	1-1
1.2	Scope of This Manual	1-1
2.	RUNNING A HAL/S PROGRAM	2-1
2.1	Communication with OS/360 - Job Control Language	2-1
2.1.1	Introduction	2-1
2.1.2	The Catalogued Procedure	2-1
2.1.3	The Optional Parameters: OPTION and RUNPARM	2-2
2.1.4	Specifying the Source Language Input	2-2
2.1.5	Specifying the Standard Execution-Time Input	2-3
2.1.6	Specifying the Standard Execution Time Output	2-3
2.1.7	Specifying Additional Execution-Time JCL	2-4
2.1.8	A typical Run Submission	2-4
2.2	Compiler Outputs	2-7
2.2.1	Source Listing	2-7
2.2.2	Tables	2-7
2.2.3	Summaries	2-7
2.2.4	Diagnostics	2-8
2.3	Subsequent Steps	2-8
2.3.1	Link Step	2-8
2.3.2	The Execution Step	2-9
2.4	Creating and Running Program Complexes	2-9

2.4.1	Introduction	2-9
2.4.2	The Form of Compools and Comsubs	2-9
2.4.3	Compiling a Program Complex	2-11
2.4.4	An Example of Program Complex Compilation	2-12
3.	COMPILATION LISTINGS	
3.1	General Description	3-1
3.2	Formats	3-1
3.2.1	Headings	3-1
3.2.2	Statement Number	3-1
3.2.3	Line Type	3-2
3.2.4	Source Field	3-2
3.2.5	Current Scope Field	3-2
3.2.6	Information Field	3-3
3.3	The Output Writer	3-3
3.3.1	Concept	3-3
3.3.2	Auto-Indentation	3-4
3.3.2.1	Declaration Statements	3-4
3.3.2.2	Labels	3-4
3.3.2.3	Scope	3-4
3.3.2.4	IF Statements	3-5
3.3.2.5	DO Groups	3-5
3.3.2.6	Continuations	3-5
3.3.2.7	Page Boundaries	3-5
3.3.2.8	First Lines	3-6
3.3.3	Multi-line Expansion and Annotation	3-6
3.3.3.1	Overpunches	3-6
3.3.3.2	Array and Structure Notation	3-6
3.3.3.3	Subscripts and Exponents	3-7
3.3.3.4	REPLACE'd Symbols	3-7
3.3.4	Comments	3-7
3.3.4.1	Comment Cards and Directive Cards	3-7
3.3.4.2	In-Line Comments	3-7

3.4	Block Summaries	3-9
3.4.1	Concept	3-9
3.4.2	Informaiton Provided	3-9
3.5	Program Layout Summary	3-11
3.6	Symbol & Cross Reference Listing	3-11
3.6.1	"DCL" Field	3-12
3.6.2	"NAME" Field	3-12
3.6.3	"TYPE" Field	3-12
3.6.4	"ATTRIBUTE & CROSS REFERENCE" Field	3-13
3.7	Macro Table	3-15
3.8	Optional Unformatted Listing	3-15
3.8.1	Format	3-16
3.9	Additional Information	3-16
3.10	Phase II Listing	3-16
3.11	Phase III Listing	3-17
4.	DEBUGGING AIDS (Non-Real Time)	4-1
4.1	Compilation Errors	4-1
4.1.1	Message Format	4-1
4.1.2	Classification Scheme	4-1
4.1.3	Error Severity	4-2
4.1.4	Phase I error Summary	4-2
4.1.5	Phase II Errors	4-2
4.2	Execution Errors	4-3
4.2.1	Introduction	4-3
4.2.2	GO TO Action	4-3
4.2.3	SYSTEM Action	4-4
4.2.4	IGNORE Action	4-5

4.2.5	The Error Summary	4-5
4.3	Execution Dumps and Traces	4-6
4.3.1	Direct Execution	4-6
4.3.2	Execution Monitoring System	4-6
4.3.3	Location of Diagnostic Dump	4-9
5.	HAL/S REALTIME PROGRAMS	
5.1	Using the Real-Time Features of HAL/S-360	5-1
5.1.1	Introduction	5-1
5.1.2	Terms and Concepts	5-1
5.1.3	Timing	5-4
5.1.4	PARM Field Options	5-5
5.1.5	Compile Time	5-5
5.1.6	Execution Time	5-5
5.1.7	List of Real-Time Messages	5-6
5.1.8	List of Real-Time Wait-Types	5-8
5.2	HAL/S Load Module and Operating Environment	5-9
5.3	Processes and the Stack Mechanism	5-9
5.4	Procedures and the Procedure Caller	5-10
5.4.1	Calling	5-10
5.4.2	Exiting	5-14
5.5	Intrinsics	5-14
5.6	User-Written Assembly Language Subroutines	5-15
5.6.1	HMAIN	5-15
5.6.2	HENTRY	5-16
5.6.3	HCALL	5-16

5.6.4	HEXIT	5-16
5.6.5	HERRMSG	5-16
5.6.6	HERROR	5-17
6.	HAL/S CHARACTERISTICS SPECIFIC TO THE 360	
6.1	Introduction	6-1
6.2	Compile Time Characteristics	6-1
6.2.1	Character Set	6-1
6.2.2	Internal Table Capacities	6-1
6.2.3	Data Type Size Limitations	6-1
6.2.4	Program Organization Limits	6-2
6.2.5	Input/Output Statements	6-2
6.2.6	Program Naming Convention	6-3
6.2.7	The INCLUDE Compiler Directive	6-5
6.2.8	ACCESS Rights Implementation	6-6
6.2.9	Template Generation	6-9
7.	HAL/S-360 Input/Output Operations	
7.1	FILE I/O	7-1
7.2	File Type Characteristics	7-1
7.2.1	Type I: Dense Fixed-length Blocks	7-2
7.2.2	Type II: Sparse Fixed-length Blocks with Keys	7-3
7.2.3	Type III: Sparse Differing-length Blocks with Keys	7-4
7.3	File Type Selection	7-4
7.3.1	Choosing a File Type	7-4
7.3.2	Specifying a Chosen Type	7-6
7.3.3	DCB Parameters by File Type	7-6
7.3.4	Summary of DCB Errors	7-7
7.4	Run Time Errors	7-8

7<

7.5	HAL Data Type Considerations	7-10
7.6	OS/360 Considerations	7-10
7.6.1	DD Cards	7-10
7.6.2	Separate Initialization of a File	7-11
7.7	360 Execution Time Characteristics	7-12
7.7.1	Input/Output	7-12
7.8	User-Defined Execution Time JCL	7-13
7.8.1	General Rules Used by HAL to Create DCB Attributes	7-13
7.8.2	General Rules Governing HAL/S Sequential Output	7-14
7.8.3	General Rules Governing HAL/S Sequential Input	7-14

APPENDICES

- A. COMPILE-TIME JCL OPTIONS
- B. EXECUTION-TIME JCL OPTIONS
- C. PROTOTYPE CATALOGUE PROCEDURES
- D. COMPILE TIME ERROR MESSAGES
- E. EXECUTION-TIME ERRORS
- F. USER ABEND CODES
- G. NAMES RESERVED FOR HAL/S PROGRAMS
- H. COMPILER DIRECTIVES
- I. THE HALLINK PROGRAM
- J. BLOCK LOCATION AND SEARCH ALGORITHMS FOR FILE TYPES II AND III

1. INTRODUCTION

1.1 Purpose of This Manual

The purpose of this manual is to provide the information needed by programmers to compile and execute a HAL/S program. It also provides a detailed discussion of the printed matter that will be produced as a result of the compilation and execution of a HAL/S program. This manual is not intended as a guide to the HAL/S language. It is a reference document to be used in the process of getting HAL/S programs compiled and debugged on the IBM/360. A knowledge of the HAL/S language syntax and programming techniques is presumed in some of the discussions.

1.2 Scope of This Manual

The succeeding sections of this document present a system guide for all phases in the development of a successful HAL program. Topics range from operating system communication to interpretation of debugging aids. A final section presents features of the HAL programming system that have specific System/360 dependencies.

2. RUNNING A HAL/S PROGRAM

2.1 Communication with OS/360 - Job Control Language

2.1.1 Introduction

All communication between the programmer and the operating system of the host computer must be done through Job Control Language (JCL). In this section, the basic JCL that must be provided to invoke HAL/S will be presented. A detailed discussion of JCL is not attempted. The intent is to give first-time and average users sufficient information to begin running. A more detailed description of the HAL/S JCL is available in Appendix C. That description is written for persons experienced in handling JCL and therefore does not "teach" the use of JCL.

2.1.2 The Catalogued Procedure

Because JCL is a complex language, the operating system (OS/360) allows for the grouping and saving of whole blocks of JCL. Such a saved block of JCL is known as a catalogued procedure. When this facility is used, the programmer need only submit a minimum of his own JCL to make a run. The descriptions that follow presume the existence of a catalogued procedure that will compile, load, and execute a HAL/S program. A listing of a prototype catalogued procedure (HALSCLG) is presented in Appendix C. Any JCL modifications that are desired may be made in the standard manner described in the IBM publication:

IBM System/360 Operating System:
Job Control Language Reference
Order #GC28-6704

The user calls in the catalogued procedure by referencing it by name on an EXEC card as follows:

```
//ANYNAME EXEC HALSCLG
```

This card is sufficient to call in the catalogued JCL and begin execution of the steps in the compilation process. If no other information is supplied on this EXEC card, all options available to the programmer for specification will

default to that set of options saved in the catalogued procedure. The user may change or add to these options by specifying more information on the EXEC card.

2.1.3 The Optional Parameters: OPTION and RUNPARM

The HAL/S compiler has various options that the user may specify through JCL. These options invoke functions in both the compilation and execution steps of a HAL/S job. The catalogued procedure HALSCLG allows the specification of these options via the keyword parameters OPTION or RUNPARM. These parameters are coded on the EXEC card as follows:

```
//ANYNAME EXEC HALSCLG,OPTION='??',RUNPARM='??'
```

The OPTION parameter is put into the PARM field of the compilation step and is available to the compiler for interpretation and action (valid options are listed in appendix A). The RUNPARM parameter is similarly made available to the HAL/S execution-time monitoring system.

2.1.4 Specifying the Source Language Input

The user must identify, through JCL, the location of the source program that he wants compiled. The typical input is from punched cards. The compiler reads the source input from the DD card named SYSIN. This card is not supplied in the catalogued procedure because the user must do the specification. For card input, the specification would be:

```
//HAL.SYSIN DD *  
.  
.  
.  
Source cards  
.  
.  
.  
/*
```

122<

where the * on the DD card indicates input to follow. When source images are saved on some other medium, the HAL.SYSIN DD card must still be included, and the specification on the card must correctly identify the source file. Refer to the IBM JCL manuals for the techniques needed.

2.1.5 Specifying the Standard Execution-Time Input

The catalogued procedure makes the assumption that the primary data input to the running HAL/S program will be made via sequential input file #5. This means that the catalogued procedure supplies a DD card with the name CHANNEL5. The use of the HAL/S statement:

```
READ(5) <specification list>;
```

causes data to be read from the data set defined by the CHANNEL5 DD card. The catalogued procedure is organized to associate the following JCL cards with CHANNEL5.

```
//GO.SYSIN DD *  
.  
.  
Data cards  
.  
.  
/*
```

This DD card may alternately be defined in any suitable manner to reflect the location of the desired input data.

2.1.6 Specifying the Standard Execution Time Output

The catalogued procedure provides the necessary JCL to direct the results of the following HAL/S statement to a line printer:

```
WRITE(6) <specification list>;
```

The JCL statement responsible is the

```
//CHANNEL6 DD ...
```

statement.

2.1.7 Specifying Additional Execution-Time JCL

The HAL/S programmer may reference 10 separate sequential files with HAL/S I/O statements. In HAL/S statements of the form:

```
READ
WRITE      (n)    <specification listing>
READALL
```

where n may vary from 0 through 9. These statements cause the requested I/O operation to occur on data defined by JCL cards of the following form:

```
//CHANNELn DD <appropriate specification>
```

where n is the same as in the HAL/S I/O statement. Cards of this form may be added to the JCL brought in from the catalogued procedure by following the rules described in the IBM JCL publications.

2.1.8 A Typical Run Submission

The following JCL is an example of a typical user run. The user has his HAL/S program and his execution-time data on punched cards. In addition, his program contains a WRITE(7) ... statement that he wishes to direct to a card punch.

```
1 //ANYNAME1 JOB <installation dependent parameters>
2 //TRYHAL EXEC HALSCLG,OPTION='LISTING2'
3 //HAL.SYSIN DD *
  .
  .
  .
4 Source Program
  .
  .
  .
5 /*
6 //GO.CHANNEL7 DD SYSOUT=B
7 //GO.SYSIN DD *
  .
  .
  .
8 Data Cards
  .
  .
  .
9 /*
```

Comments on individual lines in the example:

- 1: The user must supply a JOB card to identify himself to the operating system and to give pertinent accounting information. The form of this card is installation dependent. Questions regarding its form should be directed to the installation operations staff.
- 2: This, the EXEC card, causes the JCL saved in catalogued procedure HALSCLG to be read by the operating system. The label TRYHAL is optional and if included may be any 1 to 8 character name beginning with a letter. If omitted, at least one blank must separate the // and the word EXEC. Following the name of the catalogued procedure, the user has coded some optional parameters as specified in Section 2.1.3 of this document. The OPTION keyword shown causes the string 'LISTING2' to be available to the compiler. The compiler recognizes this as a directive to produce an auxiliary source listing (See Sec. 3.8).
- 3: This card identifies the primary compiler input as cards immediately following.
- 4: The source cards follow.
- 5: The /* delineates the end of the in-line source cards and indicates a return to JCL card processing.
- 6: The CHANNEL7 DD card defines the destination of the HAL/S program's references to device #7 as system output class B (SYSOUT=B). At a typical 360 installation, this class refers to the card punch.
- 7: GO.SYSIN defines the input data set associated with HAL/S references to device #5.

Note: Because of the way the HALSCLG catalogued procedure is written, this could also have been specified as:

```
//GO.CHANNEL5 DD *
```

- 8: The data cards to be read from channel 5 come next.

9: The /* terminates the input card data and indicates a return to JCL processing. No special end-of-job JCL indication is needed. The operating system will determine the job boundaries by the occurrence of subsequent JOB cards.

16<

2-6

2.2 Compiler Outputs

2.2.1 Source Listing

As a result of the compilation process, a listing of the user-supplied source code is printed by the compiler. This primary listing has been formatted by the compiler to conform with standard output rules. The primary listing is always produced and is written to the data set defined by the SYSPRINT card.

The HAL/S compiler operates as three separate phases: Phase I is the syntax analysis phase; Phase II is the code generation phase; and Phase III is the diagnostic table generation phase. Each phase produces some informational and diagnostic output which together make up the primary source listing.

An optional unformatted listing is available. The user must specify the LISTING2 option in the OPTION field of the EXEC card which invokes the HAL/S catalogued procedure.

The formats of both of these listings are discussed in Section 3.

2.2.2 Tables

In addition to reproducing the HAL/S language source code, the compiler also prints various tables that contain information of interest to the programmer. The tables include the Symbol Table & Cross Reference Table, giving name, type and usage information of identifiers, and the Macro Table giving a summary of replaced names. See Sections 3.6 and 3.7 for descriptions of these tables.

2.2.3 Summaries

The HAL/S compiler produces summaries of programmer actions taken within a particular program block at the close of that block and a quick-reference program layout description at the end of the compiled program. See Sections 3.3 and 3.4.

2.2.4 Diagnostics

The compiler produces error messages when syntax errors or other abnormal conditions occur. These error messages are interlisted with the source listing. An error summary is provided at the end of the Phase I listing. Section 4 discusses compile time diagnostics in detail.

2.3 Subsequent Steps

2.3.1 Link Step

After an object file has been produced as described above, it must be further processed into a form suitable for loading and execution. This process includes the resolution of any references to HAL/S library routines and the generation of appropriately-sized work areas required by the HAL/S programs at run time. These tasks are accomplished in the second step of HALSCLG. This step invokes a program known as HALLINK, a HAL/S compiler system program which performs all necessary functions. The HALLINK program dynamically invokes the System/360 Linkage Editor as part of its operation.

The printed matter generated by this step in the HALSCLG procedure appears in three parts:

- 1) A standard output produced by the Linkage Editor which may consist of a module map and size statistics. Descriptions of this listing may be found in the appropriate IBM system manual:

IBM System/360 Operating System
Linkage Editor and Loader
Form GC28-6538

- 2) A HALLINK listing which documents the tree structure of all HAL/S modules involved in the link edit.
- 3) A second standard linkage editor listing as described above. This listing will incorporate changes made to the module structure by the HALLINK program. This second link editor listing is the one corresponding to the final load module produced by this step.

The HALLINK step puts its final result on a direct access device suitable for subsequent loading and execution. The load module thus produced requires a third step to be executed. A more thorough description of the HALLINK program may be found in Appendix I.

2.3.2 The Execution Step

The execution of a compiled HAL/S program may produce both user-defined output and system diagnostic output. The user output occurs as a result of HAL/S I/O statements. The system diagnostic output can occur as a result of execution errors detected by the system or as a result of user requests for dynamic dumps and traces.

2.4 Creating and Running Program Complexes

2.4.1 Introduction

Section 2.1 has explained how to run a self-contained HAL/S program. However, the form of the language allows a HAL/S program to use data external to itself (COMPOOLS), and to call external procedures or functions (COMSUBS). A HAL/S program and the compools and comsubs it uses are collectively known as a PROGRAM COMPLEX. This section explains how to create and run a program complex.

2.4.2 The Form of Compools and Comsubs

This subsection briefly recapitulates the forms taken by compools and comsubs in the HAL/S language. Both compools and comsubs are treated by the HAL/S compiler as independently compilable entities in the same way as a program.

The form of a compool is illustrated by the following example:

```
DATA: COMPOOL;  
  DECLARE S SCALAR INITIAL(2.5);  
  DECLARE I INTEGER INITIAL(5);  
CLOSE DATA;
```

Fig. 2.1

The form of a typical comsub is illustrated by the following example:

```
ROUTINE: PROCEDURE(X);
        DECLARE X SCALAR;
        WRITE(6) X;
CLOSE ROUTINE;
```

Fig. 2.2

A program using data in the compool DATA and calling the comsub ROUTINE must contain the appropriate matching templates for them. Such a program is illustrated by the following example:

```
DATA: EXTERNAL COMPOOL;
      DECLARE S SCALAR INITIAL(2.5);      compool
      DECLARE I INTEGER INITIAL(5);      template
CLOSE DATA;

ROUTINE: EXTERNAL PROCEDURE(X);      comsub
        DECLARE X SCALAR;              template
CLOSE ROUTINE;

TEST: PROGRAM;                          program
      CALL ROUTINE(I/S);                proper
CLOSE TEST;
```

Fig. 2.3

The HAL/S language of course also allows comsubs themselves to access compool data and/or other comsubs. Templates, like those shown in the example above, are produced automatically by the HAL/S-360 compiler when a compilation unit requiring a template is compiled. The template produced is written as a member of a partitioned dataset which may then be INCLUDE'd at the appropriate place in another compilation which is to make use of the template. See Section 6.2 for the technical details of the HAL/S-360 implementation of templates.

2.4.3 Compiling a Program Complex

To compile a program complex, the program module and the compool and comsub modules must each be compiled separately. Each module is compiled in exactly the same way. However, due to a requirement dictating that a template (as described in Section 2.4.2) be included in a compilation referencing a previous compilation, the compools and comsubs themselves should be compiled in an order such that their templates are available when needed.

For each compilation, a catalogued procedure should be used which incorporates JCL enabling the object module (or optionally a load module form of the object module) to be saved until all modules are compiled; e.g. the HALSC or HALSCL catalogued procedures in Appendix C.

When compiling the individual pieces of a program complex, any templates produced should be directed via JCL to a common template library which can then be used as an INCLUDE library for referencing the templates.

When all compilations have been completed, the individual modules are linked together with the runtime library by using the HALLINK program (e.g. via the HALSL catalogued procedure listed in Appendix C). The resulting finished load module may then be executed as previously described for a simple program.

Each compilation also produces symbolic data used as a run-time debugging aid. In compiling and executing simple programs using the catalogued procedure HALSCLG, this symbolic data is written on a member of a temporary PDS passed to the load step of the JCL. Here are two methods for insuring that the symbolic data for all modules of a program complex are correctly made available at execution time:

- (i) The symbolic data for each module is saved on a different member of a PDS common to all compilations. In the execution step this dataset should be specified on the HALSYMB DD card. (See catalogued procedure in Appendix C.)
- (ii) The symbolic data for each module is saved on a different PDS (each of which will therefore only have one member). In the execution step, the HALSYMB DD cards should specify the catenation of all the PDS's used.

2.4.4 An Example of Program Complex Compilation

To illustrate the procedure for the creation of an executable program complex, the following example is provided. Let there be four separately compilable units, as follows:

Unit 1 - A COMPOOL named DATA defined as:

```
DATA: COMPOOL;
  DECLARE I INTEGER;
  DECLARE S SCALAR;
CLOSE DATA;
```

Unit 2 - A PROCEDURE named PROC1 defined as:

```
D INCLUDE @DATA
  PROC1: PROCEDURE(X);
  DECLARE X SCALAR;
  WRITE (6) 'THE ANSWER IS:' (X+S);
  RETURN;
CLOSE PROC1;
```

Unit 3 - A PROGRAM named PROG1 defined as:

```
D INCLUDE @DATA
D INCLUDE @PROC1
  PROG1: PROGRAM;
  CALL PROC1(S);
CLOSE PROG1;
```

Unit 4 - A PROGRAM named DRIVER defined as:

```
D INCLUDE @DATA
D INCLUDE @PROC1
D INCLUDE @PROG1
  DRIVER: PROGRAM;
```

22<

```

CALL PROC1(S);
SCHEDULE PROG1;
CLOSE DRIVER;

```

Note the relationships between the various pieces of this program complex. Units 2, 3, and 4 all reference data in Unit 1. Units 3 and 4 reference Unit 2. Unit 4 also references Unit 3. The HAL/S-360 Compiler must be given information about these separately compiled units whenever references to such units occur. The necessary information is generated automatically when each unit is compiled. It is, therefore, necessary to compile the units in an order which makes the information in a given unit available when needed by subsequent compilations. An example of a single job to perform these compilations and execute the resultant module is presented next. Comments on the example follow.

```

//TRYIT JOB <PARAMETERS>
//STEP1 EXEC HALSC
//HAL.SYSIN DD *
    <UNIT #1 SOURCE>
/*
//STEP2 EXEC HALSC
//HAL.SYSIN DD *
    <UNIT #2 SOURCE>
/*
//STEP3 EXEC HALSC
//HAL.SYSIN DD *
    <UNIT #3 SOURCE>
/*
// STEP4 EXEC HALSCLD, RUNPARM='FIRSTPGM=DRIVER'
//HAL.SYSIN DD *
    <UNIT #4 SOURCE>
/*

```

STEP1 above compiles the COMPOOL named DATA. Because of the way the HALSC catalogued procedure is defined, the object file for the compilation is placed on a temporary file which is passed on to the later steps. This compilation also produces a template for the COMPOOL as described in Sections 2.4.2 and 6.2.9. The template is also saved in a temporary data set and passed to later steps. In a similar way, a simulation data file (SDF) for the COMPOOL is produced and passed on.

Step 2 compiles the PROCEDURE named PROC1. Note that the source code for the compilation contains the following include compiler directive:

```

D INCLUDE @DATA

```


This directive is a request for the compiler to locate and read the template ("@" indicates a template name) for the COMPOOL named DATA. The template must be available for the successful compilation of statements referencing items in that compool. The compiler finds and uses the template that was automatically generated in STEP1. (see Section 6.2.9 for more details on template look-up.) Step 2 adds the object file for the PROCEDURE named PROC1 to the object file passed from step 1. It also produces a template for PROC1 and an SDF and adds them to the existing files. These files are then passed to later steps.

Step 3 performs in a manner similar to step 2, this time compiling the PROGRAM named PROG1. This compilation requires access to both of the previously compiled units. The source of the compilation therefore contains INCLUDE directives to cause the compiler to retrieve the necessary information. Again, object, template, and SDF files are produced and passed along.

Step 4 invokes the HALSCLG catalogued procedure because this step will perform the last of the compilations, and then execute the entire complex. The source code for the PROGRAM named DRIVER contains the INCLUDE directives needed to retrieve the templates for the previous compilations that are to be referenced by DRIVER. The HALSCLG catalogued procedure adds the object deck for DRIVER to the others, produces an SDF, and also produces a template for DRIVER. Production of the template in this case is unimportant because no subsequent steps will need the template. HALSCLG then proceeds to link the object decks for all four compilations together with the necessary library routines to form a module suitable for execution. The last function of HALSCLG is to initiate execution of the module. Note that there is some ambiguity inherent in this module as to which of the two PROGRAM's in the module is to begin execution. If no statement to the contrary is made, execution defaults to the first PROGRAM unit in the module which in this case would be PROG1. The intention, however, is to begin execution with DRIVER which will in turn invoke PROG1. It was the hierarchical reference requirements which caused the compilation of the units in the order illustrated. The way in which the correct PROGRAM may be specified in a multiple-PROGRAM complex is shown in step 4 JCL. The EXEC card for step 4 contains the aprameter:

```
RUNPARM='FIRSTPGM=DRIVER'
```

This specification causes the quoted string to be available to the execution step of the HALSCLG catalogued procedure. The FIRSTPGM option (see Appendix B) specifies the full name of the PROGRAM which is to begin execution in the load module.

During execution of the program complex, the SDF files created and passed by the previous steps are available to the runtime package of the executing module. If any errors occur,

or if any user requests for debugging diagnostics were entered, these SDF files would permit accurate dumps and traces of HAL data items to be produced.

The complex would be executed as directed and after execution, all temporary data sets used to collect the object files, templates, and SDF's would be deleted by the operating system.

This example shows only one method for building an executable module. It is, perhaps, the simplest because it presumes that all information passing is done via temporary datasets built automatically by the catalogued procedures. Users wishing to save the results of individual compilations may define permanent template, SDF, and load module libraries to be used in place of the temporary ones. The operations which must be performed to eventually create an executable module remain the same.

3. COMPILATION LISTINGS

3.1 General Description

The listings produced by the HAL/S compiler are designed to document the actions taken by the compiler in the generation of an executable form of the user's source program. The user's code is reproduced in an annotated form and, optionally, in its original form. All tables and error messages generated by the compiler are also considered part of the documentation. They are described in the following sections.

3.2 Formats

The numbered notes in the following discussion refer to an example of a HAL/S source listing shown in Figure 3.1.

3.2.1 Listing of Options in Effect

On the first page of a compilation listing, the HAL/S-360 compiler prints a summary of all compile-time options in effect for the compilation (27). See Appendix A for a list of available options.

3.2.2 Headings

A one-line page header (1) begins every page of the listing. It contains compiler version identification and page number within the listing.

On page one of the listing, the date and time of generation of the compiler are printed, followed by the date and time at which the current compilation was begun (2).

Following any header information on each page of the listing, a field description line is printed (3). This line breaks the page into columns, the contents of which are described below.

3.2.3 Statement Number

The statement number field (4) headed by the title "STMT" contains the compiler-assigned sequence number for

each HAL/S statement. This field is filled in for each M-line in the source listing. The "STMT" field for E-line and S-line entries as well as for comment cards and compiler directives is left blank.

Note that the statement number is associated with a complete HAL/S statement, not with the physical number of M-lines. Thus, if a HAL/S statement spans several M-lines, the same statement number will appear on each M-line.

3.2.4 Line Type

The STMT field is followed by a blank and then by a single character field (5) used to indicate the type of source line. HAL/S has multi-line subscripting and exponentiation capabilities. Such multiple line use is identified for easier reading.

The compiler places an indicator of line type in this one character field. The possible values of this field are:

- C = Comment line;
- D = Compiler directive line;
- E = Exponent line;
- S = Subscript line;
- M = Main line.

These values correspond generally to the card types (punched in column 1) of the user's source cards.

3.2.5 Source Field

The next field on the page (6) is centered under the title "SOURCE". This field contains the actual HAL/S language text. Vertical bars at either side delimit the field. The field itself is 100 characters wide and is filled by reformatted source text, complete with compiler-supplied annotation.

3.2.6 Current Scope Field

Following the source delimiter (|) at the right of the SOURCE field, is the variable length field headed by "CURRENT SCOPE" (7). This is an information field which contains the name of the HAL/S program block to which the current source line belongs. This field applies only to C, D, and M lines.

3.2.7 Information Field

To the right of the "CURRENT SCOPE" field is another variable length area that is used by the compiler to supply additional information. This information may be any compiler generated comments regarding the current line. This field is applicable only to M-lines.

3.3 The Output Writer

3.3.1 Concept

The HAL/S compiler has been designed to provide standard, automatic annotation of its output listing to enhance the readability of HAL/S source code. The HAL/S system allows each programmer to enter programs in a free-form input consistent with individual coding preferences. The compiler then edits the input during compilation into the standard form so that all program listings will observe the same coding rules.

HAL/S is a block-oriented language, and the logical indenting of program blocks can do a great deal to enhance understanding of program structure. The programmer can do this indenting himself. But the problems involved in inserting new indentation levels into existing code often result in considerable wasted time because it is necessary to re-punch existing lines to maintain consistency. HAL/S frees the programmer from this task by completely regenerating the indentation scheme each time the program is compiled. Thus, the indentation is always complete and reflects the total program structure.

Although HAL/S source input is in the form of card images, the compiler treats the input as a continuous stream of information, with only the statement-delimiting semicolons to indicate statement boundaries. Each statement is stored internally until its semicolon is found. Then, with a complete statement in hand, the HAL/S output writer completely reformats the source. The reformatting includes referencing the symbol table to obtain the types of any variables in the statement so that the characteristic HAL/S overpunch mark may be supplied by the compiler. The reformatting also includes expansion of single line input to the full HAL/S multi-line form. Finally, the resulting multi-line, annotated statement is indented to the proper level determined by the line's relative position in the program. The statement is then placed on the output listing, using as many E-M-S groups as necessary to contain it.

The specific conventions imposed on the output listing are detailed in the following sections.

3.3.2 Auto-Indentation

References in this section are made to Figure 3.1.

3.3.2.1 Declaration Statements. The output writer prints out declaration statements in a way which makes the intent of the declaration as clear as possible. The word DECLARE is aligned at the current indent-level. If the DECLARE has factored attributes, the attributes are placed on the same line as the DECLARE and a new line begun (8). If no factors are present, variable names follow the DECLARE (9). Lists of variables without individual attributes are placed on the same line (10). The occurrence of a variable with attributes causes that variable to appear on a line by itself with its attributes (11). Any lines created after the DECLARE line are indented one indent level.

Structure declarations (12) are reformatted into the commonly used form. Each level of the structure template is placed on a separate line with indenting appropriate to the level number.

3.3.2.2 Labels. All statement labels (13) are right justified against the statement to which they apply. The statement itself is placed at the proper indent level before the label is applied. If the label will not fit on the same line as the statement body because of the indent location, it is placed on a separate M-line preceeding the statement body (14).

3.3.2.3 Scope Changes. Whenever a PROGRAM, PROCEDURE, TASK, FUNCTION, or UPDATE block is encountered, the statement is placed in the output listing at the current level and then the indent level is increased one increment (15). All statements within the block follow the normal indenting rules relative to the block level indentation. Thus, all statements within a block are indented farther than the block definition.

When the corresponding CLOSE statement is found, the CLOSE statement is output at the same level as the block definition statement and the indent level is reset to its value before the block was entered

(16).

3.3.2.4 IF Statements. The IF ... THEN part of the statement is placed on the listing at the current indent level and the indent level is increased one increment. The "true part" of the statement is placed on the next level at the new indent level and the level is decremented one level (17).

If an ELSE clause is present, the "ELSE" gets a new line at the current level, which is the same level as the "IF", and the indent level is incremented. The "false part" of the statement is placed on the next line at the new level and then the level is decremented one level (18).

3.3.2.5 DO Groups. All types of DO groups receive the same treatment. The statement containing the DO is placed in the listing at the current indent level and the level is incremented (19). All statements in the range of the DO are indented relative to this new level. The END closing the group is placed at the same level as the DO (20).

The DO CASE statement obeys the same indent rules as other DO statements, but some additional notation is supplied by the output writer. The first M-line of each case is annotated in the information field beyond the current scope notation with a message of the form "CASE n" where n is the current case number (21). If the current case is really a "case within a case", (i.e., a nested DO CASE is in effect), the notation is "CASE a.b ... n" where the a.b ... indicates the structure of the case statements in the sense of: case n within case b within case a ... Also, the END associated with the DO CASE statement receives the additional information "DO CASE END" to help associate it with its group head (22).

3.3.2.6 Continuations. A reformatted E-M-S group may not fit on the printed page in a single group after the indentation rules have been applied. If this is the case, the output writer breaks the statement into as many E-M-S groups as necessary (23). The break never splits an identifier or keyword. A literal character string may be broken.

3.3.2.7 Page Boundaries. The output writer never

places pieces of an E-M-S group on separate pages; i.e. an E-line at the bottom of one page and the corresponding M-line on the next. A page eject will always be performed before such a group is written.

3.3.2.8 The first line of any PROCEDURE, FUNCTION, PROGRAM, TASK, or UPDATE block always begins on a new page.

3.3.3 Multi-line Expansion and Annotation

References are made to Figure 3.1.

3.3.3.1 Overpunches. If, after all pertinent subscripting has been applied, a variable name is of a type for which HAL/S has defined an overpunch character, the output writer supplies that character on an E-line above the variable name. The overpunch character is centered over the name.

The characters available are "*", "-", ".", ",", "+", for matrix, vector, bit, character, and structure data types respectively. The mark supplied is determined from the totally subscripted form of the variable. Thus the overpunch may be changed by subscripting. For example, an element of a matrix is a scalar. Therefore, a matrix name subscripted down to a particular element receives no overpunch (24). Similarly, a matrix variable subscripted to a particular row of the matrix receives a vector mark (25).

3.3.3.2 Array and Structure Notation. Variables which are structure terminals or which are arrayed may have additional annotation supplied. If a particular use of such a variable has multiple copies due to structure and/or array properties and if those multiple copies have not been subscripted away in the particular use of the variable, the variable is enclosed in the appropriate marks. Multiple copies due to arrayness receive brackets ("[" and "]") while multiple copies due to structure copies receive enclosing braces ("{" and "}"). (26)

As with variable typing (see Sec.3.3.3.1), it is possible to subscript away the "arrayness" or "structureness" of a variable and thus have no

special annotation appear.

Note that the array and structure notation characters are available only on certain print trains such as the IBM TN chain. If a particular installation does not have such a print chain, the notation will probably appear as blanks (unprintable characters).

3.3.3.3 Subscripts and Exponents. HAL/S allows the user to supply source statements in single or multi-line format. The output writer expands all source to full multi-line format before printing.

During the expansion process, any unnecessary subscript or exponent grouping parentheses are removed. These grouping parentheses are often needed in single line input to show the extent of a subscript or exponent field.

Subscripts applied to variables on an exponent line, and exponents applied to variables on a subscript line, are left in single line format since multiple line expansion would produce an ambiguous listing. Also, overpunch characters are not supplied for variables on exponent or subscript lines.

The multiple lines are indicated by "E" and "S" in the line type field of the listing (see 3.2.3). As many E or S lines as needed to contain the expanded source are generated.

3.3.3.4 REPLACE'd Symbols. Any symbols which are defined as replaced names in REPLACE statements are underlined by the output writer in the source listing (27).

3.3.4 Comments

3.3.4.1 Comment Cards and Directive Cards. All comment cards (C in column one) and directive cards (D in column 1) are transferred unchanged to the output listing. Both types of card are treated alike, being separated by a blank line from other lines, and single spaced within the group.

3.3.4.2 In-Line Comments. Comments appearing on the M-lines of input source cards in the form of
/* ... Comment ... */

are collected by the output writer and placed into the output listing after the statement has been processed.

The collected comments are placed on the M-line of the statement if possible. If there is not room, spill-over can occur onto as many S-lines as are necessary. Multiple comments on a single source statement are collected together and printed as one comment. If the size of the total comment text for any one HAL/S statement reaches 256, additional comment text is ignored and a warning issued.

HAL/S COMPILER PHASE 1 -- VERSION 7.0 OF MARCH 29, 1974. CLOCK TIME = 12:56:33.73.

② TODAY IS MARCH 30, 1974. CLOCK TIME = 15:25:47.95.

OVERRIDING OPTIONS: LISTING2,LIST

COMPLETE LIST OF COMPILE-TIME OPTIONS IN EFFECT

TYPE 1 OPTIONS ②⑦

NODUMP	INSTEAD OF DUMP
LISTING2	INSTEAD OF NOLISTING2
LIST	INSTEAD OF NOLIST
TRACE	INSTEAD OF NOTRACE
NODECK	INSTEAD OF DECK
TABLES	INSTEAD OF NOTABLES
NOTABLST	INSTEAD OF TABLST
NOADDRS	INSTEAD OF ADDR
NOSRN	INSTEAD OF SRN
NOSDL	INSTEAD OF SDL

TYPE 2 OPTIONS

PAGES = 250
LINE COUNT = 59

Figure 3-1 The HAL/S SOURCE LISTING

STMT	SOURCE	CURRENT SCOPE
4	6	7
1 MI DEMO:		DEMO
1 MI PROGRAM;		DEMO
CI		DEMO
CI THIS IS A DEMONSTRATION PROGRAM TO SHOW THE LISTING PRODUCED BY		DEMO
CI THE HAL/S-360 COMPILER		DEMO
CI		DEMO
2 MI REPLACE PRINTER BY "6";		DEMO
3 MI DECLARE INTEGER INITIAL (1), 8		DEMO
3 MI A, B, C;		DEMO
4 MI DECLARE D, F, G; 9		DEMO
5 MI DECLARE E VECTOR(4);		DEMO
6 MI DECLARE H, I, J, 10		DEMO
6 MI K ARRAY(5) MATRIX(3, 4), 11		DEMO
6 MI L, M, N,		DEMO
6 MI O SCALAR;		DEMO
7 MI STRUCTURE AA: 12		DEMO
7 MI 1 BB,		DEMO
7 MI 2 CC MATRIX(4, 3),		DEMO
7 MI 1 DD,		DEMO
7 MI 2 EE ARRAY(4) MATRIX(3, 4);		DEMO
8 MI STRUCTURE QQ:		DEMO
8 MI 1 RR,		DEMO
8 MI 2 AAREF AA-STRUCTURE,		DEMO
8 MI 2 SS CHARACTER(5);		DEMO
9 MI DECLARE MY_STRUCTURE QQ-STRUCTURE(5);		DEMO

2
ET
A

Figure 3-1(con't.)

STMT	SOURCE	CURRENT SCOPE
10 MI	PROC1: (14)	PROC1
10 MI	PROCEDURE;	PROC1
11 MI	(15) DECLARE A INTEGER;	PROC1
12 MI	IF A = B THEN (17)	PROC1
13 MI	DO: (19)	PROC1
14 MI	(13) LABEL1: B = C;	PROC1
15 MI	(26) [MY_STRUCTURE.RR.AAREP.DD.EE] = [MY_STRUCTURE.BR.AAREP.BB.CC] ; SI *;3:2,* *;*,2	PROC1
16 MI	(20) END;	PROC1
17 MI	ELSE (18)	PROC1
17 MI	A = C;	PROC1
18 MI	(16) CLOSE PROC1;	PROC1

6.3
A

B L O C K S U M M A R Y

OUTER VARIABLES USED:

B, B*, C, MY_STRUCTURE*, MY_STRUCTURE

Figure 3-1 (con't.)

STMT	SOURCE	CURRENT SCOPE
19 M	DO FOR C = 1 TO 100;	DEMO
20 M	D = K 24 ;	DEMO
SI	C:2,3	
21 M	END;	DEMO
22 M	DO CASE A:	DEMO
23 M	A = B;	DEMO CASE 1 (21)
E	- -	
24 M	E = K (25) ;	DEMO CASE 2
SI	B:A,*	
25 M	END;	DEMO DO CASE END (22)
E	*	
26 M	[K] = 0;	DEMO
E	(27)	
27 M	WRITE (PRINTER) MY_STRUCTURE.RR.AAREF.BB.CC	DEMO
SI	3;1 TO 3,* [MY_STRUCTURE.RR.AAREF.DD.EE] 2;2 TO 4: D,	
E	-	
27 M	(23) E, F, G, H, I, J, [K], L, M, N, O;	DEMO
E	*	
28 M	CLOSE DEMO;	DEMO

25
^

Figure 3-1 (con't.)

3.4 Block Summaries

3.4.1 Concept

The HAL/S compiler provides a summary of action taken within a program block at the close of the particular block. The blocks for which summaries are given are PROGRAM, TASK, FUNCTION and UPDATE. When the matching CLOSE to such a block is found, the summary is issued and the listing of the program resumes with a skip to the new page.

3.4.2 Information Provided

Information contained in block summaries consists of lists of labels or variable names used in various contexts within the block. The title "BLOCK SUMMARY" begins the list. For all potentially summarized contexts within the block, a descriptive heading is printed followed by the list of names involved. The headings and their meanings are listed below.

- a) PROGRAMS AND TASKS SCHEDULED - A list of PROGRAM and/or TASK names scheduled in the current block via the HAL/S SCHEDULE statement.
- b) PROGRAMS AND TASKS TERMINATED - A list of PROGRAM and/or TASK names terminated in the current block via the HAL/S TERMINATE statement.
- c) PROGRAMS AND TASKS CANCELLED - A list of PROGRAM and/or TASK names cancelled in the current block via the HAL/S CANCEL statement.
- d) EVENTS SIGNALLED, SET, OR RESET - A list of event variables declared outside the current scope and appearing in the current block in a SIGNAL, SET, or RESET statement.
- e) EVENT VARIABLES USED - A list of event variables declared outside the current scope that appeared in one or more EVENT expressions.
- f) PROGRAM OR TASK EVENTS USED - A list of PROGRAM and/or TASK names appearing in one or more EVENT expressions.
- g) PRIORITIES UPDATED - A list of PROGRAM and/or TASK names whose priorities have been updated in the current block via the UPDATE PRIORITY statement.

- h) EXTERNAL PROCEDURES CALLED - A list of procedures called within the current block which were defined via EXTERNAL PROCEDURE templates.
- i) EXTERNAL FUNCTIONS INVOKED - A list of functions invoked within the current block which were defined via EXTERNAL FUNCTION templates.
- j) OUTER PROCEDURES CALLED - A list of PROCEDURE names which are called from within the current block, but are defined outside the current block.
- k) OUTER FUNCTIONS INVOKED - A list of FUNCTION names which are invoked from within the current block but which are defined outside the current block.
- l) ERRORS SENT - A list of the HAL/S error numbers which are sent explicitly by a SEND ERROR statement somewhere in the current block.
- m) COMPOOL VARIABLES USED - A list of identifiers which are defined in one or more COMPOOL blocks, and referenced or assigned within the current block (a * beside the name indicates "assigned into").
- n) COMPOOL REPLACE DEFINITIONS USED - A list of REPLACE variable names which are defined in a COMPOOL block and used within the current block.
- o) COMPOOL STRUCTURE TEMPLATES USED - A list of STRUCTURE template names which are defined in a COMPOOL block and used within the current block.
- p) OUTER VARIABLES USED - A list of identifiers which are defined outside the current block, but are referenced or assigned within the current block (a * beside the name indicates "assigned into").
- q) OUTER REPLACE DEFINITIONS USED - A list of REPLACE names which are used within the current block, but which are defined outside of the current block.
- r) OUTER STRUCTURE TEMPLATES USED - A list of STRUCTURE which are used within the current block, but which are defined outside of the current block.

Note that in all categories except c and l, only variables which have a NEST level less than that of the current block are included, as the block summary is an indication

of the impact of this block outside of its own local variables or sub blocks. A block which is completely self-contained, no matter how complex, will not have a block summary issued.

The order of names in any list within the block summary indicates the order of occurrence of first usages of these names in the block within the identified context.

The block summary for any block will not duplicate the information in the block summary for any block nested within it (i.e. if Procedure A contains Procedure B which schedules Program C, Procedure B's block summary will indicate that Program C was called, but Procedure A's will not).

3.5 Program Layout Summary

Immediately preceding the Symbol Table printout at the CLOSE of the HAL/S program, there is a program layout map, indicating the way in which PROGRAMS, TASKS, PROCEDURES, FUNCTIONS, and UPDATE blocks were defined. The indent level in this printout indicates the nesting level definition of the block shown. This serves to give a quick overview of the program structure. Such a listing can be of assistance not only as a documentation aid, but also as a guide to locating the definition of procedures and functions which have been diagnosed as undefined by the compiler.

3.6 Symbol & Cross Reference Table Listing

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

Any structure templates defined in the compilation appear first in the symbol and cross reference table. The template names appear in alphabetical order. The body of each template (i.e. the levels defined under the template name) is listed under the template name in the order of definition. This ordering provides a quick reference to the organization of the structure template. Special action is taken in the "NAME" and "TYPE" fields (described below) to highlight the template organization.

Following any listing of the templates, an alphabetized listing of all programmer-defined symbols is printed. Symbols previously listed as elements of a structure template are included in this list. However, the list is completely alphabetized and template organization is not shown. When a

particular symbol is independently defined in more than one name scope, the symbol is multiply listed in order of definition. Figure 3.2 illustrates the form of a HAL/S compiler symbol and cross reference table.

3.6.1 "DCL" Field

The "DCL" field is used to list the compiler-assigned statement number at which the identifier was first declared. For explicit declarations, this number will point to a DECLARE statement somewhere in the program. For implicit declarations, this number is the statement at which the identifier was first used in the program.

3.6.2 "NAME" Field

The "NAME" field lists the symbolic name of the programmer-defined symbol. The width of this field is determined by the length of the longest symbol in the compilation. An asterisk preceeding the name indicates that the variable was implicitly defined.

Within the first part of the symbol table listing which contains structure template names and their organizations as described in Section 3.6, all parts of the template body have their names indented one space from the structure template name under which they are defined.

3.6.3 "TYPE" Field

The "TYPE" field describes the type of each programmer-defined identifier. This field will contain one of the following descriptions:

- INTEGER
- SCALAR
- n - VECTOR
- n X n MATRIX
- BIT(n)
- CHARACTER(n)
- EVENT
- PROGRAM
- PROCEDURE
- TASK
- COMPOOL
- STATEMENT LABEL
- UPDATE LABEL
- STRUCTURE
- STRUCTURE(n)
- MINOR NODE
- STRUCTURE TEMPLATE
- REPLACE MACRO
- MACRO ARG.

The values of n and m indicate size or dimensionality in their particular contexts.

Additional information may appear in the type field. If the identifier has an array specification, the word "ARRAY" will follow the basic type specification as in:

"INTEGER ARRAY"

If the identifier is a function name, the type specification will be followed by the word "FUNCTION" as in:

"3 - VECTOR FUNCTION"

If the identifier has the NAME attribute, the type specification will be preceded by the word "NAME" as in:

"NAME SCALAR"

Within the first part of the symbol table listing which shows the organization of structure templates, the "TYPE" field contains additional information to indicate the hierarchical relationships which exist within the structure templates. For each symbol which is part of a structure template, the "level number" of the symbol is printed in front of the type information for the symbol.

3.6.4 "ATTRIBUTES & CROSS REFERENCE" Field

This field contains all declared attributes of the symbol, lists all cross reference information, and contains comments about the identifier's use.

Attributes: This part of the field lists all data declaration attributes of variables or labels. In addition, special information about structure template elements is provided. The possible attributes which may appear are:

ARRAY(n,n,n)
SINGLE
DOUBLE
TEMPORARY
LOCKED
DENSE
ALIGNED
ASSIGN PARM

```

INPUT PARM
AUTOMATIC
STATIC
LATCHED
INITIAL
CONSTANT
ACCESS
REENTRANT
EXCLUSIVE
EXTERNAL
NONHAL
<template name>-STRUCTURE

```

Elements of structure templates have their full attributes listed in the first part of the symbol table listing where they are shown under their appropriate template name. In the alphabetized second part of the listing, these structure template elements will appear again but their attributes will not be repeated. Instead the notice:

```

*** SEE STRUCTURE TEMPLATE <template name>

```

will appear directing attention to the hierarchical template listing of the structure template whose name is given.

The attributes of a REPLACE MACRO entry will contain only the message:

```

"MACRO-TEXT INDEX=<number>"

```

which directs the reader to the appropriate entry in the Macro Text Listing for a definition of the replace text.

Cross References: The remainder of the line, following any attributes and the word "XREF:", is devoted to a list of all references to the identifier in the format:

```

N XXXX

```

where XXXX is a four digit specification of the line number of the HAL/S statement containing the identifier. N is a cross reference flag which specifies the identifiers usage:

Flag Code:	N	Use of Identifier
	0	Definition
	1	Subscript
	2	Reference
	3	Subscript and reference
	4	Assignment

P R O G R A M L A Y O U T

DEMO: PROGRAM:

PROC1: PROCEDURE:

23
^

Figure 3-2 The HAL/S Program Layout and Symbol
& Cross Reference Table Listing

SYMBOL & CROSS REFERENCE TABLE LISTING :

(CROSS REFERENCE FLAG KEY: 4 = ASSIGNMENT, 2 = REFERENCE, 1 = SUBSCRIPT USE, 0 = DEFINITION)

DCL	NAME	TYPE	ATTRIBUTES & CROSS REFERENCE
7	AA	STRUCTURE TEMPLATE	ALIGNED XREF: 0 0007 2 0008
7	BB	1 MINOR NODE	ALIGNED XREF: 0 0007 POSSIBLY NOT USED
7	CC	2 4 X 3 MATPIX	SINGLP, ALIGNED XREF: 0 0007 2 0015 2 0027 POSSIBLY NOT ASSIGNED
7	DD	1 MINOR NODE	ALIGNED XREF: 0 0007 POSSIBLY NOT USED
7	EE	2 3 X 4 MATRIX ARRAY	ARRAY(4), SINGLE, ALIGNED XREF: 0 0007 4 0015 2 0027
8	OQ	STRUCTURE TEMPLATE	ALIGNED XREF: 0 0008 2 0009
8	RR	1 MINOR NODE	ALIGNED XREF: 0 0008 POSSIBLY NOT USED
8	AAREF	2 STRUCTURE	AA-STRUCTURE, ALIGNED XREF: 0 0008 6 0015 2 0027
8	SS	2 CHARACTER(5)	ALIGNED XREF: 0 0008 POSSIBLY NOT USED
3	A	INTEGER	SINGLE, ALIGNED, STATIC, INITIAL XREF: 0 0003 2 0022 4 0023 1 0024
11	A	INTEGER	SINGLE, ALIGNED, STATIC XREF: 0 0011 2 0012 4 0017
8	AAREF	STRUCTURE	**** SEE STRUCTURE TEMPLATE OQ
3	B	INTEGER	SINGLE, ALIGNED, STATIC, INITIAL XREF: 0 0003 2 0012 4 0014 2 0023 1 0024
7	BB	MINOR NODE	**** SEE STRUCTURE TEMPLATE AA
3	C	INTEGER	SINGLE, ALIGNED, STATIC, INITIAL XREF: 0 0003 2 0014 2 0017 4 0019 1 0020
7	CC	4 X 3 MATRIX	**** SEE STRUCTURE TEMPLATE AA
4	D	SCALAR	SINGLE, ALIGNED, STATIC XREF: 0 0004 4 0020 2 0027
7	DD	MINOR NODE	**** SEE STRUCTURE TEMPLATE AA
1	DEMO	PROGRAM	XREF: 0 0001
5	E	4 - VECTOR	SINGLE, ALIGNED, STATIC XREF: 0 0005 4 0024 2 0027
7	EZ	3 X 4 MATRIX ARRAY	**** SEE STRUCTURE TEMPLATE AA
4	F	SCALAR	SINGLE, ALIGNED, STATIC XREF: 0 0004 2 0027
4	G	SCALAR	**** ERROR **** REFERENCED BUT NOT ASSIGNED
6	H	SCALAR	SINGLE, ALIGNED, STATIC XREF: 0 0004 2 0027
6	I	SCALAR	**** ERROR **** REFERENCED BUT NOT ASSIGNED
6	J	SCALAR	SINGLE, ALIGNED, STATIC XREF: 0 0006 2 0027
6	K	3 X 4 MATRIX ARRAY	**** ERROR **** REFERENCED BUT NOT ASSIGNED
6	L	SCALAR	ARRAY(5), SINGLE, ALIGNED, STATIC XREF: 0 0006 2 0020 2 0024 4 0026 2 0027
14	LABEL1	STATEMENT LABEL	SINGLE, ALIGNED, STATIC XREF: 0 0006 2 0027
6	M	SCALAR	**** ERROR **** REFERENCED BUT NOT ASSIGNED
9	MY_STRUCTURE	STRUCTURE(5)	QO-STRUCTURE, ALIGNED, STATIC XREF: 0 0009 6 0015 2 0027
6	N	SCALAR	SINGLE, ALIGNED, STATIC XREF: 0 0006 2 0027
6	O	SCALAR	**** ERROR **** REFERENCED BUT NOT ASSIGNED
2	PFINTER	REPLACE MACRO	SINGLE, ALIGNED, STATIC XREF: 0 0006 2 0027
10	PROC1	PROCEDURE	**** ERROR **** REFERENCED BUT NOT ASSIGNED
8	RR	MINOR NODE	MACRO-TEXT INDEX=1 XREF: 0 0002 2 0027
8	SS	CHARACTER(5)	XREF: 0 0010 NOT REFERENCED
			**** SEE STRUCTURE TEMPLATE OQ
			**** SEE STRUCTURE TEMPLATE OQ

Figure 3-2 (con't.)

DCL NAME TYPE ATTRIBUTES & CROSS REFERENCE

**** ERROR **** ONE OR MORE VARIABLES REFERENCED BUT NOT ASSIGNED.

M A C R O T E X T L I S T I N G :

LOC TEXT

1 6

1
2
3
4

CALLS TO SCAN = 311
CALLS TO IDENTIFY = 85
NUMBER OF REDUCTIONS = 866
MAX STACK SIZE = 11
MAX IND. STACK SIZE = 10
END INC. STACK SIZE = 1
END ARRAY STACK SIZE = 0
MAX EXT_ARRAY INDEX = 6
XREF LIST ENTRIES = 71
STATEMENT COUNT = 28
NUMBER OF SYMBOLS = 30
MINOR COMPACTIFIES = 1
MAJOR COMPACTIFIES = 0
MAX NESTING DEPTH = 2
FREE STRING AREA = 33063

END OF HAL/S PHASE 1, MARCH 30, 1974. CLOCK TIME = 15:25:52.46.

39 CARDS WERE PROCESSED.
ONE ERROR WAS DETECTED IN PHASE 1.

**** SUMMARY OF ERRORS DETECTED IN PHASE 1 ****
ERROR #1 OF SEVERITY 1 IN CROSS-REFERENCE

TOTAL CPU TIME FOR PHASE 1 0:0:1.93.
CPU TIME FOR PHASE 1 SET UP 0:0:0.05.
CPU TIME FOR PHASE 1 PROCESSING 0:0:1.63.
CPU TIME FOR PHASE 1 CLEAN UP 0:0:0.25.
PROCESSING RATE: 1435 CARDS PER MINUTE.

***** COMPILATION ERRORS INHIBITED TEMPLATE GENERATION

- 5 Subscript and assignment
- 6 Reference and assignment
- 7 Reference, assignment, and subscript

Comments: Following the last reference listed for each variable, the compiler may insert a comment about the variable usage. Possible comments are:

NOT USED; if the identifier appears in a DECLARE statement but is neither referenced or assigned.

NOT REFERENCED; if the variable is assigned a value but never used in any reference context.

NOT ASSIGNED; if the variable value is referenced but never appears in any context where it can receive a value. This is obviously an error situation involving the use of an uninitialized variable. Therefore, the Symbol Table & Cross Reference listing contains the error message:

"*****ERROR*****REFERENCED BUT NOT ASSIGNED"

to call attention to this situation.

3.7 Macro Table

Following the cross references, there will be a MACRO TEXT listing if any REPLACE definitions appear in the program. Figure 3 contains a MACRO TEXT listing. The LOC field refers to the TYPE field in the symbol table listing for REPLACE class variables, and the TEXT indicates the string which was substituted whenever the identifier was encountered. REPLACE definitions are cross-referenced as well as any identifiers which may occur within replacement string.

3.8 Optional Unformatted Listing

The primary source listing generated by the HAL/S compiler is completely reformatted by the output writer. As a result, the source text has lost its relationship to the original card image input.

In order to give the programmer a correlation between the primary source listing and his original input, the HAL/S compiler will supply an optional unformatted source listing. A sample of such a listing is shown in Figure 3.3.

This option is requested by including the word "LISTING2" in the OPTION field of the JCL (see Section 2.1.3).

1	NI	DEMO:PROGRAM;		1	DEMO
	CI			2	DEMO
	CI	THIS IS A DEMONSTRATION PROGRAM TO SHOW THE LISTING PRODUCED BY		3	DEMO
	CI	THE HAL/S-360 COMPILER		4	DEMO
	CI			5	DEMO
2	NI	REPLACE PRINTER BY "6";		6	DEMO
3	NI	DECLARE INTEGER INITIAL(1),A,B,C;		7	DEMO
4	NI	DECLARE D ,P,G;		8	DEMO
5	NI	DECLARE Z VECTOR(4);		9	DEMO
6	NI	DECLARE H,I,J,K ARRAY(5) MATRIX(3,4),L,H,N,O SCALAR;		10	DEMO
7	NI	STRUCTURE AA:		11	DEMO
7	NI	1 BB,2 CC MATRIX(4,3), 1 DD,2 EE ARRAY(4) MATRIX(3,4)		12	DEMO
7	NI	:		13	DEMO
8	NI	STRUCTURE QQ:		14	DEMO
8	NI	1 RR,		15	DEMO
8	NI	2 AAREF AA-STRUCTURE,		16	DEMO
8	NI	2 SS CHARACTER(5);		17	DEMO
9	NI	DECLARE MY_STRUCTURE QQ-STRUCTURE(5);		18	DEMO
10	NI	PROC1:PROCEDURE;		19	PROC1
11	NI	DECLARE A INTEGER;		20	PROC1
12	NI	IF A=B THEN DO;		21	PROC1
14	NI	LABEL1:B=C;		22	PROC1
15	NI	MY_STRUCTURE.RR.AAREF.DD.EE\$(*;3:2,*) =		23	PROC1
15	NI	MY_STRUCTURE.RR.AAREF.BB.CC\$(*:* ,2);		24	PROC1
16	NI	END;		25	PROC1
16	NI	ELSE A=C;		26	PROC1
18	NI	CLOSE PROC1;		27	DEMO
19	NI	DO FOR C = 1 TO 100;		28	DEMO
20	NI	D=K\$(C:2,3);		29	DEMO
21	NI	END;		30	DEMO

49
>67

Figure 3-3 The HAL/S Unformatted Source Listing and Additional Phase I Information

22 N DO CASE A;	31	DEMO
22 N A=B;	32	DEMO
24 N E=K\$(B:A,*);	33	DEMO
25 N END;	34	DEMO
26 N K=0;	35	DEMO
27 N WRITE(PRINTER) MY_STRUCTURE.RR.AAREF.BB.CC\$(3;1 TO 3,*),	36	DEMO
27 N MY_STRUCTURE.RR.AAREF.DD.EE	37	DEMO
27 N \$(2;2 TO 4:),D,E,F,G,H,I,J,K,L,M,N,O;	38	DEMO
28 N CLOSE DEMO;	39	

Figure 3-3 (con't.)

37
A

CALLS TO SCAN = 311
CALLS TO IDENTIFY = 85
NUMBER OF REDUCTIONS = 866
MAX STACK SIZE = 11
MAX IND. STACK SIZE = 10
END IND. STACK SIZE = 1
END ARRAY STACK SIZE = 0
MAX EXT_ARRAY INDEX = 6
XREF LIST ENTRIES = 71
STATEMENT COUNT = 28
NUMBER OF SYMBOLS = 30
MINOR COMPACTIFIES = 1
MAJOR COMPACTIFIES = 0
MAX NESTING DEPTH = 2
FREE STRING AREA = 33063

END OF HAL/S PHASE 1, MARCH 30, 1974. CLOCK TIME = 15:25:52.46.

39 CARDS WERE PROCESSED.
ONE ERROR WAS DETECTED IN PHASE 1.

**** SUMMARY OF ERRORS DETECTED IN PHASE 1 ****
ERROR #1 OF SEVERITY 1 IN CROSS-REFERENCE

TOTAL CPU TIME FOR PHASE 1 0:0:1.93.
CPU TIME FOR PHASE 1 SET UP 0:0:0.05.
CPU TIME FOR PHASE 1 PROCESSING 0:0:1.63.
CPU TIME FOR PHASE 1 CLEAN UP 0:0:0.25.
PROCESSING RATE: 1435 CARDS PER MINUTE.

***** COMPILATION ERRORS INHIBITED TEMPLATE GENERATION

3.8.1 Format

The optional listing contains card images as read from the input stream. These images receive no compiler-generated annotation. Each card image is bracketed by vertical bars (|).

To the left of each image, the compiler prints the card type identification from column one of the input card. The HAL/S statement number of the first statement included on the card is also printed to the left of the card type. This statement number is the same as the one assigned to the expanded source by the compiler in the primary source listing.

To the right of the card image, the compiler places a sequential card number, one number per card, which simply indicates a card's relative position in the input deck.

Current Scope information is supplied in the same manner as in the primary listing. No other information is put into the auxiliary listing. This listing does not receive any error indication messages.

3.9 Additional Information

Immediately following the Macro Text Listing, the compiler prints a list of internal statistics for use in compiler development. These statistics can be ignored by the programmer.

3.10 Phase II Listing

The Phase II listing of the HAL/S Compiler identifies itself and lists current date and time before entering into the code generation process (see Figure 3.4).

If the 'OPTIONS=' field in the EXEC statement of the JCL specifies the 'LIST' option, a listing of all control section names needed by the compiled program is produced (see fig.). The format of this listing is typical of IBM model 360 compilers and translators.

Following this, a completely formatted object module listing will be produced (see fig. 3-4). This listing shows the complete code generated for each control section in both hexadecimal and pseudo-assembler formats. This is listed after a line in the following format:

```
ST#n      EQU      *
```

where n is the compiled HAL/S statement number, and 'ST#n' is in the label field. The label field is also used to indicate both HAL/S label names and internal branch points in the same format as the HAL/S statement number indicator. The comments field gives information about the symbolic operand referenced by the instruction.

Following this is a map of relocation information included in the produced object module.

Regardless of whether 'LIST' was specified or not, a listing of several performance statistics is printed next. These are included to aid in compiler generation, and are of no concern to the programmer.

3.11 Phase III

The .Phase III listing of the HAL/S compiler identifies itself and lists the current time and date before beginning construction of the simulation data file.

If the 'OPTIONS=' field in the EXEC statement of the JCL specifies the 'TABLIST' option, the following printout will result:

- 1) Translation Table - Lists the identification codes (decimal and hexadecimal) assigned to blocks, symbols, and statements (statement #) and gives the associated block name, symbol name, and SRN (if present), together with the virtual memory pointer to the corresponding file node.
- 2) Hex dump of the file, page by page.

Following the optional TABLIST data, Phase III prints a message stating that the SDF has been either created or replaced (updated).

A listing of performance statistics is printed next. These are included to aid in compiler generation, and are of no interest to the programmer.

HAL/S COMPILATION

I N T E R M E T R I C S I N C .

MARCH 30, 1974

15:25:47.95

PAGE 9

HAL/S COMPILER PHASE 2 -- VERSION 360-7.0 OF MARCH 29, 1974. CLOCK TIME = 5:57:59.03

HAL/S PHASE 2 ENTERED MARCH 30, 1974. CLOCK TIME = 15:25:58.96

Figure 3-4

CT
15
A

ESDID	NAME	TYPE	LENGTH	BLOCK NAME
0001	\$DEMO	0000	000226	DEMO
0002		0004	00008C	PROC1
0003	.DEMO	0000	00006B	
0004	%DEMO	0000	000070	
0005	*DEMO	0000	000608	
0006	@DEMO	0002		
0007	!DEMO	0002		
0008	#DEMO	0002		
0009	\$DEMO	0002		

BT
BT
A

LOCCTR	CODE	LABEL	INSN	OPERANDS	SYMBOLIC OPERAND
000000		ST#1	EQU	* TIME = 39	
000000	\$DEMO		CSECT	ESDID= 0001	
000000	DEMO		EQU	*	
000000	47F0F010		BC	15,16(0,15)	
000004	000002B8		DC	A'000002B8'	
000008	0050		DC	X'0050'	
00000A	04C4C5D4		DC	X'04C4C5D4'	
00000E	D6		DC	X'D6'	
00000F	00		DC	X'00'	
000010	58B0F004		L	11,4(0,15)	
000014	986AB028		LM	6,10,40(11)	
000018	05EB		BALR	14,11	
00001A	0001		DC	X'0001'	
00001C		ST#2	EQU	* TIME = 0	
00001C		ST#3	EQU	* TIME = 0	
000398	\$DEMO		CSECT	ESDID= 0005	
000398	0001		DC	X'0001'	
00039A		ST#3	EQU	* TIME = 0	
00039A	0001		DC	X'0001'	
00039C		ST#3	EQU	* TIME = 0	
00039C	0001		DC	X'0001'	
00039E		ST#4	EQU	* TIME = 0	
00039E		ST#4	EQU	* TIME = 0	
00039E		ST#5	EQU	* TIME = 0	
00039E		ST#6	EQU	* TIME = 0	
00039E		ST#7	EQU	* TIME = 0	
00039E		ST#8	EQU	* TIME = 0	
00039E		ST#9	EQU	* TIME = 0	
00039E		ST#10	EQU	* TIME = 10	
00001C	\$DEMO		CSECT	ESDID= 0001	
000228	\$DEMO		CSECT	ESDID= 0002	
000228	PROC1		EQU	*	
000228	47F0F010		BC	15,16(0,15)	
00022C	000002B8		DC	A'000002B8'	
000230	0050		DC	X'0050'	
000232	05D7D9D6		DC	X'05D7D9D6'	
000236	C3F1		DC	X'C3F1'	
000238	05EB		BALR	14,11	
00023A	000A		DC	X'000A'	
00023C		ST#11	EQU	* TIME = 0	
00023C		ST#12	EQU	* TIME = 33	
00023C	05EB		BALR	14,11	
00023E	000C		DC	X'000C'	
000240	4820A006		LH	2,6(0,10)	A
000244	4920A002		CH	2,2(0,10)	B
000248	476F0078		BC	6,120(15,0)	0002A0 LBL#4
000250		ST#13	EQU	* TIME = 0	
000250	05EB		BALR	14,11	
000252	000D		DC	X'000D'	
000254		ST#14	EQU	* TIME = 18	
000254		LABEL1	EQU	*	
000254	4820A004		LH	2,4(0,10)	C
000258	4020A002		STH	2,2(0,10)	B
00025C	05EB		BALR	14,11	
00025E	000E		DC	X'000E'	

Figure 3-4

LOCCTR	CODE	LABEL	INSM	OPERANDS	SYMBOLIC OPERAND
000260		ST#15	EQU	* TIME = 1601	
000260	41900001		LA	9,1(0,0)	
000264		LBL#5	EQU	*	
000264	1889		LR	8,9	
000266	4C80B04C		MH	8,76(0,11)	H'248'
00026A	1879		LR	7,9	
00026C	4C70B04C		MH	7,76(0,11)	H'248'
000270	4128A0D4		LA	2,212(8,10)	MY_STRUCTURE+156
000274	4137A03B		LA	3,56(7,10)	MY_STRUCTURE
000278	4140000C		LA	4,12(0,0)	
00027C	41000C04		LA	0,4(0,0)	
000280	45E0C060		BAL	14,96(0,12)	V1SNP
000284	41909001		LA	9,1(0,9)	
000288	4990B04E		CH	9,78(0,11)	H'5'
00028C	47CF003C		BC	12,60(15,0)	000264 LBL#5
000290	05EB		BALR	14,11	
000292	000F		DC	X'000F'	
000294		ST#16	EQU	* TIME = 0	
000294	05EB		BALR	14,11	
000296	0010		DC	X'0010'	
000298		ST#17	EQU	* TIME = 35	
000298	47FF0084		BC	15,132(15,0)	0002AC LBL#6
0002A0		LBL#4	EQU	*	
0002A0	4820A004		LH	2,4(0,10)	C
0002A4	4020A006		STH	2,6(0,10)	A
0002A8	05EB		BALR	14,11	
0002AA	0011		DC	X'0011'	
0002AC		ST#18	EQU	* TIME = 10	
0002AC		LBL#6	EQU	*	
0002AC	05EB		BALR	14,11	
0002AE	0012		DC	X'0012'	
0002B0	47F0C004		BC	15,4(0,12)	
00001C		\$DEMO	CSECT	ESDID= 0001	
00001C		ST#19	EQU	* TIME = 41	
00001C	41900C01		LA	9,1(0,0)	
000020		LBL#7	EQU	*	
000020	4090A004		STH	9,4(0,10)	C
000024	05EB		BALR	14,11	
000026	0013		DC	X'0013'	
000028	4990B04A		CH	9,74(0,11)	H'100'
00002C	472F0058		BC	2,88(15,0)	000058 LBL#8
000034		ST#20	EQU	* TIME = 58	
000034	4C90B048		MH	9,72(0,11)	H'3'
000038	8B900002		SLA	9,2	
00003C	89900002		SLL	9,2	
000040	7829A028		LE	2,40(9,10)	K+28
000044	7020A008		STE	2,8(0,10)	D
000048	05EB		BALR	14,11	
00004A	0014		DC	X'0014'	
00004C		ST#21	EQU	* TIME = 24	
00004C		LBL#9	EQU	*	
00004C	41900001		LA	9,1(0,0)	
000050	4A90A004		AH	9,4(0,10)	C
000054	47FF0020		BC	15,32(15,0)	000020 LBL#7
000058		LBL#8	EQU	*	

U
A

Figure 3-4

LOCCTR	CODE	LABEL	INSM	OPERANDS	SYMBOLIC OPERAND
000058	05EB		BALR	14,11	
00005A	0015		DC	X'0015'	
00005C		ST#22	EQU	* TIME = 95	
00005C	4820A000		LH	2,0(0,10)	A
000060	05EB		BALR	14,11	
000062	0016		DC	X'0016'	
000064	413F00E4		LA	3,228(15,0)	0000E4 LBL#10
00006C	59203000		C	2,0(0,3)	
000070	472F008A		BC	2,138(15,0)	00008A LBL#11
000078	8B200002		SLA	2,2	
00007C	47DF008A		BC	13,138(15,0)	00008A LBL#11
000084	58E230C0		L	14,0(2,3)	
000088	07FE		BCR	15,14	
00008A		LBL#11	EQU	*	
00008A	45E0C018		BAL	14,24(0,12)	ZRRSND
00008E	0D000308		DC	A'0D000308'	
000092		ST#23	EQU	* TIME = 35	
000092	47FF00F0		BC	15,240(15,0)	0000F0 LBL#12
00009A		LBL#13	EQU	*	
00009A	4820A002		LH	2,2(0,10)	B
00009E	4020A000		STH	2,0(0,10)	A
0000A2	05EB		BALR	14,11	
0000A4	0017		DC	X'0017'	
0000A6		ST#24	EQU	* TIME = 256	
0000A6	47FF00F0		BC	15,240(15,0)	0000F0 LBL#12
0000AE		LBL#14	EQU	*	
0000AE	4890A002		LH	9,2(0,10)	B
0000B2	4C90BC48		MH	9,72(0,11)	H'3'
0000B6	4A90A000		AH	9,0(0,10)	A
0000BA	8B900002		SLA	9,2	
0000BE	89900002		SLL	9,2	
0000C2	4890B046		SH	9,70(0,11)	H'16'
0000C6	4120A02C		LA	2,44(0,10)	E
0000CA	4139A00C		LA	3,12(9,10)	K
0000CE	41000004		LA	0,4(0,0)	
0000D2	45E0C040		BAL	14,64(0,12)	V1SN
0000D6	05EB		BALR	14,11	
0000D8	0018		DC	X'0018'	
0000DA		ST#25	EQU	* TIME = 17	
0000DA	47FF00F0		BC	15,240(15,0)	0000F0 LBL#12
0000E4		LBL#10	EQU	*	
0000E4	00000002		DC	X'00000002'	
0000E8	0000009A		DC	A'0000009A'	DEMO
0000EC	000000AE		DC	A'000000AE'	DEMO
0000F0		LBL#12	EQU	*	
0000F0	05EB		BALR	14,11	
0000F2	0019		DC	X'0019'	
0000F4		ST#26	EQU	* TIME = 481	
0000F4	41900001		LA	9,1(0,0)	
0000F8		LBL#15	EQU	*	
0000F8	1889		LR	8,9	
0000FA	4C80B044		MH	8,68(0,11)	H'12'
0000FE	89800002		SLL	8,2	
000102	2B00		SDR	0,0	
000104	4128A00C		LA	2,12(8,10)	K

LOCCTR	CODE	LABEL	INSN	OPERANDS	SYMBOLIC OPERAND
000108	4100000C		LA	0,12(0,0)	
00010C	45E0C0P8		BAL	14,248(0,12)	V16SN
000110	41909001		LA	9,1(0,9)	
000114	4990B04E		CH	9,78(0,11)	H'5'
000118	47CF00P8		BC	12,248(15,0)	0000P8 LBL#15
00011C	05EB		BALR	14,11	
00011E	001A		DC	X'001A'	
000120		ST#27	EQU	* TIME = 1062	
000120	41100006		LA	1,6(0,0)	
000124	41000003		LA	0,3(0,0)	
000128	05EC		BALR	14,12	
00012A	00000000		DC	A'00000000'	IOINIT
00012E	4130A31C		LA	3,796(0,10)	MY_STRUCTURE+740
000132	1B44		SR	4,4	
000134	41000003		LA	0,3(0,0)	
000138	41100003		LA	1,3(0,0)	
00013C	05EC		BALR	14,12	
00013E	00000000		DC	A'00000000'	M21SNP
000142	41900001		LA	9,1(0,0)	
000146		LBL#16	EQU	*	
000146	1889		LR	8,9	
000148	4C80B048		NH	8,72(0,11)	H'3'
00014C	8B800002		SLA	8,2	
000150	89800002		SLL	8,2	
000154	4138A254		LA	3,596(8,10)	MY_STRUCTURE+540
000158	1B44		SR	4,4	
00015A	41000003		LA	0,3(0,0)	
00015E	41100004		LA	1,4(0,0)	
000162	05EC		BALR	14,12	
000164	00000000		DC	A'00000000'	M21SNP
000168	41909001		LA	9,1(0,9)	
00016C	4990B048		CH	9,72(0,11)	H'3'
000170	47CF0146		BC	12,326(15,0)	000146 LBL#16
000174	7800A008		LE	0,8(0,10)	D
000178	05EC		BALR	14,12	
00017A	00000000		DC	A'00000000'	EOUT
00017E	4130A02C		LA	3,44(0,10)	E
000182	1B44		SR	4,4	
000184	41000001		LA	0,1(0,0)	
000188	41100004		LA	1,4(0,0)	
00018C	05EC		BALR	14,12	
00018E	00000000		DC	A'00000000'	M21SNP
000192	7800A00C		LE	0,12(0,10)	F
000196	05EC		BALR	14,12	
000198	00000000		DC	A'00000000'	EOUT
00019C	7800A010		LE	0,16(0,10)	G
0001A0	05EC		BALR	14,12	
0001A2	00000000		DC	A'00000000'	EOUT
0001A6	7800A014		LE	0,20(0,10)	H
0001AA	05EC		BALR	14,12	
0001AC	00000000		DC	A'00000000'	EOUT
0001B0	7800A018		LE	0,24(0,10)	I
0001B4	05EC		BALR	14,12	
0001B6	00000000		DC	A'00000000'	EOUT
0001BA	7800A01C		LE	0,28(0,10)	J

Figure 3-4

LOCCTR	CODE	LABEL	INSN	OPERANDS	SYMBOLIC OPERAND
0001BE	05EC		BALR	14,12	
0001C0	00000000		DC	A'00000000'	EOUT
0001C4	41900001		LA	9,1(0,0)	
0001C8		LBL#17	EQU	*	
0001CB	1889		LR	8,9	
0001CA	4C80B044		MH	8,68(0,11)	H'12'
0001CE	89800002		SLL	8,2	
0001D2	4138A00C		LA	3,12(8,10)	K
0001D6	1B44		SR	4,4	
0001D8	41000003		LA	0,3(0,0)	
0001DC	41100004		LA	1,4(0,0)	
0001E0	05EC		BALR	14,12	
0001E2	00000000		DC	A'00000000'	H21SNP
0001E6	41909001		LA	9,1(0,9)	
0001FA	4990B04E		CH	9,78(0,11)	H'5'
0001EE	47C701C8		BC	12,456(15,0)	0001CB LBL#17
0001F2	7800A020		LE	0,32(0,10)	L
0001F6	05EC		BALR	14,12	
0001F8	000000C0		DC	A'00000000'	EOUT
0001FC	7800A024		LE	0,36(0,10)	H
000200	05EC		BALR	14,12	
000202	00000000		DC	A'00000000'	EOUT
000206	7800A028		LE	0,40(0,10)	N
00020A	05EC		BALR	14,12	
00020C	00000000		DC	A'00000000'	EOUT
000210	7800A02C		LE	0,44(0,10)	O
000214	05EC		BALR	14,12	
000216	00000000		DC	A'00000000'	EOUT
00021A	05EB		BALR	14,11	
00021C	001B		DC	X'001B'	
00021E		ST#28	EQU	* TIME = 10	
00021E	05EB		BALR	14,11	
000220	001C		DC	X'001C'	
000222	47F0C004		BC	15,4(0,12)	
0002B8		\$DEMO	CSECT	ESDID= 0003	
0002B8	47F0C174		BC	15,372(0,12)	STRACE
0002BC	00000398		DC	A'00000398'	
0002C0	00000328		DC	A'00000328'	
0002C4	FF000324		DC	A'FF000324'	
0002C8	00000000		DC	A'00000000'	DEMO
0002CC	0001		DC	X'0001'	
0002CE	001C		DC	X'001C'	
0002D0	00000000		DC	X'00000000'	
0002D4	00000000		DC	X'00000000'	
0002D8	00000000		DC	X'00000000'	
0002DC	00000000		DC	X'00000000'	
0002E0	00000398		DC	A'00000398'	
0002E4	00000398		DC	A'00000398'	
0002E8	00000398		DC	A'00000398'	
0002EC	00000398		DC	A'00000398'	
0002F0	00000398		DC	A'00000398'	
0002F4	00000001		DC	X'00000001'	
0002F8	00000000		DC	A'00000000'	@DEMO
0002FC	000C		DC	X'000C'	
0002FE	0010		DC	X'0010'	

Figure 3-4

LOCCTR	CODE	LABEL	INSN	OPERANDS	SYMBOLIC OPERAND
000300	0003		DC	X'0003'	
000302	0064		DC	X'0064'	
000304	00F8		DC	X'00F8'	
000306	0005		DC	X'0005'	
000308	1AC3C1E2		DC	X'1AC3C1E2'	
00030C	C540E5C1		DC	X'C540E5C1'	
000310	D9C9C1C2		DC	X'D9C9C1C2'	
000314	D3C540D6		DC	X'D3C540D6'	
000318	E4E340D6		DC	X'E4E340D6'	
00031C	C640D9C1		DC	X'C640D9C1'	
000320	D5C7C5		DC	X'D5C7C5'	
000328		SDemo	CSECT	ESDID= 0004	
			END		

RLD	POS	REF	FLAG	ADDRESS
-----	-----	-----	------	---------

0001	0009		CB	000217
0001	0009		08	00020D
0001	0009		08	000203
0001	0009		CB	0001F9
0001	0008		08	0001E3
0001	0009		08	0001C1
0001	0009		08	0001B7
0001	0009		08	0001AD
0001	0009		08	0001A3
0001	0009		CB	000199
0001	0008		08	00018F
0001	0009		08	00017B
0001	0008		08	000165
0001	0008		08	00013F
0001	0007		08	00012B
0001	0001		CB	000CED
0001	0001		08	000CE9
0001	0003		08	00008F
0001	0003		CB	000C05
0002	0003		08	00022D
0003	0006		CB	0002F9
0003	0005		CB	0002F1
0003	0005		CB	0002ED
0003	0005		CB	0002E9
0003	0005		CB	0002E5
0003	0005		CB	0002E1
0003	0001		08	0002C9
0003	0004		CB	0002C5
0003	0004		CB	0002C1
0003	0005		08	0002BD

LOC	B	DISP	NAME
-----	---	------	------

UNDER DEMO

000398	A	000	A
00039A	A	002	B
00039C	A	004	C
0003A0	A	008	D
0003A4	A	00C	F
0003A8	A	010	G
0003C8	A	02C	E
0003AC	A	014	H
0003B0	A	018	I
0003B4	A	01C	J
0003D8	A	00C	K
0003BB	A	020	L
0003BC	A	024	M
0003C0	A	028	N
0003C4	A	02C	O
0004C8	A	038	MY_STRUCTURE

UNDER PROC1

00039E	A	006	A
--------	---	-----	---

INSTRUCTION FREQUENCIES

INSN	COUNT
BALB	15
BCE	1
LR	5
SR	4
SDR	1
STH	4
LA	34
BAL	4
BC	18
LH	6
CH	6
AH	2
SH	1
MH	7
L	2
C	1
STE	1
LE	11
SLL	5
SLA	4
LH	1

*** CONVERSION ERRORS INHIBITED EXECUTION

102 HALMAT OPERATORS CONVERTED

920 BYTES OF PROGRAM, 1544 BYTES OF DATA

MAX. OPERAND STACK SIZE	=7
END OPERAND STACK SIZE	=0
NUMBER OF STATEMENT LABELS USED	=17
MAX. STORAGE DESCRIPTOR STACK SIZE	=0
END STORAGE DESCRIPTOR STACK SIZE	=0
NUMBER OF MINOR COMPACTIFIES	=1
NUMBER OF MAJOR COMPACTIFIES	=0

END OF HAL/S PHASE 2 MARCH 30, 1974. CLOCK TIME = 15:26:4.76

TOTAL CPU TIME FOR PHASE 2	0:0:1.43
CPU TIME FOR PHASE 2 SET UP	0:0:0.03
CPU TIME FOR PHASE 2 GENERATION	0:0:0.35
CPU TIME FOR PHASE 2 CLEAN UP	0:0:1.05

4. DEBUGGING AIDS

4.1 Compilation Errors

4.1.1 Message Format

When Phase 1 of the HAL/S compiler (the syntax checking phase) detects an error condition, a diagnostic message is placed in the primary source listing at the point of detection. These error messages have the following form:

```
***** c ERROR #n OF SEVERITY s. *****  
  
***** text of message
```

In this message:

c = mnemonic error name uniquely identifying this error message (see Sec. 4.1.2)

n = indication that this is the nth error in the current compilation

s = severity of error (see Sec. 4.1.3)

For error messages other than the first in a given compilation, the following line is placed after all error messages referring to a particular HAL/S statement:

```
***** LAST ERROR WAS DETECTED AT STATEMENT m *****
```

where m is the HAL/S statement number of the most recent previous statement that received an error message.

4.1.2 Classification Scheme

Each error message that may be generated by Phase 1 of the HAL/S compiler has a unique mnemonic designation which appears in the error printout. These mnemonics have been assigned according to general error-type classes. The first letter of each error message mnemonic indicates the major class to which the error belongs (see Appendix D). The second letter, if present, indicates a sub-class further describing the error. These one or two letters

are followed by a number which simply indicates members of a class-subclass group. The table at the beginning of Appendix D shows the meaning of the possible letter combinations.

4.1.3 Error Severity

The severity indication in the error message shows the effect of the error on the compilation process. The possible severities and their effects are as follows:

- 0 = warning (compilation proceeds normally)
- 1 = error (compilation proceeds, execution prevented)
- 2 = severe error (syntax check continues,
code generation prevented)
- >2 = abortive error (compilation halts immediately)

4.1.4 Phase I Error Summary

Near the end of the Phase I source listing and table printout, a summary of detected errors is printed in the following form.

```
END OF HAL/S PHASE 1, <date>.  CLOCK TIME = <time>
n CARDS WERE PROCESSED.
x ERRORS WERE DETECTED.  THE MAXIMUM SEVERITY WAS y.
  THE LAST ERROR DETECTED WAS AT STATEMENT z.
***** SUMMARY OF ERRORS DETECTED IN PHASE 1 *****
      ERROR #1 AT STATEMENT e1 OF SEVERITY s1
      ERROR #2 AT STATEMENT e2 OF SEVERITY s2
      .
      .
      .
```

4.1.5 Phase II Errors

Phase II of the HAL/S compiler (the code generation phase) may also produce some error messages (see Sec. 3.6). These messages have the form:

```
*** ERROR #1 DURING CONVERSION OF HAL/S STATEMENT n
      text of message
```

where n is the HAL/S compiler-assigned statement number to which the error refers. Following these specific messages, Phase II supplies a disposition message indicating the total effect of all Phase II messages on the compilation.

4.2 Execution Errors

4.2.1 Introduction

This section describes how error handling has been implemented in the HAL/S 360 system. In the HAL/S system every error is assigned a non-zero positive number. There are two classes of errors:

- i) system-defined errors, (1-99) which arise as a result of failure during the execution of a user's program, and are signalled internally;
- ii) user-defined errors, (100-120) which are signalled by the user through a SEND ERRORn statement.

System-defined errors may also be signalled by the user through a SEND ERRORn statement.

The HAL/S Error Processor processes all errors and determines what action is to be taken. The user can gain control after an error by means of the ON ERRORn GO TO xxx statement, or leave the system in control by default or by means of the ON ERRORn SYSTEM statement. Additionally, the user can specify that the HAL/S Error Processor take no action for a specific error by means of the ON ERRORn IGNORE statement. These three possible actions will be referred to as GO TO action, SYSTEM action, and IGNORE action respectively.

In the following sections, the error format given shows the maximum information printed. If the compile time TRACE option is not in effect then there will be no information on the statement number of the last HAL/S statement to be executed.

4.2.2 GO TO Action

If the user has requested control for a particular error via a HAL/S ON ERROR GO TO statement, control is transferred to the appropriate location. If the runtime parameter MSGLEVEL has been specified with a value greater than zero, the following message is printed:

```
***HAL ERROR n - (message) -->(location)
LAST STATEMENT WAS m
TRANSFER TO GO TO label -->(location)
**
```

If the MSGLEVEL value is zero, no message is printed. Certain errors cannot result in GO TO action. If such an error occurs after the error has been specified in an ON ERROR GO TO statement, SYSTEM action ensues.

4.2.3 SYSTEM Action

One of three possible SYSTEM actions may ensue:

UNLIMITED - The standard fixup for ERRORn is taken and the following message is printed:

```
***** HAL ERROR n - (message) --> (location)
LAST STATEMENT WAS m
STANDARD FIXUP, EXECUTION RESUMED
```

where (message) is the error text corresponding to ERRORn. The last HAL/S statement to be fully executed is specified by m and the address of the error is specified by (location). Following this, the Error Processor returns control to the user's program.

LIMITED - The error count for ERRORn is updated. If it equals or exceeds a user-set maximum count (refer to the the ERRORLIM option in Appendix B. - note that this is applied to all LIMITED type errors for which SYSTEM action is specified), the following message is printed:

```
***** HAL ERROR n - (message) --> (location)
LAST STATEMENT WAS m
ERROR COUNT EXCEEDED
```

and execution of the user's program abnormally terminates.

Otherwise, if the maximum count is not exceeded, the standard fixup for ERRORn is taken and the following message is printed:

```
***** HAL ERROR n - (message) --> (location)
LAST STATEMENT WAS m
STANDARD FIXUP, EXECUTION RESUMED
```

and the Error Processor returns control to the user's program.

In both cases, (message) is the text for

67<

ERRORn, (location) is the address where the error occurred, and m is the last HAL/S statement to be fully executed.

TERMINATE - The following message is printed:

```
***** HAL ERROR n - (message) --> (location)
LAST STATEMENT WAS m
SYSTEM ACTION IS TERMINATE
```

where (message) is the text for ERRORn, (location) is the address where the error occurred, and m is the last HAL/S statement to be fully executed. Following this, execution of the user's program abnormally terminates.

Note that the form of the SYSTEM action and standard fixup is specified for each error message in Appendix E.

4.2.4 IGNORE Action

If a particular error has been specified in a HAL/S ON ERROR IGNORE statement, the occurrence of the error causes the standard fixup for the error to occur. However, no message is generated and the occurrence of the error does not contribute towards the termination error count for that error. Certain errors cannot be ignored. If such an error occurs after an ON ERROR IGNORE statement has been executed for the error, SYSTEM action ensues.

4.2.5 The Error Summary

Whether execution terminates normally or abnormally, if there were any errors signalled either internally or by the user, a summary in the following form is printed on the output channel defined by the MCHAN RUNPARAM option (see Appendix B):

```
*** SUMMARY OF ERRORS ***

ERROR#  SYSTEM  GOTO  IGNORE
<e>    <ne1>  <ne2>  <ne3>
      .
      .
      .
```

where <e> is an error number and <ne1>, <ne2>, and <ne3> are the individual counts, by action type, for the error.

If the MSGLEVEL parameter has a value greater than

zero and if no errors occurred, the following message is printed:

NO ERRORS OCCURRED IN THIS RUN

4.3 Execution Dumps and Traces

When an executable form of a HAL/S program (or program complex) has been produced by the HALLINK program as described in Section 2, its execution may be carried out in one of two ways:

- 1) direct execution of the program;
- 2) execution of the program under control of an execution monitoring system.

In either mode of execution, actual operation of the HAL/S program is identical. The differences between the two modes lie in the type and degree of diagnostic aids available.

4.3.1 Direct Execution

Direct Execution of a HAL/S program is achieved by naming the program in the PGM= field of an EXEC card of an OS/360 job step. This type of execution is automatic if the HALSLG or HALSCLG catalogued procedures of Appendix C are used. Parameters may be passed to the module via the PARM field or, if the catalogued procedures are used, via the RUNPARM field. The legal parameters and their effects are listed in Appendix B with one exception: the DUMP= parameter has no effect under direct execution. This means that no dump of HAL/S variables may be obtained (even during abnormal termination) when a HAL/S program is executed directly. HAL/S variable dumps may be obtained by running under the execution monitoring system described next in Section 4.3.2 Under direct mode, all features of the simulated Real Time executive (see section 5) are available since the necessary routines are made part of the HAL/S load module during the HALLINK step.

4.3.2 The Execution Monitoring System

To obtain the most complete diagnostic capability, the user may run a HAL/S load module under control of the execution monitoring system. Execution in this mode is produced by using the HALSCLD and HALSLD catalogued procedures in Appendix C. Under this mode, initial execution control is given to a monitor program (RUNMON) by specifying RUNMON in the PGM= field of the EXEC card. RUNMON is a processing program supplied with the basic HAL/S-360 compiler system.

RUNMON performs two basic tasks:

- 1) loading and execution of a diagnostic request processor;
- 2) loading and execution of a HAL/s program as directed by the inputs to the diagnostic request processor.

Under this system, the user indicates diagnostic requests via an input dataset. The dataset is indicated in the JCL of the execution step as:

```
//GO.REQUESTS DD *  
.  
.  
  <request cards>  
.  
.  
/*
```

The REQUEST DD card may also indicate any existing dataset into which the requests have been previously stored.

Under Release 7 of the HAL/S-360 Compiler System, the requests which may be made are quite simple. The legal requests are:

```
EXECUTE <module name>  
PARAMETERS '<run time options>';
```

<module name> is the name of the load module which contains the HAL/S program to be executed. The specified module is loaded from the dataset indicated on the HALMOD DD card. In the above illustration, <runtime options> is a set of options as described in Appendix B. All options may be specified including requests for dumps of HAL/S variables.

If the HALSCLD catalogued procedure were used to compile, link, and execute a HAL/S program, the following REQUESTS DD card could be used:

```
//GO.REQUESTS DD *  
EXECUTE TEMPNAME  
PARAMETERS 'TRACE=4,DUMP=2'  
/*
```

The EXECUTE request indicates a module named TEMPNAME. This name is the default module name generated by HALLINK. If specific HALLINK requests to change the default are not made. (Changing the module name may be accomplished by the inclusion of a NAME card in the HALLINK input stream, or by specifying a member name on

the SYSLNOD DD card in the HALLINK step.) The above PARAMETERS request indicates that executive tracing of realtime activity (TRACE=4) and an unconditional termination dump (DUMP=2) are to be provided.

Note that to perform any type of dump activity, the execution monitoring system must be provided with the Simulation Data Files (SDF) for any HAL/S compilation units for which dumps are expected. These SDF datasets are specified via the HALSDF DD card. See Section 2 for more information about the creation and saving of SDF's.

The RUNMON first loads and executes the diagnostic request processor which reads and interprets the requests from the REQUESTS DD card. The module indicated in the EXECUTE request is then executed under control of the special requests given in the PARAMETERS request. If a REQUESTS DD card is defined, any information found in the PARM field passed to the RUNMON program is ignored.

In future releases of the system, many of the requests which are now specified in the PARAMETERS option will be expanded and will allow much more detailed control of diagnostics. This will in general be done by removal of options from the set of legal options in Appendix B and by definition of new legal inputs on the REQUESTS DD card. The future requests will be made via a diagnostic language which will allow such actions as tracing of single statements and dumps of specified variables at specific points of execution, including dumps based on time.

An alternate method for executing a HAL/S program under control of the execution monitoring system has been defined. If no REQUESTS DD card is present when the RUNMON program is executed, certain default actions will take place:

- 1) the HAL/S module to be executed is presumed to have the name TEMPNAME;
- 2) any PARM field information provided to the RUNMON program is used as if it had been found on a PARAMETERS request card. In the HALSCLD and HALSLD procedures, the term field may be specified via the RUNPARM JCL option.

This alternate system allows users to run their HAL/S programs under the execution monitoring system in much the same way as they can under the direct execution mode. If the user is doing initial debugging of a new program, simply changing the name of a catalogued procedure being used from HALSCLG to HALSCLD will provide identical program behavior, but will provide the benefit of a dump

of all HAL/S variables in the event of an abnormal termination.

4.3.3 Location of Diagnostic Output

Any printed output generated by the HAL/S runtime system in either direct execution or execution under the execution monitoring system is sent to the dataset defined as the message dataset for the run. This dataset defaults to the one indicated on the CHANNEL6 DD card and may be altered by the MCHAN option as described in Appendix B.

5. HAL/S REALTIME PROGRAMS

5.1 Using the Real-Time Features of HAL/S-360

5.1.1 Introduction

The Real-Time features of HAL/S-360 - SCHEDULE, WAIT, SIGNAL, etc. - have been implemented in Releases 360-4 and 360-5. This section supplements Chapter 8 of the HAL/S Language Specification with details about compilation and run-time options, timing, and executive trace mode messages.

5.1.2 Terms and Concepts

Process:

The dynamic entity created when a SCHEDULE statement is executed.

Program, Task:

The static blocks which are scheduled and executed as processes; the only allowed object of a SCHEDULE statement.

Priority:

A number, implicitly or explicitly assigned to a process when it gets scheduled, and changeable by the UPDATE PRIORITY statement, that determines which of two or more ready processes is executing. A higher value indicates greater importance. In HAL/S-360, priority may range from 0 to 255. A value specified outside this range results in a value MOD 256 with no error message. The priority of the initial process is 50.

Process Queue:

A list of active processes ordered by priority.

Process States:

- a) Inactive: A non-existent process. A program or task block that has not yet been scheduled or has been terminated.

- b) Active: The process exists on the process queue. A process becomes active when it is scheduled. An active process may be waiting, ready, or executing. A process remains executing until it terminates in any of several ways.

- c) Waiting: The process is unable to execute because some condition has not been met. Different kinds of conditions exist:
 - 1) time;
 - 2) event expression;
 - 3) completion of dependent processes;
 - 4) the use of an exclusive procedure.

- d) Ready: The process is able to execute but is not because some other process is executing.

- e) Executing: The process is progressing through a sequence of HAL statements via CPU control.

Process Swap: A process state transition where one process leaves the executing state and another process changes from ready to executing. This may be due to the first process entering the wait state or inactive (terminated) state, or because the second process has a higher priority.

Breakpoint: A place in the code sequence at which the existence of a higher priority ready process is allowed to cause a process swap. Currently in HAL/S-360, this is at the end of every statement.

Event (variable): A HAL variable with a boolean ON/OFF (or TRUE/FALSE) value. The value (or state) of an event variable is under programmer control via the SIGNAL, SET, and RESET statements. Process event variables process values that indicate the state of the associated process: active-ON, inactive-OFF.

Event expression: A logical combination of event variables (using AND, OR, NOT) which specifies a condition for which a process may wait or be cancelled.

HAL/S-360 load module: An OS/360 load module which is the result of the HALLINK step. It contains one or more HAL compilation units (program, comsub, or compool) and all modules in the run-time library to which they refer. It runs under OS/360 as a single OS task.

Termination or Completion: The transition of a process to the inactive state; its removal from the process queue.

a) Normal: When an executing process reaches the highest level RETURN or CLOSE statement in the program or task, it automatically waits for the completion of all its dependent processes, if any exist. Following this, the process is terminated provided it is not cyclic (scheduled with a REPEAT option). If it is a cyclic process, it is prepared for another cycle of execution, unless it has been cancelled (see below).

b) Cancellation: This provides a safe way to stop a cyclic process. If a process has not yet begun execution or is between cycles of execution, cancellation causes immediate termination. If a process has begun but not yet completed a cycle, cancellation causes normal termination, as above, at the end of the current cycle. Thus, cancellation does not affect non-cyclic processes which have already begun execution. Cancellation can occur in three ways:

1) a CANCEL statement;

- 2) at a time specified in the UNTIL <time value> expression on the SCHEDULE statement;
- 3) by an UNTIL or WHILE event expression on the SCHEDULE statement (see HAL/S Language Specification for the special case when using UNTIL <event expression>).

c) Abnormal: Immediate termination of a process and all its dependent processes, if any exist, occurs only as the result of a TERMINATE statement. The process(es) may be in any of the active states. Clean-up measures during termination guarantee that all system resources are freed, but the result with regard to values of COMPOOL variables and the effects of partial completion of specific computations and control sequences are unpredictable. If this proves to be a problem, the language specification and implementation of TERMINATE could be changed to allow execution of block-level specific clean-up procedures via the ON ERROR GO TO mechanism.

Real Time Executive: Those routines in the library which implement the real time features of HAL/S-360. This includes:

- a) a process manager which selects and executes ready processes and controls cyclic execution and normal termination;
- b) timer handling routines;
- c) event handling routines;
- d) process service routines corresponding to the real time statements.

5.1.3 Timing

The timing implied by the real time statements and returned by the built-in function RUNTIME is an AP-101 pseudo-time in units of seconds. The pseudo-time is maintained in HAL/S-360 by a post-statement processor in the run-time library that is called at the end of each executable statement. The compiler will assign a time cost for each statement in terms of an integral number of micro-second timer units based on the generated code sequence. There is a global constant which is the number of these units in one second of CPU operation. This

conversion number is modifiable by the SPEED Execution option.

A pseudo interval timer which contains the number of machine cycles until the next timed action is decremented at the end of each statement by its time cost. When the timer is decremented to or through zero, a pseudo-interrupt routine is called which takes the appropriate actions, including re-loading the timer with the interval to the next timed action on the timer queue.

HAL variables which contain absolute or relative time values should be declared as single or double precision scalars. The timer routines maintain time in double precision floating point seconds.

Both the initial time value at the start of a HAL run and the constant equating machine cycles per second are under control of the programmer via PARM field options (see Below).

5.1.4 PARM Field Options

See Section 2.1.3 and Appendix B of this manual for other compile-time and run-time options.

5.1.5 Compile Time

The compile time TRACE option must be specified to get the compiler to include the calls to the pseudo-time statement processor. Without this, the pseudo-time will not advance properly.

5.1.6 Execution Time

- 1) SIMTIME= number: initial pseudo-time value
(DEFAULT=0)
- 2) SPEED = number: timer units per second
(DEFAULT = 10,000,000).
- 3) PCBS = number: maximum number of simultaneously
Active processes (DEFAULT = 10).

Note: This parameter will be eliminated in a later compiler release when the value will be set automatically to the total number of programs and tasks in the HAL/S-360 load module.

4) TRACE = 4: Specifies a mode of tracing where all "significant interactions in the Real Time Executive are printed on the message channel. The format of these trace messages is:

***RTC TRACE: TIME = <time> <process-id> - <message>

where <time> is the current value of the pseudo-time in seconds; <process-id> is PROGRAM name (priority) or TASK name (priority); and <message> is the action being taken. The different messages and their meanings are listed in the next section.

5.1.7 List of Real Time Messages

1) READIED FROM WAIT: wait type

A condition, indicated by "wait type", for which the process was waiting has occurred. The named process can now compete for CPU execution based upon its priority.

2) SCHEDULED

A SCHEDULE statement creating the named process was successfully executed.

3) INITIATED

The process is about to become executing for the first time since being scheduled.

4) RESUMED

The process is about to become executing after being readied from a wait state or suspended at a breakpoint.

5) AT RETURN OR CLOSE

The process finished normally. What happens next depends on whether or not there were any active dependents (if so, it enters a wait state - message #11), and whether or not it was scheduled with a REPEAT option (if so, it is either repeated immediately - message #6, or put into an inter-cycle wait state - message #11, or terminated, if it has been cancelled or if it is not cyclic - message #9).

6) REPEATED

78<

A process scheduled with the REPEAT option is about to begin another cycle of execution. The beginning of the first cycle is traced with the INITIATED message.

7) FASTTIME

This indicates that the process manager discovered no ready processes, but there was at least one process waiting for a future time. The pseudo-time is advanced to that time so that work can be done.

(Note that no <process-id> appears with these messages because the action does not relate to a specific process.)

8) END OF RUN

There are no active processes. There is no more work to do and none can be generated.

(Note that no <process-id> appears with these messages because the action does not relate to a specific process.)

9) TERMINATED

The process was terminated because of one of the following:

- a) a TERMINATE statement was executed;
- b) normal termination occurred;
- c) cancellation resulted in termination.

10) CANCELLED (CANCEL)

The process was canceled via the CANCEL statement. The TERMINATED message may or may not follow immediately, depending upon the state of the process.

11) ENTERED WAIT: wait type

The process entered the wait state either explicitly with a WAIT statement or implicitly at the beginning of an exclusive procedure, or at the RETURN or CLOSE at the highest level.

12) SIGNALLED EVENT

The process executed a SIGNAL statement. If the change of state of the event variable had any effect, further messages will follow.

13) SUSPENDED AT BREAKPOINT

The process manager suspended execution of a process because there was a higher priority process.

14) PRIORITY UPDATED

An UPDATE PRIORITY statement was executed, changing the priority of the process.

15) CANCELLED (EVENT)

The cancel condition specified in the UNTIL or WHILE <event expression> phrase of the SCHEDULE statement which created the process has been satisfied.

16) CANCELLED (time)

The cancel time specified in the SCHEDULE UNTIL phrase has arrived.

17) SET/REST (event)

A SET or RESET statement altered the state of an event variable. Any effect will be indicated in following messages.

5.1.8 List of Real Time Wait-Types

The wait types and the statements causing them are:

Wait Types in Trace Message Statements

1. TIME	WAIT <value>
2. UNTIL TIME	WAIT UNTIL <value>
3. FOR EVENTS	WAIT UNTIL <event expression>
4. SCHED IN TIME	SCHEDULE X IN <value>
5. SCHED AT TIME	SCHEDULE X AT <value>
6. SCHED ON EVENTS	SCHEDULE X ON <event expression>
7. FOR DEPENDENTS	WAIT FOR DEPENDENT
8. FOR DEPENDENTS	

AT RETURN/CLOSE	RETURN or CLOSE
9. EXCLUSIVE PROCEDURE	entry to busy exclusive procedure
10. REPEAT EVERY	RETURN or CLOSE
11. REPEAT AFTER	RETURN or CLOSE

5.2 HAL/S Load Module and Operating Environment

Object modules output from separate compilations of one or more HAL/S-360 programs, zero or more COMSUBS (external HAL/S procedures and functions), and zero or more COMPOOLS (external data blocks) are linked together in a HALLINK step which automatically includes any needed members of the run time library. The result is a HAL/S-360 load module which may be run in batch mode as an OS/360 job step.

The HAL/S-360 load module executes as a single task under OS/360 MVT. Within this single OS/360 task, a HAL/S multiprocessing environment exists in which programs and tasks may be scheduled as dynamic processes that share CPU execution based on priority. The HAL/S programmer controls the multiprocessing environment via execution of the real time statements (SCHEDULE, WAIT, SIGNAL, TERMINATE, CANCEL, etc.). These statements in a HAL/S program are compiled as calls to a small subset of the routines in the run time library known as the real time "executive" which models the process management function of the Flight Computer Operating System (FCOS). The first HAL program in a load module is scheduled by the "executive" as one of its initialization functions, so that a single HAL/S program with no real time statements executes as if it were merely called in single process system.

5.3 Processes and the Stack Mechanism

Since processes may execute independently of each other, a mechanism is required which cleanly guarantees non-interference in the use of temporary storage for work areas, register save areas, call and return linkages, parameter passing, and any other process specific information. The stack not only satisfies this requirement in a process swapping environment, but also results in minimum temporary storage requirements and easy implementation of reentrant procedures.

The stack is a contiguous area of storage defined as a

CSECT of sufficient size to handle all temporary storage requirements. A stack CSECT is created in the HAL/S load module. During execution, the address of the portion of the stack in use by the current procedure (called the stack frame) is contained in general purpose register 13 (R13).

The layout of the initial portion of the stack frame is fixed in format and is identical for user procedures and functions (internal or external CONSUBS) and run time library routines (see Figure 5-3). The rest of the stack frame depends on the temporary storage requirements of the individual routine. The total size of the stack frame is specified in the procedure prologue. The minimum stack frame size is 80 bytes, the size of the fixed portion.

5.4 Procedures and the Procedure Caller

In the following discussion, the word procedure applies to all run time library routines (excluding intrinsics) as well as to all user-written HAL procedures and functions.

5.4.1 Calling

When a procedure is called, parameters are loaded and control is transferred to the procedure caller (see Figure 5-4). The procedure caller itself is code located in the HALSYS CSECT. Its function is to:

- 1) increment the stack frame pointer (R13) to the end of the current stack frame, thus defining the beginning of the new stack frame;
- 2) save the registers of the calling procedure in the new stack frame;
- 3) initialize certain control words in the stack;
- 4) branch to the entry address (or prologue) of the procedure (see Figure 5-5).

In the new stack frame, saving of the registers establishes the link to the previous stack frame, the return address, and any parameters passed in registers R0 through R4. Parameters passed in floating point registers are not saved in the stack automatically.

The entry address of the called procedure is copied from the 4 bytes following the BALR instruction to the stack in order to:

- 1) align it to a fullword boundary for loading into R15;
- 2) save it if the procedure calls for any other procedures.

Thus, the base of the calling procedure (R15) need not be saved in the new stack frame, allowing the use of R15 in updating the stack frame pointer.

At any time during execution, the stack represents the dynamic nesting of the called procedures. It is a last-in, first-out stack of stack frames, each frame representing a level of dynamic procedure nesting.

The procedure prologue (see Figure 5-6) is very simple, consisting of a single branch instruction that skips over some constant data located at fixed offsets from the entry point. In the case of alternate entry points, two additional instructions set the base to the main entry point.

Figure 5-4

PROCEDURE AND FUNCTION CALLS

Load	R0,	Argument 0
Load	R1,	Argument 1
Load	R4,	Argument 4
Load	F0,	Floating Argument 0
BALR	R14,	R12 GO TO PROCEDURE CALLER
DC	,	Adcon of Entry Point

PROCEDURE AND FUNCTION EXITS

B 4(R12) GO TO PROCEDURE EXITER

Notes:

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

b) The one-byte is the lexical level of the called procedure or function, as follows:

	Block Type
0	Programs and tasks, library routines;
1	COMSUBS, and first level procedures;
>2	Nested procedures.

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

Figure 5-5

PROCEDURE CALLER		
LH	R15, 8(R15)	Load current stack frame size
AR	R15, R13	Increment stack frame pointer to next stack frame;
STM	R0, R14, 20(R15)	Save caller's register in new stack frame;
LR	R13, R15	Set new stack frame ptr.
SR	R15, R15	
ST	R15, 16(R13)	Zero error link field
ST	R15, 4(R13)	Zero statement index and and flag field;
MVC	0(4, R13), 0(R14)	Copy called proc. base to stack frame;
L	R15, 0(R13)	Load called proc. base;

84<

```
BR      R15          GO TO Procedure;
```

PROCEDURE EXITER

```
LM      R2, R14, 28(R13)  Restore caller's register  
                                (including old stack frame  
                                and return address);  
L       R15, 0(R13)      Restore caller's base;  
B       4(R14)           Return to caller,  
                                skipping -Adcon;
```

Note: R0 and R1 are not restored by the procedure exiter.

Figure 5-6

PROCEDURE PROLOGUE

```
MAIN  
ENTRY: B    *+X(R15)          SKIP AROUND CONSTANTS  
        DC    A(FSIM)          ADDRESS OF SIMULATION DATA CSECT  
        DC    H'stack frame size'  FOR PROCEDURE CALLER  
        DC    AL1(name length, C'name'  NAME OF PROCEDURE  
        .    first  
        .  
        .    instructions
```

ALTERNATE ENTRY: same as main entry with additional instructions:

```
L      R15, =A(MAIN ENTRY)  GET BASE OF CSECT  
ST     R15, 0(R13)         UPDATE IN STACK FRAME
```

PROCEDURE EPILOGUE

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

Notes:

- 1) Main and alternate entries at least full-word aligned.
- 2) A(FSIM) is zero for assembled library routines.
- 3) High order byte of DC A(FSIM) is used as follows:

- a) Process event - in Programs and tasks.
- b) Exclusive control event - in Exclusive procedures.

5.4.2 Exiting

Procedure exits are simple. A branch instruction (Figure 5-4) transfers control to a procedure exitter (Figure 5-5). The procedure exitter, like the procedure caller, is code located in the HALSYS CSECT. It restores registers R2 - R14, leaving R0 and R1 for function value returns. This sets the stack frame pointer (R13) to the caller's stack frame and loads the return address (R14). The caller's base (R15) is loaded from his stack frame, and a branch is made to the instruction following the call, skipping the 4 bytes after the BALR instruction that contains the entry address of the called procedure.

5.5 Intrinsic

Some of the routines in the run-time library are known as intrinsic. Intrinsic have the following characteristics:

- 1) calls and returns are done directly, without the use of the procedure caller or exitter;
- 2) as a result, a new stack frame is not established;
- 3) all intrinsic are part of the HALSYS CSECT;
- 4) as a result, they are always included in the HAL load module;
- 5) entry addresses of intrinsic are not external symbols, but unique displacements from R12, which always points to the beginning of the HALSYS CSECT;
- 6) intrinsic have special register conventions.

Figure 5-7 shows an example of a system intrinsic call. Not all intrinsic use the parameter registers shown in this example.

SYSTEM INTRINSIC CALLS

LA R2, result ptr.
LA R3, left-hand arg. ptr.
LA R4, right-hand arg. ptr. (if needed).
LA R0, size
LA R1, 2nd size parameter (if needed).
BAL R14, SIDISP (R12)

SIDISP = Unique displacement for each system
intrinsic. FT1,0.50

Figure 5-7

5.6 User Written Assembly Language Subroutines

The assembly language programmer may write programs accessible to the HAL programmer. The standard macros illustrated in Figures 5-8 through 5-13 should be used to ensure compatibility with the HALLINK process and the run-time routines.

Below is a description of the use of these macros.

5.6.1 HMAIN

Purpose - to define symbolic registers R0-R10, F0-F4;
 - to set up the first few bytes of control section
 to be compatible with HALLINK.

Notes: - The label on the HMAIN must begin with an
 #, to enable the HAL compiler to reference the
 program as a COMSUB.

 - Registers R2-R10 may be freely used, because
 procedure caller restores them.

 - User programs should be re-entrant. If any
 temporaries are needed, then the user must
 declare the temporaries in a macro named
 WORKAREA and code MACRO=YES in the operand field
 of the HMAIN macro.

For example, if the programmer needed three temporaries, T1, T2, and T3, he would code:

MACRO
WORKAREA

T1	DS	F
T2	DS	XL8
T3	DS	D

MEND

#ANYNAME HMAIN MACRO=YES

5.6.2 HENTRY

Purpose: to define secondary HAL-accessible entry to assembly program.

5.6.3 HCALL

Purpose: to call another HAL-accessible program.

Notes: Registers R2-R10 will always be saved; R0, R1, F0, F2, F4 are never saved.

- no form of recursion is permitted.

5.6.4 HEXIT

Purpose: to return to caller.

Note: Values may be returned in R1 or F0; if more than one value is expected, the compiler will provide an area into which the values are to be placed. The address of this area is in R1.

5.6.5 HERRMSG

Purpose: to set up message text in format compatible with HERROR macro.

Notes: A label is required.

- One operand, the message text, enclosed in quotes is required.

Example:

```
ANYLABEL      HERRMSG      'TEXT DESIRED'
```

5.6.6 HERROR

Purpose: Simulate the HAL 'SEND ERROR'.

Notes: Takes two arguments. First is error number; Specified either as a self-defining value in range 75-100 (the user errors), or as a register number specified in parentheses. In the case where the register option is chosen, the contents of the register must be pre-loaded with a number in the range 75-100. The second is the label on the HERRMSG macro with the desired text. Again, a register may be specified in parentheses. The register must contain the address of the label of the HERRMSG macro.

Example:

```
FIRST      HERROR      80,MSG
SECOND     LA          R6, 80
           LA          R8, MSG
           HERROR     (R6), (R8)
MSG        HERRMSG     'USER TEXT'
```

```

MACRO
&L      HMAIN &MACRO=NO
STACK   DSECT
        USING STACK, 13
CURRCODE DS      A .      CURRENT CODE BASE
        DS      H .      H.O. BIT IS EXCLUSIVE FLAG
STMTNUMB DS      H .      STATEMENT NUMBER
AVAILABLE DS      D      AVAILABLE DOUBLE WORD
ERRLINK  DS      A .      ON ERROR LINKAGE
ARGO     DS      F .      CALLER'S ARGUMENTS
ARG1     DS      F
ARG2     DS      F
ARG3     DS      F
ARG4     DS      F
        DS      10F      REGS 5 - 14
        AIF     ('&MACRO' EQ 'NO').NOMAC
*        ADDITIONAL STORGAE SPACE REQUIRED FOR THIS ROUTINE
        WORKAREA
        AGO     .END
.NOMAC  ANOP
*        NO ADDITIONAL STORAGE SPACE REQUIRED
.END    ANOP
STACKEND DS      OD
STACKLEN EQU     *-STACK
*        REGISTER EQUATES
R0      EQU     0 ***NOTE*** NOT RESTORED BY HEXIT
R1      EQU     1 ***NOTE*** NOT RESTORED BY HEXIT
R2      EQU     2
R3      EQU     3
R4      EQU     4
R5      EQU     5
R6      EQU     6
R7      EQU     7
R8      EQU     8
R9      EQU     9
R10     EQU     10
R14     EQU     14
F0      EQU     0
F2      EQU     2
F4      EQU     4
*F6 MAY BE USED ONLY IF SAVED AND RESTORED
&L      CSECT
        USING  *, 15
        B      *+20      SKIP AROUND JUNK
        DC     F'0' .    ALIGN NEXT TO PROPER BOUNDARY
        DC     AL2(STACKLEN) STACK SIZE OF THIS ROUTINE
        DC     AL1(8), CL8'&L'
        MEND

```

Figure 5-8 HMAIN

```

MACRO
&L HENTRY
    DS      OF ALIGN ON FULLWORD
    ENTRY  &L
    USING  &L,15
&L B      *+20 SKIP AROUND JUNK
    DC      F'0'
    DC      AL2(STACKLEN) SIZE OF STACK
    DC      AL1(8), CL8'&L'
L    15, =A(&SYSECT) LOAD BASE OF MAIN ROUTINE
    ST      15, CURRCODE
    USING  &SYSECT, 15
MEND

```

Figure 5-9 HENTRY

```

MACRO
&L HACALL &N
&L BALR 14, 12 CALL VIA PROCEDURE CALLER
    DC      VL4(&N) . FOR NO ALIGNMENT
MEND

```

Figure 5-10 HCALL

```

MACRO
&L HEXIT
&L B      4(0,12) . RETURN
MEND

```

Figure 5-11 HEXIT

```

MACRO
&L HERRMSG &MSG
    AIF    (T'&L EQ 'O' OR T'&MSG EQ 'O').ERROR
&L DC      AL1(L'ERRH&SYSNDX)
ERRM&SYSNDX DC C&MSG
    HEXIT
.ERROR HNOTE 8, 'MACRO IGNORED. NO LABEL OR MESSAGE TEXT
MEND

```

Figure 5-12 HERRMSG

```

MACRO
&LAB      HERROR  &NUM,&MSG
          LCLB   &NEEDMVI
          LCLC   &M,&N,&L
          AIF    (N'&SYSLIST EQ 2).OK
          HNOTE 2, 'MISSING OR EXTRA OPERANDS'
          HEXIT

.OK      ANOP
&L      SETC   '&LAB'
&M      SETC   '&MSG'
&N      SETC   '&NUM'
          AIF   ('&MSG'(1,1) NE '(').TESTNUM
          CNOP  0,4
&L      ST     &MSG(1),*+12
&L      SETC   ''
&M      SETC   '0'
& NEEDMVI SETB  1
.TESTNUM AIF   ('&NUM'(1,1) NE '(').TESTMVI
&L      STC   &NUM(1),*+8
&L      SETC   ''
&N      SETC   '0'
          AGO   .BAL
.TESTMVI AIF   (NOT &NEEDMVI).BAL
          MVI   *+8,&NUM
.BAL     ANOP
&L      BAL   14,24(,12) LINK TO ERROR HANDLER
          DC   AL1(&N),AL3(&N)
          MEND

```

Figure 5-13 HERROR

6. HAL/S Characteristics Specific To The 360

6.1 Introduction

The HAL/S language as available on the IBM/360 has several implementation dependencies. These dependencies arise due to the specific hardware and software facilities available on the machine. The following subsections enumerate these characteristics.

6.2 Compile Time Characteristics

6.2.1 Character Set

The character set specified in the HAL/S Specification document is available in its entirety on the 360. The HAL/S-360 compiler will therefore recognize this full character set. The internal coding scheme for the characters is EBCDIC. No other coding scheme is recognized.

6.2.2 Internal Table Capacities

- a) The maximum number of symbol table entries is 1000. The symbol table is filled with the names of user-defined variables and labels as the input source is scanned.
- b) A maximum of 32767 literals may be used in a single compilation.
- c) The maximum number of characters permitted in the sum of all character literals in one compilation is 3000.
- d) Storage for replace text is limited to 1000 characters. However, multiple blanks within such text are stored in a compressed form.

6.2.3 Data Type Size Limitations

- a) Arrays are limited to 3 dimensions.
- b) Each dimension in an array is limited to the range 1 to 32767.
- c) The maximum row or column dimension of a matrix is 64.

- d) The maximum dimension of a vector is 64.
- e) Character strings are limited to 255 characters.
- f) Bit strings may not be over 32 bits long.
- g) Single precision integers are 16-bit signed quantities.
- h) Double precision integers are 32-bit signed quantities.
- i) Single precision scalar values are represented internally in the standard 360 floating point format using 32 bits (1 sign, 7 exponent, 24 mantissa).
- j) Double precision scalar values are represented internally in the standard 360 double precision format using 64 bits (1 sign, 7 exponent, 56 mantissa).
- k) Individual characters within character strings are represented internally as the 8-bit EBCDIC pattern for the character. Hence, the result of converting a single character to a bit string by means of the BIT conversion function is an 8-bit string corresponding to the EBCDIC bit pattern for the character.

6.2.4 Program Organization Limits

- a) The number of external names allowed in any one compilation is 100. An external name is defined as any user-defined or built-in function which causes an external reference to be satisfied by the link editor.
- b) Function invocation (whether built-in or user-defined) may not be nested more than 10 levels deep.
- c) DO groups may not be nested to a depth exceeding 15 levels.
- d) The total number of elements in initial lists in any one compilation is limited to 32767.

6.2.5 Input/Output Statements

- a) Device numbers in input or output statements are limited to the range 0-9.

6.2.6 Program Naming Convention

Each successful HAL/S compilation produces at least one named control section (CSECT) and one or more unnamed control sections (PCs). The CSECT name is derived according to the following rules:

- a) HAL/S compilation unit names are transferred to the emitted object code by using only the first seven characters of the HAL/S name. The name will be padded or truncated to seven characters where necessary. In the case of a CSECT corresponding to a TASK, the name is truncated to five characters. Care should be exercised in naming to insure that only unique names will be generated for a given compilation.
- b) Any occurrence of the underscore character (`_`) in the first 7 characters of a PROGRAM, PROCEDURE, FUNCTION, TASK, or COMPOOL is eliminated. The resulting characters are joined together to produce the characteristic name of the compilation unit (e.g. `A_B_C` becomes `ABC`). An additional character is placed on the front of the resultant name to form the final name for each of the individual situations in which the name is used. These additional names for internal control sections are:

`$` - the executable code control section name for a PROGRAM

`#` - the executable code control section name for a COMPOOL or comsub, a nested FUNCTION, or a nested PROCEDURE.

`@` - the control section used by a PROGRAM or TASK for a temporary work area

`$nn` - assigned to TASKs within a PROGRAM but using only the first five characters of the characteristic name, where `nn` is a two digit number assigned uniquely to each TASK in a single compilation unit. "`nn`" is 01 for the first TASK defined within a compilation; 02 for the second; etc. Note that the name of a CSECT corresponding to a TASK is

derived from the PROGRAM in which it is defined, not from the label of the TASK declaration itself.

- c) A separate CSECT called FIRSTPGM is generated for each compilation which is a PROGRAM. Its contents in 360 Assembler language are as follows (the name of the HAL program in this example is HALPROG):

```
FIRSTPGM                CESECT
                        EXTRN HALSTART
                        DC V(HALPROG)
                        END HALSTART
```

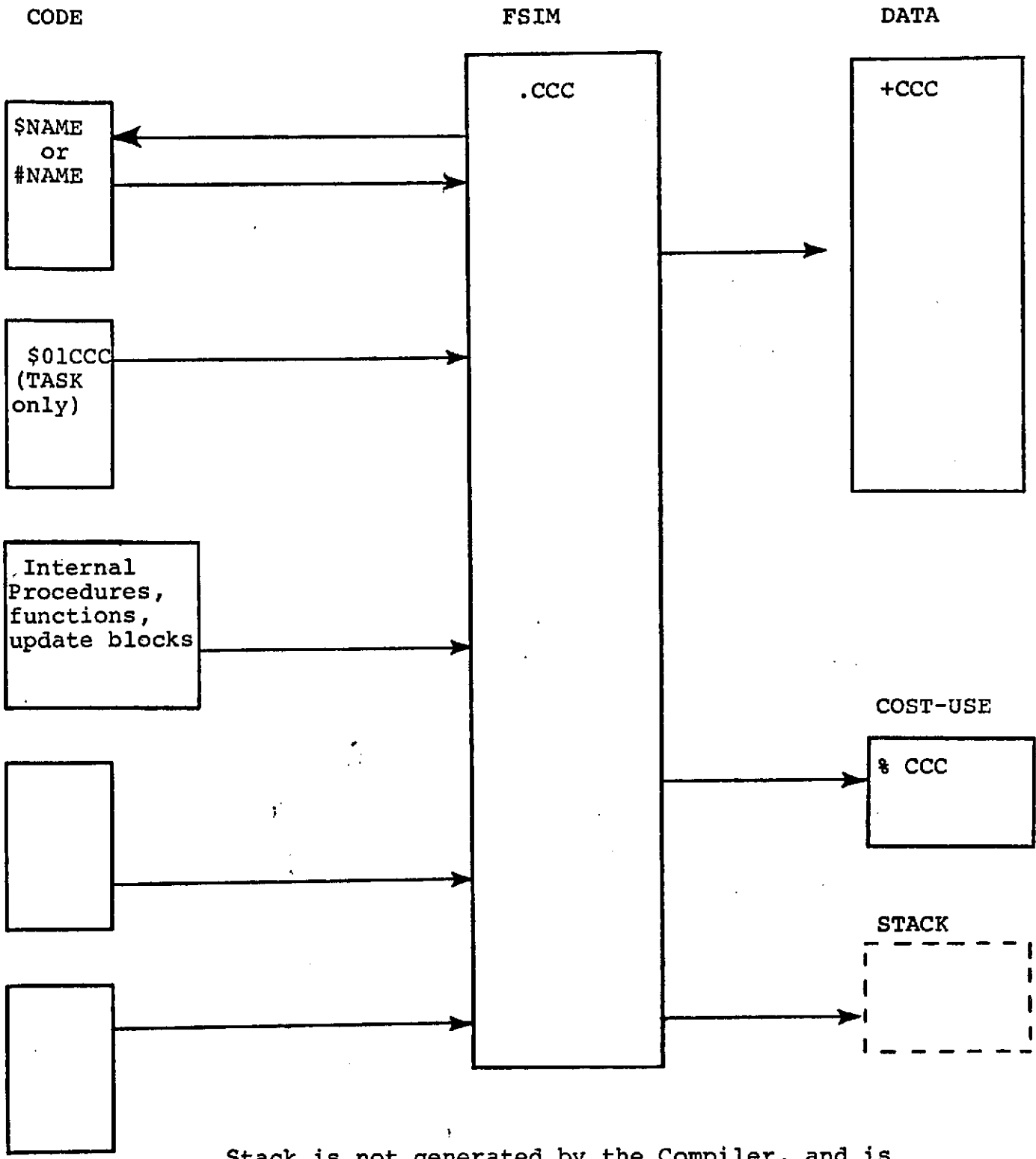
This establishes the link between the HAL set up routines and the HAL program.

In addition to control section names, the characteristic name of a compilation unit may appear in certain external contexts preceded by the following characters:

- @ - the member name of the template created for a compilation unit
- # - the member name of the simulation data file created for a compilation unit

Each PROCEDURE or FUNCTION declared internal to the main compilation unit results in a Private Control Section (PC) containing the executable code.

- d) Each compilation unit has associated with it a PC containing all the literals used in the program, as well as address constants needed by the program. If the compilation unit is a PROGRAM, there is an address constant referring to the stack associated with the PROGRAM. All TASKS also have stacks associated with them, and their address constants are also in this PC. The name of the stack associated with a PROGRAM or TASK is obtained by replacing the \$ in the CSECT name with an @.
- f) All STATIC variables declared within a compilation unit are allocated space in a separate PC. If no variables are defined within a compilation unit, a PC of zero bytes may or may not be generated depending upon the implementation. AUTOMATIC variables defined in



Stack is not generated by the Compiler, and is present only for PROGRAMS. In addition, a separate stack exists for each TASK.

Figure 6-1

C-2

A typical example:

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

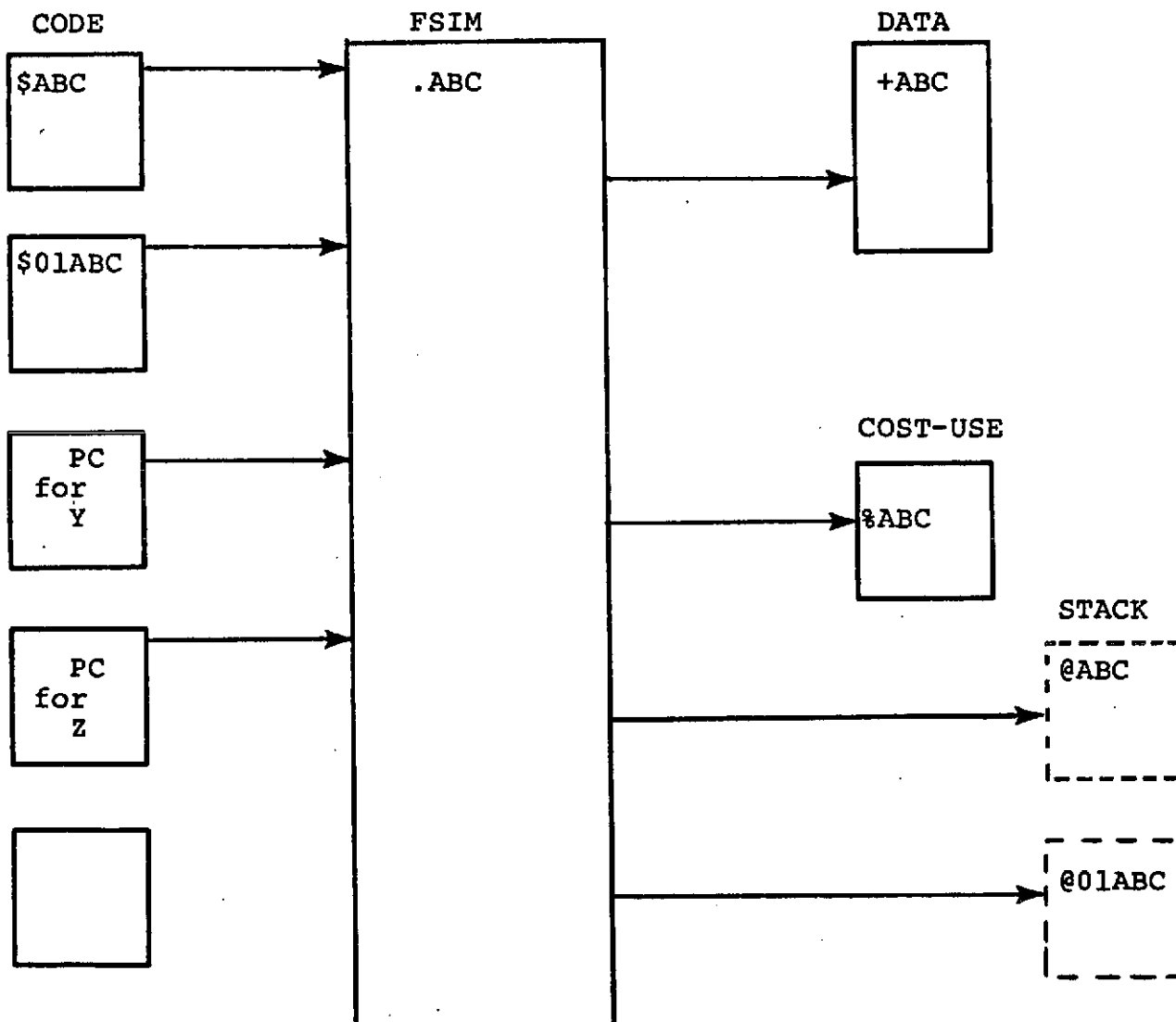
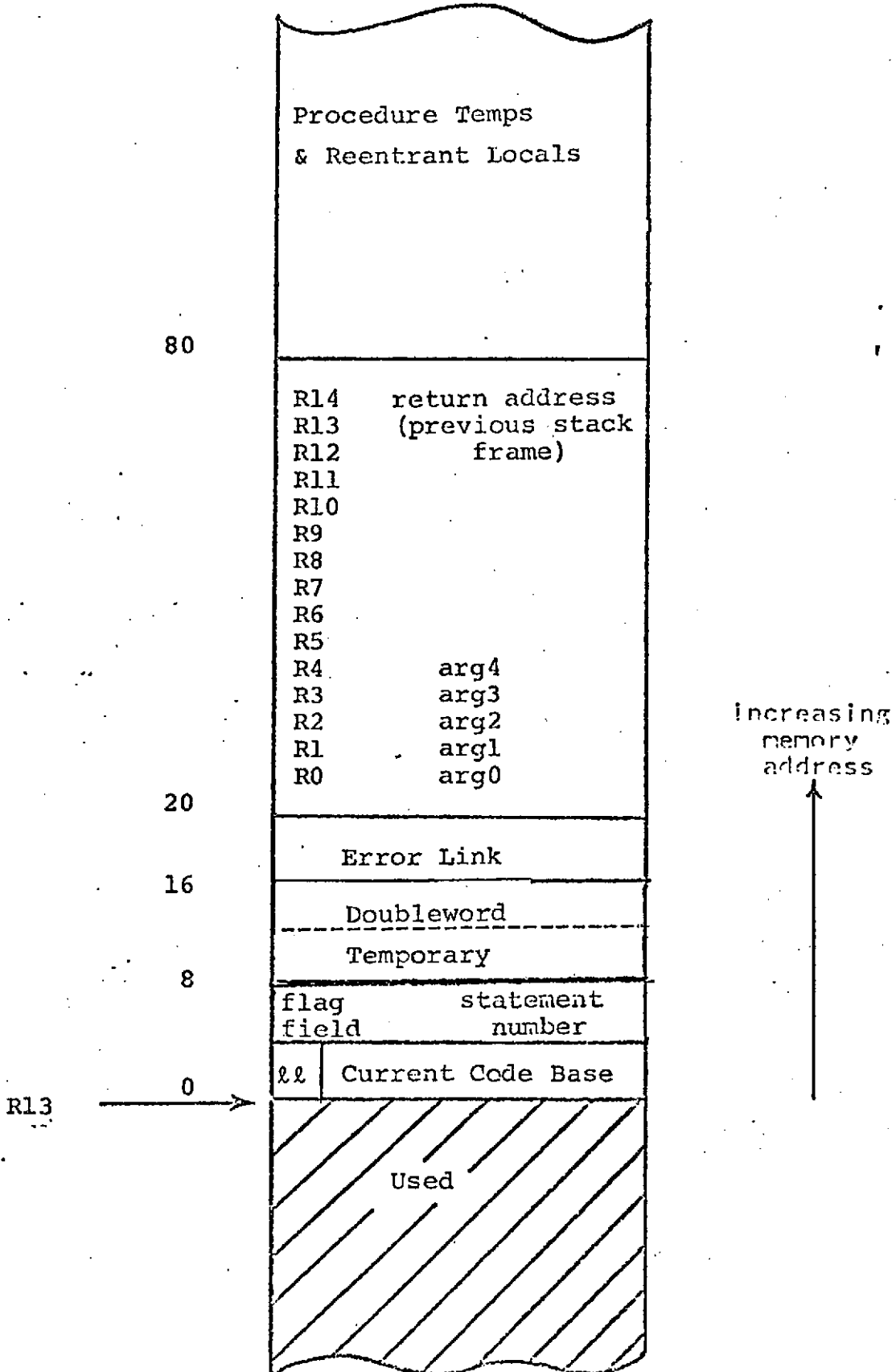


Figure 6-2

STACK LAYOUT



NOTE: High order bit of flag field is "1" if exclusive procedure; other 15 bits "0" (reserved).

Fig. 6-3 99<

REENTRANT PROCEDURES are allocated in the stack, not in this PC.

Refer to Figure 6-1 and 6-2 for examples of typical layouts of generated object decks.

Figure 6-3 indicates the logical relationships between the CSECTs and PCs generated for the following PROGRAM. An arrow (A→B) indicates reference in A to B:

6.2.7 The INCLUDE Compiler Directive

Use of the INCLUDE compiler directive to access a symbolic library has been implemented in the following way. The name found on the INCLUDE directive such as:

D INCLUDE COMPOOLA

will be used to search the symbolic library. The name must be a 1 to 8 character string.

The symbolic library must be defined by the following JCL:

```
//HAL.INCLUDE DD DSN=<library name>,<other parameters>
```

The <library name> must be the data set name of the library. The library data set must have partitioned organization and must be referred to in the JCL by data set name only (DSN=HAL.LIB), not as data set and member name (DSN=HAL.LIB(COMPOOLA)).

The library thus defined is searched for a member whose name is exactly the name found on the INCLUDE directive. The library must contain card images of the same size as the primary input, but may be blocked in any legal manner. Two or more libraries may be concatenated in the JCL if they conform to the standard JCL rules for for such concatenation.

If an INCLUDE request cannot be satisfied by referencing the INCLUDE DD card, an attempt will be made to find the member by using the OUTPUT6 DD card. This search technique allows multiple-compilation jobs to reference template libraries created automatically in earlier steps without the need to indicate an INCLUDE DD card.

PREVIOUS PAGE BLANK NOT FLIPPED

6.2.8 ACCESS Rights Implementation

The HAL/S language allows managerial restrictions to be placed upon the usage of user-defined variables and external routines. The existence of such a restriction is indicated by the use of the ACCESS attribute as described in the HAL/S Language Specification. The manner in which the restrictions are enforced in the HAL/S-360 compiler system is described below.

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

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

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

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

where the <PAF name> is the dataset name of the PAF without any member specification. Note that a PAF need not be defined if the access control facilities of the HAL/S compiler are not being used.

Each member of the PAF contains the information about ACCESS controlled names which are to be available to one unit of compilation. The member name is defined by a Program Identification Name (PIN). The PIN is specified to the HAL/S-360 compiler by using the PROGRAM compiler directive in the primary input stream:

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

The <id> field of the directive is a 1 to 8 character identifying name which is used to select the member of the PAF to be processed for the current compilation's ACCESS information. The appearance of the PROGRAM directive in the compiler's input stream causes immediate processing of the PAF member specified. Therefore, the PROGRAM directive must follow any COMPOOL or COMSUB templates which may specify ACCESS-controlled data. Also, the PROGRAM directive must appear before the definition of the block which is the primary unit of compilation. In general, the input stream seen by the compiler should look like the following:

102<

6-7

```

    <COMPOOL or COMSUB template>
      .
      .
      .
    <COMPOOL or COMSUB template>
    D PROGRAM ID=<id>
    M PROG NAME: PROGRAM;
      .
      .
      .

```

The format of an individual PAF member is described below.

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

```
<COMPOOL-name>(<var-name>,<var-name>, ... <var-name>)
```

or

```
<COMPOOL-name>($ALL)
```

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

- d) Access to external block names is specified as:


```
$BLOCK(<ext-name>, <ext-name>, ... <ext-name>)
```
- e) Blanks are allowed anywhere in the record except that names may not be broken by a blank.
- f) Either of the constructions (c) or (d) above may span more than one record.
- g) The name of the particular COMPOOL in the form (c) above may appear more than once; i.e. the variables in a particular COMPOOL do not have to be specified at one time. Similarly, the form \$BLOCK may appear more than once.

Some validity checking is performed by the compiler

while processing the PAF member. Warnings are issued for the following conditions:

- 1) A syntax error on a PAF record - the bad record is printed;
- 2) Names mentioned in the PAF are not defined;
- 3) Elements of \$BLOCK in the PAF are not block names;
- 4) Requests for names which are not ACCESS protected;
- 5) Variables found, but not within the COMPOOL specified; 6) Names used in the context of a COMPOOL-name which are not COMPOOLS.
- 7) Elements of an ACCESS-protected COMPOOL block are mentioned in the PAF before the COMPOOL block itself is freed for use.

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

6.2.9 Template Generation

If a HAL/S compilation contains invocations of external PROGRAMS, procedures, functions ("COMSUBS") or references to variables in an external COMPOOL, the block templates for those external blocks must appear in the compilation (see Section 2.4). The HAL/S-360 compiler features automatic generation of block templates during the compilation of these blocks. Compilations which later require use of such templates can retrieve them by means of the INCLUDE compiler directive. The INCLUDE directive causes any desired HAL/S source text to be taken from a specified member of a PDS rather than from the regular input stream. It is recommended that block templates be maintained on a different dataset from that used for other kinds of INCLUDE text. If block templates are maintained on a PDS called <templib> and other material on a PDS called <otherlib>, then the appropriate JCL for such a compilation is:

```
//HAL.INCLUDE DD DSN=<templib>,...(other parameters)
//           DD DSN=<otherlib>, ... (other parameters)
```

For the compilation of any compool, or external procedure or function block, the following JCL must be specified:

```
//HAL.INCLUDE DD DSN=<templib>,...(other parameters)
//           DD DSID=<otherlib>,...(other parameters)
//HAL.OUTPUT6 DD DSN=<templib>,...(other parameters)
```

During compilation, the following events take place:

The HAL/S compiler ascertains whether or not a block template for the block already exists in the dataset referenced by the INCLUDE DD cards. The search algorithm is applied in the same way as for the INCLUDE compiler directives (see 6.2.7).

If no such template exists, then it is added as a new member of <templib>. The compiler emits the following message in the output listing:

```
*****TEMPLATE LIBRARY MEMBER <membername> NOT FOUND - ADDED
```

If the template already exists, and the new template is different from the old, the member of <templib> is replaced and the compiler emits the message:

```
*****TEMPLATE LIBRARY MEMBER <membername> FOUND AND CHANGED
```

If the new template is the same as the old, then replacement does not take place and the compiler emits the message:

```
*****TEMPLATE LIBRARY MEMBER <membername> FOUND - CHANGE
                                         NOT REQUIRED
```

Any error discovered during Phase I of the compilation will inhibit the adding or replacement of the member of <templib>, and the compiler emits the message:

```
*****COMPILATION ERRORS INHIBITED TEMPLATE GENERATION
```

Each template has attached to it a version number between 01 and 255. On first generation, the version number assigned is 01. Every time the template is updated in <templib>, the version number is incremented by 1. If the version number exceeds 255, it is reset to 01, and thereafter begins incrementing again.

The format of a template as maintained in the template library is exactly as described in the Language Specification. It is followed by a VERSION directive card specifying its version number:

Example:

```
INC: PROCEDURE;  
CLOSE;  
D VERSION @
```

@ Is a generally unreadable character whose hexadecimal value is the version number.

The <membername> of a block template is formed by taking the characteristic name of the compilation unit (see Section 6.2.6) and prefixing it with the character .

Note that this naming convention determines the form of the INCLUDE directive which can be used to retrieve the block template; e.g. compilation unit A_B_C produced a block template as member ABC which must be specified as ABC on an INCLUDE directive in order to find the proper template.

7. HAL/S-360 Input/Output Operations

7.1 HAL/S-360 FILE I/O

This section presents a detailed description of the FILE I/O implementation for HAL/S-360 release 7. The implementation is consistent with the current Language Specification. It allows for three different types of files, giving the user the freedom to choose the type most appropriate for the application. The types cover a range of characteristics, from the most efficient but least flexible (Type I) to the least efficient but most flexible (Type III). If the user does not choose a type, the default is Type III.

Familiarity with the Language Specification for FILE I/O is assumed.

7.2 File Type Characteristics

A FILE I/O statement results in an unformatted core image transfer between a contiguous data area in memory and a physical block in a direct access data set, hereafter referred to as a file. Blocks may be read, written, or updated in any desired order. The OS/360 BDAM (Basic Direct Access Method) routines are used. The distinguishing characteristics of each type of file are described in the following paragraphs. The descriptions use the following concepts:

Block length: The number of bytes in the block. Both the input and output file expressions make a request for a block transfer of a specific number of bytes, determined by the size of the variable or expression on the opposite side of the assignment (=) operator. This size must be consistent with (and in some cases automatically determines) the block length attribute of the file (see Sec. 7.3.4). The maximum size block possible is 32,760 bytes

B: Block identification number (an integral value) supplied by the user in the file expression FILE

(F,B). (F is the file number).

Bmax: The number of blocks or dummy blocks in a Type I or II file, fixed at initialization time and determined by the block length and the number of tracks allocated to the file.

Track: An OS/360 Direct Access Storage Device (DASD) concept. It is the smallest unit of DASD space allocation. Track capacity varies with different device types. The track capacity of a 2314 disk, for example, is 7294 bytes.

7.2.1 Type I: Dense Fixed-length Blocks

Block length: Every block is the same length.

Block identification: B is the sequentially relative block number in the data set and directly determines the location of the block in the file.

Valid range of B: $0 \leq B \leq B_{max} - 1$. If the file contains 200 blocks, then $0 \leq B \leq 199$. $B_{max} \leq 2^{24} - 1$ (16, 777, 215).

Block existence: All blocks in the valid range exist in an initialized file, whether or not an output file statement merely updates the identified block. An input file statement reading a block not written by an output file statement assigns binary zeroes to the variable (this may not be appropriate for some applications, or for certain data types - see Sec. 7.5)

Block access time: Block access time is less (faster) than for Types II or III.

Space utilization: Space for a block in the valid range is guaranteed, since the blocks already exist. DASD space utilization is good only if most values of B in the valid range are used for real blocks. DASD space utilization will be poor if the range of values used for B is sparse.

7.2.2 Type II: Sparse Fixed-length Blocks with Keys

Block length: Every block is the same length.

Block identification: B is the hardware key of the block. The location and sequence of blocks is not directly a function of B, so that a search for the block using B as the key must be made each time it is accessed.

Valid range of B: B is translated to EBCDIC numeric characters for the key, so that the valid range of B is $0 \leq B \leq 10^{**n-1}$, where n=number of characters in the key (see KEYLEN in the DCB parameters 7.3.4). If KEYLEN=8, then $0 \leq B \leq 99999999$. Since B is communicated as an integer, the upper limit is 2^{**31-1} (2, 147, 483, 647) for double precision, or 2^{**15-1} (32, 767) for single precision.

Block existence: A block exists only if it has been written. Available space in an initialized file consists of dummy blocks (high-order byte of key is X'FF'). The initial number of dummy blocks (Bmax) limits only the total number of real blocks, not the maximum value for B. Dummy blocks are overwritten with real blocks (with key=B) when a block is written for the first time. Existing blocks are updated in place when a block is re-written. An attempt to input a dummy block will result in a run-time error.

Block access time: Block access time is slower than Type I because of the hardware search for the block by the key B. The time taken depends on the DCB LIMCT parameter and other factors (see Sec. 7.3.4 and Appendix J). Writing a block for the first time takes at least twice as long as an update of an existing block because it occurs only after a search for a non-existent block fails.

Space utilization: An attempt is made to spread the blocks evenly throughout the file. A run-time error of "no space in file" occurs when available space on a track or sequence of tracks is exhausted. This may occur even though there is available space elsewhere in the file, unless OPTCD=E is specified. See Appendix J for a description of the block location

and search algorithms for sparse files (Types II and III). DASD space utilization is good if the values of B are uniformly sparse.

7.2.3 Type III: Sparse Differing-length Blocks with Keys

Block length: Blocks may have different lengths. The same block may not be re-written with a different length, except in the case indicated below.

Block identification: Same as for Type II.

Valid range of B: Same as for Type II.

Block existence: A block exists only if it has been written. Available space in an initialized file is recorded in capacity records, one per track. A new block is written in the available space of a track, and the capacity record for that track is updated to reflect the reduction in available space. When a block is re-written, the old block is updated in place if the new block has the same length; otherwise a length error occurs. If OPTCD=F was specified, and the standard fixup (ON ERROR SYSTEM or IGNORE) is taken, then the old block is deleted and a new block with the new length is written in available space. The old block and the space it occupies remain unavailable. An attempt to input an unwritten block will result in a run-time error.

Block access time: Same as for Type II.

Space utilization: Same as for Type II.

7.3 Selecting a File Type

7.3.1 Choosing a File Type

A number of factors should be taken into consideration when choosing a file type. The primary factor is whether or not blocks in the same file need to be of different lengths. If so, then Type III is the only choice (barring user control over the "block length" error

- see Sec. 7.4), since different length blocks are not possible in file Types I and II.

Another factor is the range of values for B. If it is known that the maximum value of B will be less than or equal to the number of possible blocks in the file (determined only by the size of the file), and that most values of B in that range will be used as a real block identification, then Type I is the best choice (most efficient). On the other hand, if the range of values of B is sparse (for example, B=100, 200, 300..etc), then Type II or III should be used.

Another factor may be that in a Type I file any block in the valid range exists; that is, it may be input without ever having been output. If the occurrence of the "block not found" error is needed for proper processing in a certain application, then a Type I file cannot be used, unless the program can test for and recognize a block of binary zeroes as an invalid block.

7.3.2 Specifying a File Type

The type of a particular file is determined only by its DCB parameters. DCB parameters may be specified by the user on the DD card for a new file, or they are supplied automatically by the operating system from the dataset label (DSCB) of an existing file.

A file must be initialized for a particular type before it can be used. Initialization causes the file to be filled with available space of the proper format and destroys any data existing previously in the file. The initialization may be performed in a single job or job step from the HAL program execution, or it may be automatically initialized by HAL before its first use. Automatic initialization is performed only if a file is created and used in the same job step (i.e. if the DD card specifies DISP=NEW). In any case, the DCB parameters supplied at initialization time on the DD card determine the file type. Defaults for omitted DCB parameters are indicated below. If no DCB parameters are supplied, the defaults cause initialization of a Type III file, the most flexible but least efficient.

7.3.3 DCB Parameters by File Type

These DCB parameters are the ones that determine the file type.

7.3.3.1 Type I

RECFM=F or FT
and
KEYLEN=0 or omitted

7.3.3.2 Type II

RECFM=F or FT
and
KEYLEN>0

7.3.3.3 Type III

RECFM=V, VS, or U
and
KEYLEN>0
DEFAULT: RECFM=U, KEYLEN=8

7.3.4 Summary by DCB Parameter

Options and defaults specific to a file type are so indicated.

BLKSIZE=number: This specifies the block length for Type I or II, or the maximum block length for Type III. It must be less than or equal to the track capacity of the DASD device unless RECFM=FT or VS is specified, in which case the the maximum is 32, 760. If RECFM=V or VS, the BLKSIZE should be 4 more than the largest block length.

DEFAULT: Types I and II: length of first block transfer;
Type III: maximum blocksize of the device.

DSORG=DA: This identifies the dataset organization of the file as direct access. Although it is not required for HAL FILE I/O, it should be specified if the file is to become permanent and be manipulated by OS utilities. The OS utility program IEHMOVE requires this identification to successfully copy a file.

KEYLEN=number: This specifies the number of characters in the key for file types II and III. It must be in the range 1-16. If KEYLEN=1, note that only 10 blocks may exist in the file no matter how big it is, since it would limit the the range of keys to 0-9. When B is converted to a key, it must fit in a field of n decimal digits, where KEYLEN=n, or an error occurs.

DEFAULT: KEYLEN=8 (Type III only).

LIMCT=number: This specifies the number of tracks (or blocks, if OPTCD=R) to be searched in a Type II or III file when attempting to read or update an existing block. It is also used when searching for available space for writing a new block. This parameter is ignored unless the extended search option is used (OPTCD=E). See Appendix J for block location and search algorithms.

DEFAULT: LIMCT=number of tracks (or blocks) in the file (i.e. the entire file is searched).

NOTE: This parameter is not a "remembered" attribute of the file. It must be re-specified (or altered) each time

the file is used.

OPTCD=E,F,R,ER, or EF: This parameter specifies various options.

OPTCD=E: extended search (Types I and II) This specifies the extended search option. See Appendix J.

OPTCD=F: feedback (Type III only) This option causes the standard fixup for the "block length mismatch" error in a Type III file to rewrite the block with its new length. Its absence prevents changing the length of an existing block.

OPTCD=R: relative block addressing (Types I and II only) This option is required and is defaulted to for a Type I file. In a Type II file, it causes a different record location and search algorithm to be used.

DEFAULT: OPTCD=R (Type I only)

RECFM=F,FT: Type I or II; =U,V,VS: Type III only. This specifies the block (record) format. RECFM=VS (variable spanned) must be specified if the maximum block length (BLKSIZE) is greater than the track capacity in a Type III file. RECFM=FT (fixed with track overflow) should be specified for the same case in a Type I or II file, and also when the ratio of block length to track capacity is such that there would be a serious waste of DASD space. (Without track overflow only integral numbers of whole blocks are placed on a track.) Track overflow allows a fixed block to be broken up across a track boundary, resulting in maximum utilization of track space. RECFM=FT can be specified only if the hardware track overflow option exists for the device.

DEFAULT: RECFM=U

7.4 Run Time Errors

The following errors may occur at run-time when executing a FILE I/O statement.

ERROR 31 - SYNAD ERROR: message

An uncorrectable I/O error occurred, described by "message". An uninitialized file may be the cause. The "message" is generated by the system macro SYNADAF.
SYSTEM action: limited fixups.

ERROR 32 - MISSING DD CARD - FILEn

FILE I/O was attempted without a defining DD card.
SYSTEM action: terminate.
Standard fixup: The I/O operation is ignored. NOTE: Errors 31 and 32 may also occur in sequential I/O.

ERROR 64: NO SPACE IN FILE

An attempt was made to add a new block to a Type II or III file, but available space was exhausted. To correct this error, re-run the job with a new larger file, or with the existing file and OPTCD=E.
SYSTEM action: terminate.
Standard fixup: The output FILE statement is ignored.

ERROR 65 - BLOCK LENGTH MISMATCH

The block in the file and the block in memory have different lengths. The occurrence of this error depends on the file type and whether input or output is requested. An input file statement will never read more data than requested. Even if the block in the file is larger than the variable being filled.

SYSTEM action: unlimited errors.
Standard fixup: For input, the number of bytes read is the minimum of the two lengths. For output, the block in the file is truncated or padded with binary zeroes to force it to the proper length. If the error occurs for an existing block in a Type III file and OPTCD=F is specified, the standard fixup will rewrite the block with a new length, deleting the old block.

Type I and II:

The error may occur on input, output, and update.

Type III, RECFM=U:

The error may only occur when updating an existing block.

Type III, RECFM=V or VS:

The error may occur on input or output.

ERROR 66 - BLOCK NUMBER OUT OF RANGE

The block number supplied is invalid. It is either negative or greater than the limit (Bmax-1 for Type I; $10^{**}n-1$, KEYLEN=n for Type II and III).

SYSTEM action: terminate.
Standard fixup: The I/O is ignored.

ERROR 70 to 79 - BLOCK bbbbbbbbbb NOT FOUND

Analogous to sequential I/O end of file errors 40 to 49, this error occurs in a Type II or III file when the specified block number is legal but the block does not exist. Errors 70 to 79 correspond to files 0 to 9 respectively.

7.5 HAL Data Type Considerations

The user should be aware of the implications of using FILE I/O with different data types. A block in a file is a binary core image and contains no type information. No problem exists if a block is written from and read into data of the same type and length.

If a block is written from one data type and read into another, no error indication is given, assuming that the lengths are the same. However, subsequent use of the data may cause errors ranging from the mild to the severe. The following lists by data type the kind of trouble that could be caused by reading in a block with arbitrary binary information.

Bit strings: none

Integers: none

Scalars, Vectors, and Matrices: Underflow errors may occur due to unnormalized floating point numbers.

Character strings: DECLARED maximum length may be altered. The first two bytes are the maximum and actual lengths respectively. If the maximum length is increased in this manner, subsequent assignment into the character variable could destroy the data following it.

Name variables: Although explicit use of name variables is disallowed in FILE I/O statements, they may be input and output if imbedded in a structure. Since name variables contain machine addresses, unpredictable results may occur, including ABENDs. The same holds true, even when strict type matching is observed, if a block is written in one job or job step and read in another.

7.6 OS/360 Considerations

7.6.1 DD Cards

One DD card is needed to define each file referred to in a HAL program. For example, when file 3 is used, i.e. when the expression FILE(3,<arith exp>) appears in a HAL program, the minimum DD card required is:

```
//FILE3 DD UNIT=SYSDA,SPACE=(TRK,number)
```

where "number" defines the size of the file in tracks. This results in a temporary Type III file which gets deleted at the end of the job step.

The general format of the DD card for a temporary file is:

```
//FILEn DD UNIT=unit,VOL=volume,SPACE=space,DCB=(list of DCB parameters)
```

where n is the file number (0-9). The "unit", "volume", and "space" values may be determined from a JCL manual. The DCB parameters are covered in section 7.3.4.

If a permanent file is desired, the DSN and DISP parameters must also be specified. If the file is being created, then DISP=(NEW,KEEP) or DISP=(NEW,CATLG) is needed. If an existing file is used, then DISP=OLD or DISP=SHR is needed.

7.6.2 Separate Initialization of a File

Although HAL will automatically initialize a new file, circumstances may require separate initialization. The routine in the run-time library that does the automatic initialization may be invoked as an independent utility program. The following JCL illustrates the use of this utility:

```
//INIT EXEC PGM=FORMATDA,PARM=anydd  
//STEPLIB DD DSN=HALS.RUNLIB,DISP=SHR  
//anydd DD ... appropriate parameters
```

If it is desired to re-initialize an existing file, then PARM=anydd must be changed to PARM='*anydd'.

Return Codes

0:	successful initialization
4:	not formatted - DISP=OLD
8:	missind DD card specified in parm field
16:	no parm field specified

7.7 Execution Time Characteristics

7.7.1 Input/Output

a) The I/O device defined to be the error message channel (CHANNEL6 is the default) is forced to have record format FA or FBA.

b) Numeric formatting of output items is done as follows:

integers - an 11-character field is printed with the number right justified. A floating minus sign is added if the number is negative.

single
precision

scalars - a 14-character field is printed as follows:

sx.xxxxxxxE+txx

where s is a blank or minus sign;
x is a single digit 0 to 9;
t is a plus or minus sign;

double
precision

scalars - a 23-character field is generated as follows:

sx.xxxxxxxxxxxxxxxxxxExx

c) If not specified through JCL, the HAL/S I/O routines will assume a logical record length of 80 for unpagged output devices and a logical record length of 133 for pagged output devices. The first character of the 133 character pagged output record is assumed to be an ASA carriage control character.

7.8 User-Defined Execution Time JCL

The user may require additional execution time JCL to service his program's requirements:

Additional HAL/S data sets (other than the CHANNEL5 and CHANNEL6 supplied in the procedure). These are data sets with names CHANNELn where n is in the range 0 to 9. These data sets are referenced by HAL/S I/O statements specifying the proper n as their device number (e.g. WRITE(8) references DD card CHANNEL8).

The HAL/S I/O routines make some assumptions about the characteristics of the data sets which are allocated to the various CHANNELn DD cards. These assumptions, along with a description of devices and organizations supported by HAL/S are detailed below.

The user may supply any, all, or none of the DCB attributes for CHANNELn DD cards. The record formats which are acceptable to HAL/S are:

F FB FA FBA FBSA

V VB VA VBA

U UA

Machine carriage control is acceptable but subject to interpretation described below.

7.8.1 General Rules Used by HAL/S to Create DCB Attributes

- a. BLKSIZE not supplied (by JCL or data set).
The maximum block size of the particular device is found. The BLKSIZE is set to the largest multiple of LRECL which is less than or equal to this block size. Note that for tapes this maximum size is 32767 bytes which would require a sizeable buffer area to be taken out of main storage.
- b. LRECL not supplied. For PRINT files the LRECL defaults to 133. For non-PRINT files the default is 80.
- c. RECFM not supplied. For PRINT files the RECFM

defaults to FBA. For non-PRINT files the default is FB.

7.8.2 General Rules Governing HAL/S Sequential Output

- a. If the mode is PRINT and the DCBRECFM specifies "A", carriage control characters will be automatically generated. For "H" on PAGED devices, it is the user's responsibility to see that the first character of each output line has the proper control characters. For UNPAGED devices, control characters are not allowed.
- b. Variable length record specification will cause records to be written in variable format. However, the records will actually be all the same length (LRECL).
- c. Format U records (undefined record format) will be written in the proper form, but all records will be the same length (LRECL).

7.8.3 General Rules Governing HAL/S Sequential Input

- a. Carriage control characters encountered during input will be available to the programmer; i.e. scanning of the input will begin at the carriage control character.
- b. Variable length records may be read. The 4 byte descriptor field of each record will not be available to the programmer. The effective length of a variable record will be "length read minus 4" (subject to further modification due to carriage control).

APPENDIX A.

Compile-Time JCL Options

The following is a list of options which may be coded in the "OPTION=" field of the JCL invoking the HAL/S compiler.

Type 1 options are ones which are specified by the occurrence of a keyword in the PARM field. The keyword may be preceded by the letters "NO" to indicate that the option is not in effect. The default value assumed by the compiler if a keyword is not specified is listed first.

Type 2 options are ones which have "values" associated with them. The default values are shown.

Type 1 Options

- | | |
|-------------------------|--|
| NOLISTING2/
LISTING2 | - Causes unformatted source listing to be generated |
| NODUMP/DUMP | - Requests the compiler to produce a memory dump if certain internal compiler errors occur |
| NOLIST/LIST | - Produces an assembly listing from Phase II of the compiler |
| TRACE/NOTRACE | - Causes the generation of a link to the HSS end-of-statement routine in the object module. Enables Real Time execution and debugging. |
| NODECK/DECK | - Controls production of an additional object deck on the OUTPUT4 DD card |
| NOTABLST/
TABLST | - Causes Phase III of the compiler to produce formatted dump to the simulation data file (SDF). |
| NOSRN/SRN | - 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 |

- TABLES/
NOTABLES - Controls generation of
 Simulation Data Files

- NOADDRS/
ADDRS - Indicates the presence of
 statement address information
 in the Simulation Data Files

Type 2 Options

- PAGES=250 - Sets the maximum page number to be
 allowed in generation of the primary
 compilation listing

- LINECT=59 - Sets the maximum number of lines
 which will be printed on any one page of
 either the primary or secondary source
 listing

- TITLE= ... - Specifies 1 to 60 characters used
 by the compiler when printing header
 information at the top of each page of
 the listing.

In addition to those options described above, certain others exist to support operation of the SDL. These options include:

Type I Options

- NOSDL/SDL - Informs the compiler that it is
 operating within the SDL. Certain
 actions are then taken, actions dif-
 ferent from those taken during Stand-
 Alone operation. An example of such
 action is inclusion of extra information
 in the primary source listing to allow
 tracing of source updates.

APPENDIX B.

Execution-Time JCL Options

The following is a list of options which may be coded in the "RUNPARM=" field of the JCL invoking the HAL/S compiler. This information is directed to the "PARM" field of the execution step.

- TAB=n : Specifies the automatic spacing between elements on output. The default value is 5.
- LINES=n : Specifies the number of lines per page of paged output. The default value is 55 lines.
- MCHAN=n : Specifies the channel number (DD card CHANNELn) on which error and trace messages appear. The valid range is 0-9; CHANNEL6 is the default.
- TRACE=0 : Specifies no tracing (default option).
- TRACE=1 : Statement tracing;
(Note that this option is valid only if 'TRACE' was specified on the compilation).
- TRACE=2 : Procedure & function call and return tracing.
- TRACE=3 : Statement, procedure & function call, and return tracing.
- TRACE=4 : Executive tracing (combined with other trace options by addition).
- DUMP=0 : No HAL/S variables dumped.
- DUMP=1 : HAL/S variables dumped on ABEND (default option).

DUMP=2 : same as DUMP=1 but also on normal end;
 (Note that DD cards HALSYMB and HALDUMP are required).

MSGLEVEL=0 : no backtrace on normal termination;
 (This is the default value).

MSGLEVEL=1 : backtrace occurs on normal termination.

ERRORLIM=n : specifies the maximum number of execution time
 errors allowed . The default value is 10. The
 maximum is 254; if 255 is specified, the SYSTEM
 action will be UNLIMITED.

SIMTIME=n : number specifying the starting time in
 seconds of the RUNTIME simulation clock (DEFAULT:SIMTIME=

SPEED=n : number specifying the number of machine
 cycles per second to be used in simulated execution
 time (DEFAULT:SPEED=500,000.0).

PCBS=n : whole number specifying the maximum number of
 programs and tasks in the run (DEFAULT:PCBS=10).

FIRSTPGM=
 <name> : full <name> (underscores included;
 up to 32 characters) of the PROGRAM which is
 to receive control first in the execution
 of the HAL/S Load Module.

PROFILE : Requests that an execution profile of the
 executed program be printed.

APPENDIX C.

Prototype Catalogued Procedures

This Appendix contains seven examples of the form of catalogued procedures needed to compile, link edit and execute a HAL/S program. The seven procedures perform varying degrees of processing. The procedure names and their uses are:

- HALSC - Compile a HAL/S program;
- HALSCL - Compile and link a HAL/S program;
- HALSCLG - Compile and link a HAL/S program, and execute it in direct mode;
- HALSCLD - Compile and link a HAL/S program, and execute it under control of the execution monitoring system;
- HALSL - Link a previously compiled HAL program;
- HALSLG - Link and execute a previously compiled HAL/S program;
- HALSLD - Link a previously compiled HAL/S program, and execute it under control of the monitoring system.

The most complex procedure, HALSCLG, is described line by line.

The following comments apply to the prototype catalogued procedure HALSCLG listed in this appendix. The comments generally apply equally to any of the other procedures.

Line 10000 - This PROC statement names the procedure and defines the symbolic parameters. The OPTION and RUNPARM parameters are the means of supplying optional information to the compiler and run-time system respectively.

Lines 10100 - The name of the compilation step is HAL.
10200 The name of the actual program to be executed is MONITOR. MONITOR handles all compiler/OS interfaces and also performs the actual loading and overlaying of the two phases of the compiler. The compiler requires a 350K region, although 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 supplied. This is sufficient for most average size HAL/S programs (approx. 300 HAL/S statements).

The PARM field contains the compile-time options. The OPTION field receives any user-specified options.

Line 10300 - The STEPLIB DD card specifies the location of the load module library containing the module MONITOR needed to run the compiler. This card may define any direct access library which contains the proper module or may be deleted at installations where the module has been made part of the system library (SYS1.LINKLIB).

Line 10400 - The PROGRAM DD card defines the phases of the compiler that is to be used. This data set has a DCB of the form:

```
DCB=(RECFM=F,LRECL=7200,BLKSIZE=7200)
```

and may reside on direct access or magnetic tape. It is recommended, however, that direct access be used.

Line 10500 - The SYSPRINT DD card defines the primary listing data set. This is generally assigned to a system output class, but may be associated with any sequential data set with the proper characteristics. The record format must be FA or FBA with a logical record length of 133 and any appropriate block size. If not supplied in the JCL, the DCB will default to

```
DCB=(RECFM=FBA,LRECL=133,BLKSIZE=7182)
```

Line 10600 - The LISTING2 DD card defines the secondary listing data set. It may define a system output class or any sequential data set.

The DCB requirements are the same as for SYSPRINT.

Lines 10700 -
10900

The OUTPUT3 DD card defines the data set which is to receive object code which is produced by the compiler. In the prototype procedure, this data set is given a temporary Name (&&HALOBJ) and is passed (DISP=(MOD,PASS)) to subsequent steps.

Since this data set contains card images, it must have a logical record length of 80 and a record format of F or FB. The blocking factor may be any legal multiple of 80. If no DCB information is supplied in the JCL, the default will be DCB=(RECFM=FB,LRECL=80,BLKSIZE=7200). This DD card may specify a direct access, magnetic tape, or unit record device.

Line 11000 -

The OUTPUT4 DD card defines the location to which a duplicate of the OUTPUT3 file will be sent under control of the "DECK" compiler option. Its DCB characteristics are identical to OUTPUT3.

Lines 11100 -
11200

The OUTPUT5 DD card defines the data set which will receive the Simulation Data File (SDF). This file allows the execution-time DUMP and TRACE facilities to associate memory locations with HAL/S variable names. For the compile-load-go type of run this data set is generally a temporary one which exists only for the duration of the job. However, if the results of a given compilation are saved to be executed many times without recompilation, the SDF should be saved as well.

The OUTPUT5 DD card must define a partitioned data set. The SPACE parameters used in the prototype procedure are more than adequate to contain the mapping file of any one HAL/S program.

The OUTPUT5 DD card is accessed during the code generation phase of the compilation. One member of the partitioned data set is

127-C-3

created for a compilation unit. The name of the member is the first 8 characters of the name of the program or procedure being compiled, padded with blanks if necessary.

If a member with the desired name does not exist at compile-time, one is created. If a member with the desired name already exists, it is replaced by the new member.

The use of a permanent partitioned data set makes it possible to maintain a "library" of mapping files, with the member names uniquely specifying the HAL/S compilations to which the maps may be applied.

The format of the file must have a logical record length of 1680 and a record format of F. This DCB information must be supplied in the JCL.

Line 11300 -
11400

The OUTPUT6 DD card defines the partitioned data set onto which templates for compilation units may be placed. Refer to 6.2.7 for details of template generation. The DCB attributes for the data set should be compatible with those used for primary compiler input (SYSIN) and secondary compiler input (INCLUDE).

Line 11500 -

The ERROR DD defines the partitioned data set which contains the error message texts used by the syntax analysis phase of HAL/S. This data set is supplied with the compiler and, being a partitioned data set, must reside on a direct access volume.

Lines 11600 -
11600

The FILE1 through FILE6 DD cards specify work files. These files are used for interphase communication. The device may be either direct access or magnetic tape. Space equivalent to approximately 60 tracks on a 2314 should be available for each DD card. The DCB is internally specified and should not be altered by a JCL.

Line 12100 -

The SYSUDUMP DD card specifies the location to which an abend dump is to be sent in the event that the compiler terminates abnormally.

Lines 20000 -
21100

These lines define the data sets necessary to allow the external references in the object program to be resolved and prepare the program for loading into memory. The JCL is the standard required by the OS/360 Linkage Editor plus that which is necessary for execution of the HALLINK processing program. Details of the use of individual DD cards may be found in the IBM Linkage-Editor and Loader manual. The only feature of this JCL which is peculiar to a HAL/S compilation is the form of the SYSLIB DD card and the addition of the STACKOBJ and TEMPLOAD DD cards. These two additional DD cards should be used as specified in the procedures. They serve as intermediate work data sets for the HALLINK program.

The SYSPRINT dd card should always have a BLKSIZE parameter specified in the JCL.

The HAL/S runtime library package must be the primary source of modules to resolve external references. Any references which cannot be resolved from the HAL/S library will be resolved from any libraries catenated to the first DD card.

If the user has any HAL/S program references to user-written non-HAL/S subprograms or to previously compiled HAL/S programs, it is at this point that the programs must be supplied to the linking process. This may be done through the use of a catenated DD card specifying a user library, or by direct user input of control cards to the link editor.

Lines 3000 -
3700

These cards specify the execution of the load module created in the LKED step. The JCL for the step differs depending upon the choice of either direct execution or execution under the execution monitoring system. The two modes are determined by the particular procedure invoked: HALSCLG and HALSCLD invoke the execution monitoring system.

The EXEC card either invokes the load module produced in the LKED step directly,

or it invokes the execution monitor (EUNMON).

Certain JCL cards are common to both modes of execution:

The CHANNEL6 DD card is supplied in the procedure because device 6 is commonly used as an output unit. It is also supplied because all system-generated output (e.g. runtime error messages) is directed by default to CHANNEL6. The user may alter the destination of such messages via the MCHAN optional parameter.

The CHANNEL5 DD card has been supplied in the procedure because it is common practice to assume that device 5 will be used for input. The use of the DDNAME parameter on the CHANNEL5 DD card allows the user to specify either:

//GO.CHANNEL5 DD parameters

or

//GO.SYSIN DD parameters

to define the runtime input.

The SYSUMP DD card specifies the location to which a hexadecimal memory dump will be sent in the event of an abnormal termination.

The following JCL is special to the execution monitoring system:

The STEPLIB DD card indicates the location of the RUNMON program.

The PPROGRAM DD card indicates the location of the diagnostic processing program which is executed by RUNMON.

The SYSPRINT DD card identifies the output of the diagnostic processing program. Request cards and request card errors are listed here.

The HAI SDF DD card defines the dataset containing the SDF for the program(s) being executed. In the case of the prototype procedure, this is the temporary dataset created by the compilation step as OUTPUT5.

The HALLIB DD card defines the dataset

containing the HAL/S load module to be executed. The prototype procedure indicates the module produced in the LKED step just preceding the execution step.

131<

C-7

HALSC

```

10000 //HALSC      PROC OPTION=
10100 //HAL        EXEC PGM=MONITOR,REGION=350K,TIME=1,
10200 //           PARM='&OPTION'
10300 //STEPLIB    DD DISP=SHR,DSN=HALS360.MONITOR
10400 //PROGRAM    DD DISP=SHR,DSN=HALS360.COMPILER
10500 //SYSPRINT   DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=3458)
10600 //LISTING2   DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=3458)
10700 //OUTPUT3    DD UNIT=SYSDA,DISP=(MOD,PASS),SPACE=(CYL,(1,1)),
10800 //           DCB=(RECFM=FB,LRECL=80,BLKSIZE=400),
10900 //           DSN=&&HALOBJ
11000 //OUTPUT4    DD SYSOUT=B,DCB=(RECFM=FB,LRECL=80,BLKSIZE=400)
11100 //OUTPUT5    DD DISP=(MOD,PASS),DSN=&&HALSDF,SPACE=(TRK,(2,2,1)),
11200 //           DCB=(RECFM=F,LRECL=1680,BLKSIZE=1680),UNIT=SYSDA
11300 //OUTPUT6    DD DISP=(MOD,PASS),DSN=&&TEMPLIB,SPACE=(TRK,(2,2,1)),
11400 //           DCB=(RECFM=FB,LRECL=80,BLKSIZE=1680),UNIT=SYSDA
11500 //ERROR      DD DISP=SHR,DSN=HALS360.ERRORLIB
11600 //FILE1      DD UNIT=SYSDA,SPACE=(CYL,3)
11700 //FILE2      DD UNIT=SYSDA,SPACE=(CYL,3)
11800 //FILE3      DD UNIT=SYSDA,SPACE=(CYL,3)
11900 //FILE5      DD UNIT=SYSDA,SPACE=(CYL,3)
12000 //FILE6      DD UNIT=SYSDA,SPACE=(CYL,3)
12100 //SYSUDUMP   DD SYSOUT=A

```

HALSCL

```

10000 //HALSCL      PROC OPTION=
10100 //HAL        EXEC PGM=MONITOR,REGION=350K,TIME=1,
10200 //           PARM='&OPTION'
10300 //STEPLIB    DD DISP=SHR,DSN=HALS360.MONITOR
10400 //PROGRAM    DD DISP=SHR,DSN=HALS360.COMPILER
10500 //SYSPRINT   DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=3458)
10600 //LISTING2   DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=3458)
10700 //OUTPUT3    DD UNIT=SYSDA,DISP=(MOD,PASS),SPACE=(CYL,(1,1)),
10800 //           DCB=(RECFM=FB,LRECL=80,BLKSIZE=400),
10900 //           DSN=&&HALOBJ
11000 //OUTPUT4    DD SYSOUT=B,DCB=(RECFM=FB,LRECL=80,BLKSIZE=400)
11100 //OUTPUT5    DD DISP=(MOD,PASS),DSN=&&HALSDF,SPACE=(TRK,(2,2,1)),
11200 //           DCB=(RECFM=F,LRECL=1680,BLKSIZE=1680),UNIT=SYSDA
11300 //OUTPUT6    DD DISP=(MOD,PASS),DSN=&&TEMPLIB,SPACE=(TRK,(2,2,1)),
11400 //           DCB=(RECFM=FB,LRECL=80,BLKSIZE=1680),UNIT=SYSDA
11500 //ERROR      DD DISP=SHR,DSN=HALS360.ERRORLIB
11600 //FILE1      DD UNIT=SYSDA,SPACE=(CYL,3)
11700 //FILE2      DD UNIT=SYSDA,SPACE=(CYL,3)
11800 //FILE3      DD UNIT=SYSDA,SPACE=(CYL,3)
11900 //FILE5      DD UNIT=SYSDA,SPACE=(CYL,3)
12000 //FILE6      DD UNIT=SYSDA,SPACE=(CYL,3)
12100 //SYSUDUMP   DD SYSOUT=A
20000 //LKED      EXEC PGM=HALLINK,REGION=100K,PARM=(LIST,MAP),
20100 //           COND=(0,LT,HAL)
20200 //STEPLIB    DD DISP=SHR,DSN=HALS360.MONITOR
20300 //SYSPRINT   DD SYSOUT=A,DCB=BLKSIZE=1210
20400 //SYSLIB     DD DSN=HALS360.RUNLIB,DISP=SHR
20500 //SYSLIN     DD DISP=OLD,DSN=&&HALOBJ
20600 //           DD DDNAME=SYSIN
20700 //SYSLMOD    DD DSN=&&HALMOD(GO),DISP=(,PASS),UNIT=SYSDA,
20800 //           SPACE=(CYL,(1,1,1))
20900 //SYSUT1     DD SPACE=(CYL,(1,1)),UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD))
21000 //STACKOBJ   DD SPACE=(TRK,(5,10)),UNIT=SYSDA
21100 //TEMPLOAD   DD SPACE=(CYL,(1,1,1)),UNIT=SYSDA

```

HALSCLD

```

10000 //HALSCLD  PROC OPTION=,RUNPARM=
10100 //HAL      EXEC PGM=MONITOR,REGION=350K,TIME=1,
10200 //          PARM='&OPTION'
10300 //STEPLIB  DD DISP=SHR,DSN=HALS360.MONITOR
10400 //PROGRAM  DD DISP=SHR,DSN=HALS360.COMPILER
10500 //SYSPRINT DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=3458)
10600 //LISTING2 DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=3458)
10700 //OUTPUT3  DD UNIT=SYSDA,DISP=(MOD,PASS),SPACE=(CYL,(1,1)),
10800 //          DCB=(RECFM=FB,LRECL=80,BLKSIZE=400),
10900 //          DSN=&&HALOBJ
11000 //OUTPUT4  DD SYSOUT=B,DCB=(RECFM=FB,LRECL=80,BLKSIZE=400)
11100 //OUTPUT5  DD DISP=(MOD,PASS),DSN=&&HALSDF,SPACE=(TRK,(2,2,1)),
11200 //          DCB=(RECFM=F,LRECL=1680,BLKSIZE=1680),UNIT=SYSDA
11300 //OUTPUT6  DD DISP=(MOD,PASS),DSN=&&TEMPLIB,SPACE=(TRK,(2,2,1)),
11400 //          DCB=(RECFM=FB,LRECL=80,BLKSIZE=1680),UNIT=SYSDA
11500 //ERROR    DD DISP=SHR,DSN=HALS360.ERRORLIB
11600 //FILE1     DD UNIT=SYSDA,SPACE=(CYL,3)
11700 //FILE2     DD UNIT=SYSDA,SPACE=(CYL,3)
11800 //FILE3     DD UNIT=SYSDA,SPACE=(CYL,3)
11900 //FILE5     DD UNIT=SYSDA,SPACE=(CYL,3)
12000 //FILE6     DD UNIT=SYSDA,SPACE=(CYL,3)
12100 //SYSUDUMP DD SYSOUT=A
20000 //LKED     EXEC PGM=HALLINK,REGION=100K,PARM=(LIST,MAP),
20100 //          COND=(0,LT,HAL)
20200 //STEPLIB  DD DISP=SHR,DSN=HALS360.MONITOR
20300 //SYSPRINT DD SYSOUT=A,DCB=BLKSIZE=1210
20400 //SYSLIB   DD DSN=HALS360.RUNLIB,DISP=SHR
20500 //SYSLIN   DD DISP=OLD,DSN=&&HALOBJ
20600 //          DD DDNAME=SYSIN
20700 //SYSLMOD  DD DSN=&&HALMOD,DISP=(,PASS),UNIT=SYSDA,
20800 //          SPACE=(CYL,(1,1,1))
20900 //SYSUT1   DD SPACE=(CYL,(1,1)),UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD))
21000 //STACKOBJ DD SPACE=(TRK,(5,10)),UNIT=SYSDA
21100 //TEMPLOAD DD SPACE=(CYL,(1,1,1)),UNIT=SYSDA
30000 //GO       EXEC PGM=RUNMON,REGION=100K,
30100 //          COND=((0,LT,HAL),(4,LT,LKED)),
30110 //          PARM='&RUNPARM'
30200 //STEPLIB  DD DISP=SHR,DSN=HALS360.MONITOR
30300 //PROGRAM  DD DISP=SHR,DSN=HALS360.DIAGPROC
30400 //SYSPRINT DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=3458)
30500 //HALLIB   DD DISP=OLD,DSN=&&HALMOD
30600 //CHANNEL6 DD SYSOUT=A
30700 //HALSDF   DD DISP=OLD,DSN=&&HALSDF
30800 //CHANNEL5 DD DDNAME=SYSIN
30900 //SYSUDUMP DD SYSOUT=A

```

134<

HALSCLG

```

10000 //HALSCLG PROC OPTION=,RUNPARM=
10100 //HAL EXEC PGM=MONITOR,REGION=350K,TIME=1,
10200 // PARM='NOTABLES,&OPTION'
10300 //STEPLIB DD DISP=SHR,DSN=HALS360.MONITOR
10400 //PROGRAM DD DISP=SHR,DSN=HALS360.COMPILER
10500 //SYSPRINT DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=3458)
10600 //LISTING2 DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=3458)
10700 //OUTPUT3 DD UNIT=SYSDA,DISP=(MOD,PASS),SPACE=(CYL,(1,1)),
10800 // DCB=(RECFM=FB,LRECL=80,BLKSIZE=400),
10900 // DSN=&&HALOBJ
11000 //OUTPUT4 DD SYSOUT=B,DCB=(RECFM=FB,LRECL=80,BLKSIZE=400)
11100 //OUTPUT5 DD DISP=(MOD,PASS),DSN=&&HALSDF,SPACE=(TRK,(2,2,1)),
11200 // DCB=(RECFM=F,LRECL=1680,BLKSIZE=1680),UNIT=SYSDA
11300 //OUTPUT6 DD DISP=(MOD,PASS),DSN=&&TEMPLIB,SPACE=(TRK,(2,2,1)),
11400 // DCB=(RECFM=FB,LRECL=80,BLKSIZE=1680),UNIT=SYSDA
11500 //ERROR DD DISP=SHR,DSN=HALS360.ERRORLIB
11600 //FILE1 DD UNIT=SYSDA,SPACE=(CYL,3)
11700 //FILE2 DD UNIT=SYSDA,SPACE=(CYL,3)
11800 //FILE3 DD UNIT=SYSDA,SPACE=(CYL,3)
11900 //FILE5 DD UNIT=SYSDA,SPACE=(CYL,3)
12000 //FILE6 DD UNIT=SYSDA,SPACE=(CYL,3)
12100 //SYSUDUMP DD SYSOUT=A
20000 //LKED EXEC PGM=HALLINK,REGION=100K,PARM=(LIST,MAP),
20100 // COND=(0,LT,HAL)
20200 //STEPLIB DD DISP=SHR,DSN=HALS360.MONITOR
20300 //SYSPRINT DD SYSOUT=A,DCB=BLKSIZE=1210
20400 //SYSLIB DD DSN=HALS360.RUNLIB,DISP=SHR
20500 //SYSLIN DD DISP=OLD,DSN=&&HALOBJ
20600 // DD DDNAME=SYSIN
20700 //SYSLMOD DD DSN=&&HALMOD(GO),DISP=(,PASS),UNIT=SYSDA,
20800 // SPACE=(CYL,(1,1,1))
20900 //SYSUT1 DD SPACE=(CYL,(1,1)),UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD))
21000 //STACKOBJ DD SPACE=(TRK,(5,10)),UNIT=SYSDA
21100 //TEMPLOAD DD SPACE=(CYL,(1,1,1)),UNIT=SYSDA
30000 //GO EXEC PGM=*.LKED.SYSLMOD,REGION=100K,
30100 // COND=((0,LT,HAL),(4,LT,LKED)),
30200 // PARM='&RUNPARM'
30300 //CHANNEL6 DD SYSOUT=A
30600 //CHANNEL5 DD DDNAME=SYSIN
30700 //SYSUDUMP DD SYSOUT=A

```


HALSL

```
10000 //HALSL      PROC
20000 //LKED       EXEC PGM=HALLINK,REGION=100K,PARM=(LIST,MAP)
20100 //STEPLIB    DD DISP=SHR,DSN=HALS360.MONITOR
20200 //SYSPRINT   DD SYSOUT=A,DCB=BLKSIZE=1210
20300 //SYSLIB     DD DSN=HALS360.RUNLIB,DISP=SHR
20400 //SYSLIN     DD DDNAME=SYSIN
20500 //SYSLMOD    DD DSN=H&HALMOD(GO),DISP=(,PASS),UNIT=SYSDA,
20600 //              SPACE=(CYL,(1,1,1))
20700 //SYSUT1     DD SPACE=(CYL,(1,1)),UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD))
20800 //STACKOBJ   DD SPACE=(TRK,(5,10)),UNIT=SYSDA
20900 //TEMPLOAD   DD SPACE=(CYL,(1,1,1)),UNIT=SYSDA
```

136<

HALSLD

```
10000 //HALSLD      PROC RUNPARG=
20000 //LKED        EXEC PGM=HALLINK,REGION=100K,PARM=(LIST,MAP)
20100 //STEPLIB     DD DISP=SHR,DSN=HALS360.MONITOR
20200 //SYSPRINT    DD SYSOUT=A,DCB=BLKSIZE=1210
20300 //SYSLIB      DD DSN=HALS360.RUNLIB,DISP=SHR
20400 //SYSLIN      DD DDNAME=SYSIN
20500 //SYSLMOD     DD DSN=&&HALMOD,DISP=(,PASS),UNIT=SYSDA,
20600 //              SPACE=(CYL,(1,1,1))
20700 //SYSUT1      DD SPACE=(CYL,(1,1)),UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD))
20800 //STACKOBJ    DD SPACE=(TRK,(5,10)),UNIT=SYSDA
20900 //TEMPLOAD    DD SPACE=(CYL,(1,1,1)),UNIT=SYSDA
30000 //GO          EXEC PGM=RUNMON,REGION=100K,
30100 //              COND=(4,LT,LKED),
30150 //              PARM='&RUNPARG'
30200 //STEPLIB     DD DISP=SHR,DSN=HALS360.MONITOR
30300 //PROGRAM      DD DISP=SHR,DSN=HALS360.DIAGPROC
30400 //SYSPRINT    DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,BLKSIZE=3458)
30500 //HALLIB      DD DISP=OLD,DSN=&&HALMOD
30600 //CHANNEL6     DD SYSOUT=A
30800 //CHANNEL5     DD DDNAME=SYSIN
30900 //SYSUDUMP    DD SYSOUT=A
```

137<

C-13

HALSLG

```
10000 //HALSLG   PROC RUNPARM=
20000 //LKED     EXEC PGM=HALLINK,REGION=100K,PARM=(LIST,MAP)
20100 //STEPLIB  DD DISP=SHR,DSN=HALS360.MONITOR
20200 //SYSPRINT DD SYSOUT=A,DCB=BLKSIZE=1210
20300 //SYSLIB   DD DSN=HALS360.RUNLIB,DISP=SHR
20400 //SYSLIN   DD DDNAME=SYSIN
20500 //SYSLMOD  DD DSN=H&HALMOD(GO),DISP=(,PASS),UNIT=SYSDA,
20600 //          SPACE=(CYL,(1,1,1))
20700 //SYSUT1   DD SPACE=(CYL,(1,1)),UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD))
20800 //STACKOBJ DD SPACE=(TRK,(5,10)),UNIT=SYSDA
20900 //TEMPLOAD DD SPACE=(CYL,(1,1,1)),UNIT=SYSDA
30000 //GO       EXEC PGM=*.LKED.SYSLMOD,REGION=100K,
30100 //          COND=(4,LT,LKED)
30200 //          PARM='&RUNPARM'
30300 //CHANNEL6 DD SYSOUT=A
30600 //CHANNEL5 DD DDNAME=SYSIN
30700 //SYSUDUMP DD SYSOUT=A
```

APPENDIX D.

Compile Time Error Messages

A complete list of compile time error messages is presented here. The first table gives the mnemonic naming scheme used to identify the class-subclass structure of the Phase I error messages. The complete list of Phase I errors are presented next. Phase II errors are listed last. These errors do not have a mnemonic naming scheme and are simply listed with their severities. The occurrences of double question marks (??) in the text of the messages listed here indicate positions at which text specific to each actual error will be inserted (e.g. a variable name may be inserted to make a clear identification of the error source).

Error Classifications

Note: "b" denotes a blank.

CLASS A: ASSIGNMENT STATEMENTS

A	ARRAY ASSIGNMENT
V	COMPLEX VARIABLE ASSIGNMENT
b	MISCELLANEOUS ASSIGNMENT

CLASS B: COMPILER TERMINATION

A	HALMAT BLOCK SIZE
N	NAME SCOPE NESTING
S	STACK SIZE LIMITATIONS
T	TABLE SIZE LIMITATIONS
X	COMPILER ERRORS
b	MISCELLANEOUS

CLASS C: COMPARISONS

b	GENERAL COMPARISONS
---	---------------------

CLASS D: DECLARATION ERRORS

A	ATTRIBUTE LIST
C	STORAGE CLASS ATTRIBUTE
D	DIMENSION
F	FUNCTION DECLARATION
I	INITIALIZATION
L	LOCKING ATTRIBUTE
Q	STRUCTURE TEMPLATE TREE ORGANIZATION
S	FACTORED/UNFACTORED SPECIFICATION
T	TYPE SPECIFICATION
U	UNDECLARED DATA
b	MISCELLANEOUS

T TYPE SPECIFICATION
U UNDECLARED DATA
b MISCELLANEOUS

CLASS E: EXPRESSIONS

A ARRAYNESS
B BIT STRING EXPRESSIONS
C CROSS PRODUCT
D DOT PRODUCT
L LIST EXPRESSIONS
M MATRIX EXPRESSIONS
O OUTER PRODUCT
V VECTOR EXPRESSIONS
b MISCELLANEOUS EXPRESSIONS

CLASS F: FORMAL PARAMETERS & ARGUMENTS

D DIMENSION AGREEMENT
N NUMBER OF ARGUMENTS
S SUBBIT ARGUMENTS
T TYPE AGREEMENT

CLASS G: STATEMENT GROUPINGS (DO GROUPS)

B BIT TYPE CONTROL EXPRESSION
C CONTROL EXPRESSION
E EXIT/REPEAT STATEMENTS
L END LABEL
V CONTROL VARIABLE

CLASS I: IDENTIFIERS

L LENGTH
R REPLACED IDENTIFIERS
S QUALIFIED STRUCTURE NAMES

CLASS L: LITERALS

B BIT STRING
C CONVERSION TO INTERNAL FORMS
F FORMAT OF ARITHMETIC LITERALS
S CHARACTER STRING

CLASS M: MULTILINE FORMAT

C OVERPUNCH CONTEXT
E E-LINE

O OVERPUNCH USE
S S-LINE
b COMMENTS

CLASS P: PROGRAM CONTROL & INTERNAL CONSISTANCE

A ACCESS CONTROL
C COMPOOL BLOCKS
D DATA DEFINITION
E EXTERNAL TEMPLATES
F FUNCTION RETURN EXPRESSIONS
L LABELS
M MULTIPLE DEFINITIONS
P BLOCK DEFINITION
S PROCEDURE/FUNCTION TEMPLATES
T TASK DEFINITIONS
U CALLS FROM UPDATE BLOCKS
b MISCELLANEOUS

CLASS Q: SHAPING FUNCTIONS

A ARRAYNESS
D DIMENSION INFORMATION
S SUBSCRIPTS
X ARGUMENT TYPE

CLASS R: REAL TIME STATEMENTS

E ON/SEND ERROR STATEMENTS
T TIMING EXPRESSIONS
U UPDATE BLOCKS

CLASS S: SUBSCRIPT USAGE

C SUBSCRIPT COUNT
P PUNCTUATION
Q PRECISION QUALIFIER
R RANGE OF SUBSCRIPT VALUES
S USAGE OF ASTERISKS
T SUBSCRIPT TYPE
V VALIDITY OF USAGE

CLASS T: I/O STATEMENTS

C CONTROL
D DEVICE NUMBER
b MISCELLANEOUS

CLASS U: UPDATE BLOCKS

I	IDENTIFIER USAGE
P	PROGRAM BLOCKS
T	I/O

CLASS V: COMPILE-TIME EVALUATIONS

A	ARITHMETIC OPERATIONS
C	CATENATION OPERATIONS
E	UNCOMPUTABLE EXPRESSIONS
F	FUNCTION EVALUATION

CLASS X: IMPLEMENTATION DEPENDENT FEATURES

A	PROGRAM ID DIRECTIVE
D	DEVICE DIRECTIVE
I	INCLUDE DIRECTIVE
U	UNKNOWN OR INVALID DIRECTIVE

ERROR MESSAGES FOR MAJOR CLASSIFICATION A
CLASSIFICATION "A" ERRORS ARE RELATED TO ASSIGNMENT STATEMENTS

- AA1 -SEVERITY 1
ARRAYNESS OF LEFT HAND SIDE OF ASSIGNMENT DOES NOT MATCH THAT OF RIGHT HAND SIDE
- AA2 -SEVERITY 1
ARRAYNESS OF ?? IS INCONSISTENT WITH THAT OF OTHER LEFT HAND SIDE VARIABLES
- AA3 -SEVERITY 1
ARRAYNESS OF ?? DISAGREES WITH ARRAYNESS OF ITS SUBSCRIPTING
- AV0 -SEVERITY 1
ARGUMENTS ON EITHER SIDE OF NAME ASSIGNMENT ARE INCOMPATIBLE.
- AV1 -SEVERITY 1
TYPE OF ?? IS ILLEGAL FOR ASSIGNMENT FROM GIVEN LEFT-HAND SIDE.
- AV2 -SEVERITY 1
MATRIX DIMENSIONS DISAGREE ACROSS ASSIGNMENT
- AV3 -SEVERITY 1
VECTOR LENGTHS DISAGREE ACROSS ASSIGNMENT
- AV4 -SEVERITY 1
TREE ORGANIZATIONS DO NOT MATCH ACROSS ASSIGNMENT
- AV5 -SEVERITY 1
ONLY ONE OPERAND IN ASSIGNMENT IS A NAME PSEUDO-FUNCTION
OR NULL.
- A1 -SEVERITY 1
ILLEGAL ASSIGNMENT TO CONSTANT OR PARAMETER ??
- A2 -SEVERITY 1
?? POSSESSES SUBSCRIPTS ILLEGAL FOR THE ARGUMENT OF A NAME
PSEUDO-FUNCTION IN ASSIGNMENT CONTEXT.
- A3 -SEVERITY 1
?? DOES NOT POSSESS THE NAME ATTRIBUTE - IT IS THEREFORE
ILLEGAL AS ARGUMENT OF A NAME PSEUDO-FUNCTION IN
ASSIGNMENT CONTEXT.

ERROR MESSAGES FOR MAJOR CLASSIFICATION B
CLASSIFICATION "B" ERRORS RESULT FROM ABORTIVE COMPILER FAILURES

BB1 -SEVERITY 2
INTERMEDIATE CODE STORAGE OVERFLOW: ERROR SCAN CONTINUING

BN1 -SEVERITY 3
MAX NAME SCOPE NESTING DEPTH EXCEEDED

BS1 -SEVERITY 3
MAXIMUM DEPTH OF DO...END GROUP NESTING EXCEEDED

BS2 -SEVERITY 2
INDIRECT PARSE STACK SIZE EXCEEDED

BS3 -SEVERITY 3
PARSE STACK OVERFLOW

BS4 -SEVERITY 2
CURRENT ARRAYNESS STACK SIZE EXCEEDED

BS5 -SEVERITY 1
MAXIMUM FUNCTION NESTING DEPTH EXCEEDED

BT1 -SEVERITY 3
SYMBCL TABLE OVERFLOW

BT3 -SEVERITY 3
LITERAL TABLE DATA OVERFLOW

BT4 -SEVERITY 3
LITERAL TABLE STRING OVERFLOW

BT5 -SEVERITY 3
MACRO TABLE OVERFLOW

BT7 -SEVERITY 2
INITIAL LIST STORAGE CAPACITY EXCEEDED

BX1 -SEVERITY 2
SYT_CLASS = 0 FOR ??

BX2 -SEVERITY 2
FUNC_TOKEN = 0

BX4 -SEVERITY 3
TOO MANY BUILT-IN FUNCTIONS

BX5 -SEVERITY 2
EXT_ARRAY OVERFLOW

B1 -SEVERITY 3
INSUFFICIENT CORE AVAILABLE

B2 -SEVERITY 3
INLINE FUNCTION MAY NOT BE IN A SUBSCRIPT OR EXPONENT.
THIS ERROR IS IRRECOVERABLE.

145<

D-7

ERROR MESSAGES FOR MAJOR CLASSIFICATION C
CLASSIFICATION "C" ERRORS DEAL WITH COMPARISONS

- C0 -SEVERITY 1
ARGUMENTS IN NAME COMPARISON ARE INCOMPATIBLE.
- C1 -SEVERITY 1
?? COMPARISONS MAY ONLY BE = OR !=
- C2 -SEVERITY 1
ARRAYED COMPARISONS ARE RESTRICTED TO = OR !=
- C3 -SEVERITY 1
TREE ORGANIZATIONS OF STRUCTURES COMPARED DO NOT MATCH
- C4 -SEVERITY 1
ONLY ONE OPERAND OF COMPARISON IS A NAME PSEUDO-FUNCTION
OR NULL.

146<

D-8

ERROR MESSAGES FOR MAJOR CLASSIFICATION D
CLASSIFICATION "D" ERRORS ARE RELATED TO DATA DECLARATIONS

- DA0 -SEVERITY 1
 CONFLICTING ATTRIBUTE SPECIFIED WITH THE LATCHED ATTRIBUTE
- DA1 -SEVERITY 1
 ILLEGAL ATTRIBUTE SPECIFIED FOR BIT OR BOOLEAN DATA TYPE
- DA10 -SEVERITY 1
 ILLEGAL ATTRIBUTE SPECIFIED FOR STRUCTURE DATA TYPE
- DA11 -SEVERITY 1
 CONFLICT BETWEEN ATTRIBUTES AND FUNCTION OR LABEL TYPE SPECIFICATION
 - ATTRIBUTES IGNORED
- DA2 -SEVERITY 1
 ILLEGAL ATTRIBUTE SPECIFIED FOR CHARACTER DATA TYPE
- DA20 -SEVERITY 1
 ILLEGAL ATTRIBUTE SPECIFIED FOR MINOR STRUCTURE ??
- DA21 -SEVERITY 1
 AN ARRAY SPECIFICATION IS NOT ALLOWED FOR THE MINOR STRUCTURE ??
- DA22 -SEVERITY 1
 NO ATTRIBUTES MAY BE SPECIFIED ON A NESTED STRUCTURE TEMPLATE REFERENCE
- DA23 -SEVERITY 1
 ILLEGAL ATTRIBUTE FOR THE STRUCTURE TERMINAL ??
- DA24 -SEVERITY 1
 FACTORED AND NON-FACTORED ATTRIBUTE SPECIFICATIONS FOR ?? DISAGREE;
 THE NON-FACTORED ATTRIBUTES WILL BE GIVEN PRECEDENCE.
- DA25 -SEVERITY 1
 CONTRADICTIONARY PAIR OF ATTRIBUTES SUPPLIED - FIRST
 APPEARING ATTRIBUTE WILL BE USED
- DA3 -SEVERITY 1
 ILLEGAL ATTRIBUTE SPECIFIED FOR MATRIX DATA TYPE
- DA4 -SEVERITY 1
 ILLEGAL ATTRIBUTE SPECIFIED FOR VECTOR DATA TYPE
- DA5 -SEVERITY 1
 ILLEGAL ATTRIBUTE SPECIFIED FOR SCALAR DATA TYPE
- DA6 -SEVERITY 1
 ILLEGAL ATTRIBUTE SPECIFIED FOR INTEGER DATA TYPE
- DA9 -SEVERITY 1
 ILLEGAL ATTRIBUTE SPECIFIED FOR EVENT DATA TYPE
- DC1 -SEVERITY 1
 DECLARATION CONTAINS BOTH LABEL TYPE AND DATA TYPE SPECIFICATION
 - LABEL TYPE IGNORED.
- DC2 -SEVERITY 1
 THE ATTRIBUTES STATIC AND AUTOMATIC MAY NOT BE SPECIFIED IN COMPOOL
 DECLARATIONS.

147<

DC3 -SEVERITY 1
THE PARAMETER ?? MAY NOT HAVE STATIC OR AUTOMATIC ATTRIBUTES SPECIFIED

DC4 -SEVERITY 1
FACTORED AND NON-FACTORED TYPE SPECIFICATION FOR ?? DISAGREE
THE NON-FACTORED TYPE SPECIFICATION WILL BE USED

DC5 -SEVERITY 1
ACCESS ATTRIBUTES MAY ONLY BE QUALIFIED IN COMPOOL DECLARATIONS.

DD1 -SEVERITY 1
ILLEGAL ARRAY DIMENSION SPECIFICATION

DD10 -SEVERITY 1
* ARRAY SIZE IS ILLEGAL FOR ?? - ARRAY SIZE OF 2 ASSUMED.

DD11 -SEVERITY 1
ILLEGAL FORM OF STRUCTURE DIMENSION SPECIFICATION

DD12 -SEVERITY 1
ILLEGAL CONTEXT FOR SPECIFICATION OF STRUCTURE COPIES

DD3 -SEVERITY 2
TOO MANY DIMENSIONS IN ARRAY

DD4 -SEVERITY 1
INVALID MATRIX DIMENSION SPECIFICATION; A DIMENSION OF 3 IS ASSUMED

DD5 -SEVERITY 1
INVALID VECTOR LENGTH SPECIFICATION; A 3-VECTOR IS ASSUMED

DD6 -SEVERITY 1
ONLY SINGLE DIMENSION ARRAYS MAY USE THE * TO DENOTE UNKNOWN LENGTH

DD7 -SEVERITY 1
A * MAY NOT BE USED TO SPECIFY VECTOR LENGTH; A 3-VECTOR IS ASSUMED

DD8 -SEVERITY 1
* STRUCTURE COPY NOTATION IS ILLEGAL FOR ?? -
STRUCTURE COPY SIZE OF 2 ASSUMED.

DD9 -SEVERITY 1
A * MAY NOT BE USED TO SPECIFY A MATRIX DIMENSION; A DIMENSION OF 3 IS ASSUMED

DF1 -SEVERITY 1
THE FUNCTION ?? MAY NOT BE DECLARED IN A COMPOOL

DF2 -SEVERITY 1
ILLEGAL ATTRIBUTE FOR THE FUNCTION ??

DF3 -SEVERITY 1
THE FUNCTION ?? MAY NOT HAVE AN INITIAL/CONSTANT SPECIFICATION

DI1 -SEVERITY 1
REPEAT FACTOR IN INITIALIZATION HAS NO LEGAL VALUE COMPUTABLE AT
COMPILE TIME

DI10 -SEVERITY 1
TOO MANY ELEMENTS SUPPLIED IN INITIAL LIST FOR ??

148<

- DI11 -SEVERITY 1
THE VARIABLE ?? USED IN A COMPILE-TIME EXPRESSION HAS NOT BEEN PREVIOUSLY
DEFINED
- DI12 -SEVERITY 1
FORMAL PARAMETER ?? POSSESSES ILLEGAL ATTRIBUTES CONCERNING
INITIALIZATION - ATTRIBUTES IGNORED.
- DI13 -SEVERITY 1
LABEL OR FUNCTION ?? POSSESSES ILLEGAL ATTRIBUTES CONCERNING
INITIALIZATION - ATTRIBUTES IGNORED.
- DI14 -SEVERITY 1
IN AN INITIAL LIST, THE ARGUMENT OF A NAME PSEUDO-FUNCTION
MAY NOT POSSESS A SUBSCRIPT.
- DI15 -SEVERITY 1
IN AN INITIAL LIST, THE ARGUMENT OF A NAME PSEUDO-FUNCTION
MAY NOT POSSESS ARRAYSNESS.
- DI16 -SEVERITY 1
IN AN INITIAL LIST THE ARGUMENT OF A NAME
PSEUDO-FUNCTION MAY NOT POSSESS THE NAME ATTRIBUTE.
- DI2 -SEVERITY 1
IMPLIED NUMBER OF ELEMENTS IN INITIAL LIST EXCEEDS COMPILER LIMIT
- DI3 -SEVERITY 1
EXPRESSION IN INITIAL LIST IS NOT COMPUTABLE AT COMPILE TIME
- DI4 -SEVERITY 1
INITIALIZATION OF ?? HAS ILLEGAL TERMINATING * :
NUMBER OF INITIAL VALUES MATCHES TOTAL NUMBER OF ELEMENTS
- DI5 -SEVERITY 1
TOO FEW ELEMENTS SUPPLIED IN INITIAL LIST FOR ??
- DI6 -SEVERITY 1
ILLEGALLY-TYPED INITIAL VALUE--INITIALIZATION OF ?? EXPECTS
A VALUE OF CHARACTER TYPE
- DI7 -SEVERITY 1
ILLEGALLY-TYPED INITIAL VALUE--INITIALIZATION OF ?? EXPECTS
A VALUE OF BIT TYPE
- DI8 -SEVERITY 1
ILLEGALLY-TYPED INITIAL VALUE--INITIALIZATION OF ?? EXPECTS A
VALUE OF INTEGER OR SCALAR TYPE
- DI9 -SEVERITY 1
THE DECLARATION OF ?? HAS BOTH FACTORED AND UNFACTORED INITIAL/CONSTANT
ATTRIBUTES; THE UNFACTORED ATTRIBUTES WILL BE USED
- DI1 -SEVERITY 1
?? DOES NOT APPEAR IN A COMPOOL OR PROGRAM DECLARATION,
AND IS NOT AN ASSIGN PARAMETER: THE LOCKING ATTRIBUTE
SPECIFIED IS THEREFORE ILLEGAL.
- DL2 -SEVERITY 1
THE LOCKED ATTRIBUTE MAY NOT BE USED IN CONJUNCTION WITH THE CONSTANT ATTRIBUTE

149<

DL3 -SEVERITY 1
 ILLEGAL LOCK GROUP NUMBER SPECIFIED

DN1 -SEVERITY 1
 PROGRAM, TASK, OR PROCEDURE DECLARATION FOR ?? DOES NOT CONTAIN NAME ATTRIBUTE

DN2 -SEVERITY 1
 THE NAME ATTRIBUTE MAY NOT BE USED IN THE DECLARATION
 OF TEMPORARIES - ATTRIBUTE IGNORED.

DO1 -SEVERITY 1
 FIRST NODE DECLARED IN TEMPLATE MUST BE AT LEVEL 1

DO10 -SEVERITY 1
 STRUCTURE ?? MAY NOT BE A TEMPORARY SINCE ITS TEMPLATE CONTAINS
 AT LEAST ONE TERMINAL NODE WITH THE NAME ATTRIBUTE.

DO2 -SEVERITY 1
 ILLEGAL SEQUENCE OF LEVEL NUMBERS IN TEMPLATE

DO3 -SEVERITY 1
 NAME OF STRUCTURE TEMPLATE CAUSES UNQUALIFICATION IN AN ILLEGAL CONTEXT

DO4 -SEVERITY 1
 STRUCTURE ?? CANNOT BE UNQUALIFIED - STRUCTURE TEMPLATE IS ALREADY USED
 BY AN UNQUALIFIED STRUCTURE

DO5 -SEVERITY 1
 STRUCTURE ?? CANNOT BE UNQUALIFIED - STRUCTURE TEMPLATE IS NOT
 IN SAME NAME SCOPE

DO6 -SEVERITY 1
 STRUCTURE ?? CANNOT BE UNQUALIFIED - STRUCTURE TEMPLATE CONTAINS
 A REFERENCE TO ANOTHER STRUCTURE TEMPLATE.

DO7 -SEVERITY 1
 THE DECLARED NAME ?? DUPLICATES THE NAME OF A NODE OF THE
 TEMPLATE OF AN UNQUALIFIED STRUCTURE PREVIOUSLY DECLARED
 IN THE SAME NAME SCOPE: THIS CAUSES THE UNQUALIFICATION TO BECOME
 ILLEGAL.

DO8 -SEVERITY 1
 STRUCTURE ?? CANNOT BE UNQUALIFIED - STRUCTURE TEMPLATE CONTAINS
 AT LEAST ONE NAME NOT UNIQUE TO THE NAME SCOPE

DO9 -SEVERITY 1
 THE DECLARED NODE ?? DUPLICATES A PREVIOUSLY DECLARED NODE NAME -
 CAUSING QUALIFIED REFERENCES TO THE TWO NODES TO BE INDISTINGUISHABLE

DS1 -SEVERITY 1
 INVALID BIT-LENGTH SPECIFICATION

DS10 -SEVERITY 1
 FACTORED AND NON-FACTORED STRUCTURE TEMPLATE REFERENCES DISAGREE;
 NON-FACTORED REFERENCE WILL BE USED

DS11 -SEVERITY 1
 INPUT/ASSIGN PARAMETERS OF CHARACTER TYPE CAN ONLY BE GIVEN A * LENGTH
 SPECIFICATION

DS2 -SEVERITY 1
INVALID CHAR-LENGTH SPECIFICATION

DS3 -SEVERITY 1
A * IS AN ILLEGAL CHARACTER LENGTH SPECIFICATION; A LENGTH OF 8 IS ASSUMED

DS4 -SEVERITY 1
A * IS AN ILLEGAL BIT LENGTH SPECIFICATION; A LENGTH OF 1 IS ASSUMED

DS5 -SEVERITY 1
FACTORED AND NON-FACTORED BIT SIZE SPECIFICATION FOR ?? DISAGREE
THE NCN-FACTORED SPECIFICATION WILL BE USED

DS6 -SEVERITY 1
FACTORED AND NON-FACTORED CHARACTER LENGTH SPECIFICATION FOR ?? DISAGREE
THE NCN-FACTORED SPECIFICATION WILL BE USED

DS7 -SEVERITY 1
FACTORED AND NON-FACTORED MATRIX DIMENSION SPECIFICATION FOR ?? DISAGREE
THE NCN-FACTORED SPECIFICATION WILL BE USED

DS8 -SEVERITY 1
FACTORED AND NON-FACTORED VECTOR DIMENSION SPECIFICATION FOR ?? DISAGREE
THE NON-FACTORED SPECIFICATION WILL BE USED

DS9 -SEVERITY 1
THE FACTORED AND NON-FACTORED STRUCTURE DIMENSION SPECIFICATION FOR ?? DISAGREE
THE NON-FACTORED SPECIFICATION WILL BE USED

DT1 -SEVERITY 1
CONFLICTING TYPE SPECIFICATIONS FOR ??

DT2 -SEVERITY 1
A REFERENCE TO A PREVIOUS TEMPLATE MAY NOT APPEAR IN THE CONTEXT OF A
MINOR STRUCTURE

DT3 -SEVERITY 1
LABEL TYPE CONFLICT FOR ??

DT4 -SEVERITY 1
ILLEGAL CHARACTER; HEX REPRESENTATION IS ??

DT5 -SEVERITY 1
A TYPE SPECIFICATION MAY NOT BE USED ON THE MINOR STRUCTURE ??

DT6 -SEVERITY 1
A STRUCTURE TEMPLATE MAY NOT CONTAIN A REFERENCE TO ITSELF

DT7 -SEVERITY 1
ILLEGAL TYPE FOR THE STRUCTURE TERMINAL ??

DT8 -SEVERITY 1
?? IS EVENT TYPE AND MAY NOT THEREFORE BE AN INPUT PARAMETER

DU1 -SEVERITY 1
UNDECLARED IDENTIFIER ??

DU2 -SEVERITY 1
UNDECLARED PARAMETER ??

DU3 -SEVERITY 0
?? IS NOT A PARAMETER AND CANNOT BE DECLARED IN A TEMPLATE

DU4 -SEVERITY 1
NO TYPE INFORMATION AVAILABLE FOR ??
A DEFAULT TYPE HAS BEEN ASSIGNED

DU5 -SEVERITY 1
REFERENCE TO UNDECLARED STRUCTURE TEMPLATE ??

D1 -SEVERITY 1
THE TYPE SPECIFICATION FOR THE PARAMETER ?? IS ILLEGAL

D10 -SEVERITY 1
TEMPORARY STATEMENTS MAY NOT APPEAR AT THE OPENING OF A DO CASE...END
GROUP

D11 -SEVERITY 1
TYPE SPECIFICATION OF ?? RENDERS NONHAL ATTRIBUTE ILLEGAL
ATTRIBUTE IGNORED.

D12 -SEVERITY 1
DECLARATION OF ?? SPECIFIES ATTRIBUTES INCOMPATIBLE WITH THE
NAME ATTRIBUTE.

D13 -SEVERITY 1
DECLARATION OF ?? HAS CONFLICTING FACTORED AND UNFACTORED NONHAL
ATTRIBUTES - UNFACTORED ATTRIBUTE WILL BE USED.

D2 -SEVERITY 1
THE PROCEDURE OR FUNCTION ?? MAY NOT BE DECLARED IN A COMPOOL

D3 -SEVERITY 1
THE DECLARATION OF PROCEDURE ?? MAY NOT POSSESS FACTORED ATTRIBUTES

D4 -SEVERITY 1
ARRAY SPECIFICATION ILLEGAL FOR ??

D5 -SEVERITY 1
?? HAS BOTH FACTORED AND NONFACTORED LOCK GROUP
SPECIFICATION - NON-FACTORED SPECIFICATION WILL
BE USED.

D6 -SEVERITY 1
THE DECLARATION OF ?? HAS BOTH FACTORED AND NON-FACTORED ARRAY SPECIFICATIONS;
THE NON-FACTORED SPECIFICATION WILL BE USED

D7 -SEVERITY 1
TEMPORARY STATEMENT DOES NOT APPEAR BEFORE THE FIRST EXECUTABLE STATEMENT
OF THE ENCLOSING DO...END GROUP

D8 -SEVERITY 1
DECLARATION OF ?? SPECIFIES TYPE OR ATTRIBUTES NOT LEGAL
FOR TEMPORARIES
|

D9 -SEVERITY 1
TEMPORARY ?? HAS AN ILLEGAL TYPE SPECIFICATION.

ERROR MESSAGES FOR MAJOR CLASSIFICATION E
CLASSIFICATION "E" ERRORS DEAL WITH EXPRESSIONS

- EA1 -SEVERITY 1
ARRAYNESS OF ?? IS INCONSISTENT WITH CURRENT ARRAYNESS OF EXPRESSION
- EB1 -SEVERITY 0
RESULT OF BIT CATENATION WILL BE LEFT TRUNCATED TO MAXIMUM BIT LENGTH
- EB2 -SEVERITY 1
LABEL ?? USED IN BIT OR EVENT EXPRESSION WAS NOT A PROGRAM OR TASK EVENT
- EC1 -SEVERITY 1
CROSS PRODUCT MUST BE BETWEEN THREE DIMENSIONAL VECTORS
- EC2 -SEVERITY 1
CROSS PRODUCT * USED WITHOUT A VECTOR AFTER IT
- EC3 -SEVERITY 1
CROSS PRODUCT * USED WITHOUT A VECTOR BEFORE IT
- ED1 -SEVERITY 1
DOT PRODUCT . USED WITHOUT A VECTOR AFTER IT
- ED2 -SEVERITY 1
DOT PRODUCT . USED WITHOUT A VECTOR BEFORE IT
- EL1 -SEVERITY 1
ONLY ARITHMETIC CONVERSION FUNCTIONS MAY POSSESS ARGUMENTS WITH REPEAT FACTORS
- EL2 -SEVERITY 1
REPETITION FACTOR OF EXPRESSION MUST BE AN UNARRAYED INTEGER OR SCALAR
EXPRESSION COMPUTABLE AT COMPILE TIME
- EM1 -SEVERITY 1
DIMENSIONS OF MATRIX OPERANDS IN EXPRESSION DISAGREE
- EM2 -SEVERITY 1
MATRIX ARITHMETIC TYPE CANNOT BE CONVERTED TO A CHARACTER STRING
- EM3 -SEVERITY 1
MATRIX-MATRIX MULTIPLICATION DIMENSION DISAGREEMENT
- EM4 -SEVERITY 1
INVERSE OF NON-SQUARE MATRIX ATTEMPTED
- EN1 -SEVERITY 1
THE ARGUMENT OF A NAME PSEUDO-FUNCTION MAY NOT BE A
NAME PSEUDO-FUNCTION.
- EN10 -SEVERITY 1
?? IS A TEMPORARY AND MAY NOT THEREFORE BE THE
ARGUMENT OF A NAME PSEUDO-FUNCTION.
- EN11 -SEVERITY 1
?? IS A CONSTANT OR INPUT PARAMETER AND MAY NOT
THEREFORE BE THE ARGUMENT OF A NAME PSEUDO-FUNCTION.
- EN12 -SEVERITY 1
ACCESS RIGHTS HAVE BEEN DENIED TO ?? - ITS USE
IN A NAME PSEUDO-FUNCTION IS THEREFORE ILLEGAL.

- EN13 -SEVERITY 1
?? IS NOT A MAJOR STRUCTURE BUT POSSESSES STRUCTURE COPIES - IT IS THEREFORE ILLEGAL AS THE ARGUMENT OF A NAME PSEUDO-VARIABLE.
- EN14 -SEVERITY 1
THE ARGUMENT OF A NAME PSEUDO-FUNCTION MAY NOT POSSESS A PRECISION MODIFIER.
- EN2 -SEVERITY 1
THE ARGUMENT OF A NAME PSEUDO-FUNCTION MAY NOT BE A SUBBIT PSEUDO-VARIABLE.
- EN3 -SEVERITY 1
THE PROCEDURE ?? POSSESSES PARAMETERS AND THUS MAY NOT BE USED AS THE ARGUMENT OF A NAME PSEUDO-FUNCTION.
- EN4 -SEVERITY 1
THE TYPE OF THE LABEL ?? IS ILLEGAL AS THE ARGUMENT OF A NAME PSEUDO-FUNCTION.
- EN5 -SEVERITY 1
NONHAL PROCEDURE OR FUNCTION ?? MAY NOT BE THE ARGUMENT OF A NAME PSEUDO-FUNCTION.
- EN6 -SEVERITY 1
PROCEDURE OR FUNCTION ?? IS NOT EXTERNAL AND THEREFORE MAY NOT BE THE ARGUMENT OF A NAME PSEUDO-FUNCTION.
- EN7 -SEVERITY 1
?? POSSESSES SUBSCRIPTING ILLEGAL FOR THE ARGUMENT OF A NAME PSEUDO-FUNCTION.
- EN8 -SEVERITY 1
?? HAS THE ATTRIBUTE DENSE AND MAY NOT THEREFORE BE THE ARGUMENT OF A NAME PSEUDO-FUNCTION.
- EN9 -SEVERITY 1
?? IS A MINOR NODE OF A STRUCTURE AND MAY NOT THEREFORE BE THE ARGUMENT OF A NAME PSEUDO-FUNCTION.
- EO1 -SEVERITY 1
ILLEGAL PRODUCT: OUTER PRODUCT TIMES A VECTOR
- EO2 -SEVERITY 1
A PRODUCT INVOLVING BOTH CROSS AND OUTER PRODUCTS IS INDICATED. USE MORE PARENTHESES.
- EO3 -SEVERITY 1
A PRODUCT INVOLVING BOTH DOT AND OUTER PRODUCTS IS INDICATED. USE MORE PARENTHESIS.
- EV1 -SEVERITY 1
LENGTHS OF VECTOR OPERANDS IN EXPRESSION DISAGREE
- EV2 -SEVERITY 1
MATRIX-VECTOR MULTIPLICATION DIMENSION DISAGREEMENT
- EV3 -SEVERITY 1
VECTOR-MATRIX MULTIPLICATION DIMENSION DISAGREEMENT

- EV4 -SEVERITY 1
VECTOR MAY NOT HAVE AN EXPONENT
- EV5 -SEVERITY 1
VECTOR ARITHMETIC TYPE CANNOT BE CONVERTED TO A CHARACTER STRING
- E1 -SEVERITY 1
DIVISORS MAY ONLY BE OF INTEGER OR SCALAR TYPE
- E2 -SEVERITY 1
MATRIX MUST HAVE AN EXPONENT OF INTEGER TYPE KNOWN AT COMPILE TIME
- E3 -SEVERITY 1
EXPONENT MUST BE A SINGLE VALUED QUANTITY
- E4 -SEVERITY 1
DOT OR CROSS PRODUCT SYMBOL (. OR *) USED IN A PRODUCT NOT INVOLVING VECTORS
- E6 -SEVERITY 1
INCOMPATIBLE ARITHMETIC OPERAND TYPES IN EXPRESSION

ERROR MESSAGES FOR MAJOR CLASSIFICATION P
CLASSIFICATION "P" ERRORS DEAL WITH FORMAL PARAMETERS AND ARGUMENTS

- PD1 -SEVERITY 1
MATRIX DIMENSIONS OF ARGUMENT AND CORRESPONDING FORMAL PARAMETER DO NOT AGREE
- PD2 -SEVERITY 1
VECTOR LENGTHS OF ARGUMENT AND CORRESPONDING FORMAL PARAMETER DO NOT AGREE
- PD3 -SEVERITY 1
TREE ORGANIZATIONS OF STRUCTURE ARGUMENT AND CORRESPONDING FORMAL PARAMETER ARE NOT IDENTICAL
- PD4 -SEVERITY 1
ARRAYNESS OF FUNCTION ARGUMENT DOES NOT MATCH CURRENT ARRAYNESS OF EXPRESSION CONTAINING THE INVOCATION
- PD5 -SEVERITY 1
ARRAYNESS OF ARGUMENT AND CORRESPONDING FORMAL PARAMETER ARE NOT IDENTICAL
- PD6 -SEVERITY 1
ARGUMENT OF ?? FUNCTION IS NOT A SQUARE MATRIX
- PD7 -SEVERITY 1
A NAME PSEUDO-FUNCTION MAY NOT POSSESS MULTIPLE COPIES IF AN ARGUMENT OF A PROCEDURE CALL.
- PN1 -SEVERITY 1
FUNCTION ?? WAS INVOKED WITH TOO FEW ARGUMENTS
- PN2 -SEVERITY 1
FUNCTION ?? WAS INVOKED WITH TOO MANY ARGUMENTS
- PN3 -SEVERITY 2
?? USED MORE THAN ONCE AS A PARAMETER
- PN4 -SEVERITY 1
?? FUNCTION HAS INCORRECT NUMBER OF ARGUMENTS
- PS1 -SEVERITY 1
AN ASSIGN ARGUMENT OF A PROCEDURE CALL MAY NOT BE A SUBRIT PSEUDO VARIABLE
- PS2 -SEVERITY 1
THE STRUCTURE COPIES OF ASSIGN ARGUMENT ?? MUST BE SUBSCRIBTED AWAY
- PT0 -SEVERITY 1
NAME ARGUMENT HAS PROPERTIES INCOMPATIBLE WITH CORRESPONDING NAME FORMAL PARAMETER.
- PT1 -SEVERITY 1
TYPE OF FUNCTION ARGUMENT IS INCOMPATIBLE WITH TYPE OF CORRESPONDING FORMAL PARAMETER
- PT10 -SEVERITY 1
A NAME PSEUDO-FUNCTION MAY NOT BE THE ARGUMENT OF A BUILT-IN OR SHAPING/CONVERSION FUNCTION.
- PT12 -SEVERITY 1
ONE OF THE FOLLOWING IS TRUE:
.ARGUMENT CORRESPONDING TO NAME FORMAL PARAMETER

IS NOT A NAME PSEUDO-FUNCTION OR NULL;
.NAME PSEUDO-FUNCTION OR NULL ARGUMENT CORRESPONDS

TO FORMAL PARAMETER WITH NO NAME ATTRIBUTE.

- FT2 -SEVERITY 1
?? FUNCTION HAS AN ARGUMENT OF INCORRECT TYPE
- FT3 -SEVERITY 1
ILLEGAL TYPE FOR THE FUNCTION ??
- FT4 -SEVERITY 1
THE SIZE SPECIFICATION FOR THE RETURN TYPE OF FUNCTION ?? DISAGREES WITH THE PREVIOUSLY DECLARED SIZE
- FT6 -SEVERITY 1
THE STRUCTURE TEMPLATE INDICATED IN THE TYPE SPECIFICATION OF FUNCTION ?? DISAGREES WITH THE TEMPLATE USED IN A PREVIOUS DECLARATION
- FT7 -SEVERITY 1
CONFLICTING SINGLE/DOUBLE SPECIFICATION FOR THE FUNCTION ??
- FT8 -SEVERITY 1
THE FUNCTION ?? MAY NOT POSSESS A SUBSCRIPT OR PRECISION QUALIFIER.
- FT9 -SEVERITY 1
THE INVOCATION OF NAME PROCEDURE ?? MAY NOT POSSESS ARGUMENTS.

157<

ERROR MESSAGES FOR MAJOR CLASSIFICATION G
CLASSIFICATION "G" ERRORS DEAL WITH STATEMENT GROUPINGS (DO STATEMENTS)

- GB1 -SEVERITY 1
 BIT EXPRESSION IN ?? CLAUSE MUST BE BOOLEAN
- GC1 -SEVERITY 1
 CONTROL EXPRESSION IN A DO CASE MUST BE OF UNARRAYED INTEGER OR SCALAR TYPE
- GC2 -SEVERITY 1
 BIT EXPRESSION IN WHILE OR UNTIL CLAUSE MAY NOT BE ARRAYED
- GC3 -SEVERITY 1
 CONTROL EXPRESSIONS IN A DO FOR MUST BE OF UNARRAYED INTEGER OR SCALAR TYPE
- GE1 -SEVERITY 1
 EXIT IS EITHER NOT IN A DO...END GROUP, OR NO LABEL
 MATCH WAS FOUND.
- GE2 -SEVERITY 1
 REPEAT IS EITHER NOT IN A DO FOR...END OR DO WHILE/UNTIL...END
 GROUP, OR NO LABEL MATCH WAS FOUND.
- GE3 -SEVERITY 1
 EXIT CAUSES ILLEGAL BRANCHING OUT OF CODE BLOCK DEFINITION
- GE4 -SEVERITY 1
 REPEAT CAUSES ILLEGAL BRANCHING OUT OF CODE BLOCK DEFINITION
- GL1 -SEVERITY 1
 LABEL AFTER END STATEMENT DOES NOT MATCH DO STATEMENT LABEL
- GL2 -SEVERITY 1
 LABEL IS THE DESTINATION OF A GO TO FROM OUTSIDE THE ENCLOSING DO...END GROUP
- GL3 -SEVERITY 1
 GO TO STATEMENT CAUSES A BRANCH INTO A DO...END GROUP
- GV1 -SEVERITY 1
 CONTROL VARIABLE IN A DO FOR MUST BE OF UNARRAYED INTEGER OR SCALAR TYPE

ERROR MESSAGES FOR MAJOR CLASSIFICATION I
CLASSIFICATION "I" ERRORS ARE RELATED TO IDENTIFIERS

- IL1 -SEVERITY 1
 IDENTIFIER NAME MAY NOT END WITH AN UNDERSCORE CHARACTER
- IL2 -SEVERITY 1
 NAME TOO LONG - TRUNCATED
- IR1 -SEVERITY 1
 ILLEGAL REPLACEMENT FOR LOCAL NAME: ??
- IR10 -SEVERITY 3
 MAXIMUM NUMBER OF PARAMETERS FOR SOURCE MACRO DEFINITION EXCEEDED
- IR3 -SEVERITY 1
 MACRO EXPANSION TOO LONG
- IR5 -SEVERITY 1
 DUPLICATE REPLACE FOR ??
- IR6 -SEVERITY 1
 MACRO NAME ?? NOT DEFINED
- IR7 -SEVERITY 2
 REPLACE PARAMETER STRING TOO LONG; REPLACE NOT PERFORMED
- IR8 -SEVERITY 2
 INCORRECT NUMBER OF PARAMETERS FOR MACRO CALL; REPLACEMENT NOT PERFORMED
- IR9 -SEVERITY 3
 MACRO EXPANSION STACK OVERFLOW; RECURSIVE DEFINITION LIKELY
- IS1 -SEVERITY 1
 ILLEGAL CONSTRUCTION OF QUALIFIED STRUCTURE NAME

ERROR MESSAGES FOR MAJOR CLASSIFICATION L
CLASSIFICATION "L" ERRORS DEAL WITH LITERALS

- LB1 -SEVERITY 1
BIT CONSTANTS MAY NOT BE LONGER THAN 32 BITS
- LB2 -SEVERITY 1
DECIMAL BIT CONSTANT MUST SPECIFY OR IMPLY A REPETITION FACTOR OF 1
- LB3 -SEVERITY 1
ILLEGAL DECIMAL STRING IN DECIMAL BIT CONSTANT
- LB4 -SEVERITY 1
ILLEGAL CHARACTER IN DECIMAL BIT CONSTANT
- LB5 -SEVERITY 1
ILLEGAL CHARACTER IN BINARY BIT CONSTANT
- LB6 -SEVERITY 1
ILLEGAL CHARACTER IN OCTAL BIT CONSTANT
- LB7 -SEVERITY 1
ILLEGAL CHARACTER IN HEXADECIMAL BIT CONSTANT
- LB8 -SEVERITY 1
REPETITION FACTOR OF A BIT LITERAL MUST BE GREATER THAN ZERO
- LC2 -SEVERITY 1
?? NOT EXPRESSIBLE INTERNALLY
- LF1 -SEVERITY 1
ILLEGAL NUMERIC LITERAL CONSTRUCTION
- LF2 -SEVERITY 1
ONLY ONE DECIMAL POINT ALLOWED
- LF3 -SEVERITY 1
TOO MANY SIGNIFICANT DIGITS - 74 ALLOWED
- LF5 -SEVERITY 1
EXPONENT INDICATOR BUT NO EXPONENT DIGITS
- LS1 -SEVERITY 0
CHARACTER STRING TOO LONG - TRUNCATED TO 255 CHARACTERS
- LS2 -SEVERITY 1
REPETITION FACTOR OF A CHARACTER LITERAL IS NOT GREATER THAN ZERO

ERROR MESSAGES FOR MAJOR CLASSIFICATION M
CLASSIFICATION "M" ERRORS DEAL WITH MULTI-LINE FORMATS

MC1 -SEVERITY 1
ILLEGAL CONTEXT FOR OVERPUNCH

MC2 -SEVERITY 1
OVERPUNCH ILLEGAL ON FUNCTION NAMES

MC3 -SEVERITY 1
OVERPUNCH ILLEGAL ON REPLACED NAME

MC4 -SEVERITY 0
OVERPUNCH NOT VALID ON RESERVED WORD ??

MC5 -SEVERITY 0
OVERPUNCH ILLEGAL ON PARAMETER ??

MC6 -SEVERITY 0
OVERPUNCH ILLEGAL IN DECLARATION OF ??

ME1 -SEVERITY 3
EXPONENT STRING OVERFLOW

ME2 -SEVERITY 1
E-LINE CHARACTER MORE THAN ONE LINE ABOVE PRECEDING CHARACTER

ME3 -SEVERITY 1
E-LINE OVERLAPS M-LINE

ME4 -SEVERITY 1
OVERLAPPING E-LINE CHARACTERS

MO1 -SEVERITY 0
OVERPUNCH ILLEGAL

MO2 -SEVERITY 1
INVALID OVERPUNCH ON ??

MO3 -SEVERITY 0
MULTIPLE OVERPUNCHES NOT VALID - FIRST ACCEPTED

MO4 -SEVERITY 1
USER SUPPLIED OVERPUNCH CHARACTER NOT VALID - IGNORED

MS1 -SEVERITY 3
SUBSCRIPT STRING OVERFLOW

MS2 -SEVERITY 1
S-LINE CHARACTER MORE THAN ONE LINE LOWER THAN PRECEDING CHARACTER

MS3 -SEVERITY 1
S-LINE OVERLAPS M-LINE

MS4 -SEVERITY 1
OVERLAPPING S-LINE CHARACTERS

M1 -SEVERITY 0
ILLEGAL CARD TYPE - CHANGED TO A COMMENT

161<

D-23

M2 -SEVERITY 1
INVALID SEQUENCE OF CARD TYPES

M3 -SEVERITY 0
COMMENT LONGER THAN 256 CHARACTERS - HAS BEEN TRUNCATED

162<

D-24

ERROR MESSAGES FOR MAJOR CLASSIFICATION P
CLASSIFICATION "P" ERRORS INDICATE FLOW CONTROL PROBLEMS

- PA1 -SEVERITY 0
A MEMBER CORRESPONDING TO THE PROGRAM IDENTIFICATION ?? CANNOT BE FOUND
IN THE PROGRAM ACCESS FILE. NO ACCESS VALIDATION WILL BE PERFORMED.
- PA2 -SEVERITY 0
PROCESSING OF THE PROGRAM ACCESS FILE FOR PROGRAM ID ?? HAS
CAUSED DETECTION OF ONE OR MORE ERRORS AND/OR INCONSISTANCIES
WHICH ARE LISTED BELOW:
- PC1 -SEVERITY 1
COMPOOL BLOCK CONTAINS STATEMENT(S) OTHER THAN DECLARATIONS
- PC2 -SEVERITY 1
COMPOOL TEMPLATE CONTAINS STATEMENT(S) OTHER THAN DECLARATIONS
- PD1 -SEVERITY 1
ILLEGAL DECLARATION FOR THE PARAMETER ??
- PE1 -SEVERITY 1
EXTERNAL TEMPLATES MUST NOT APPEAR WITHIN A BLOCK DEFINITION
- PE2 -SEVERITY 1
EXTERNAL TEMPLATES MUST NOT BE PLACED AFTER A BLOCK DEFINITION
- PF1 -SEVERITY 1
RETURN FROM FUNCTION BLOCK MUST BE FOLLOWED BY AN EXPRESSION
- PF2 -SEVERITY 1
RETURN MAY ONLY BE FOLLOWED BY AN EXPRESSION IN A FUNCTION BLOCK
- PF3 -SEVERITY 1
EXPRESSION TO BE RETURNED MAY NOT POSSESS ARRAYNESS
- PF4 -SEVERITY 1
ILLEGAL TYPE CONVERSION OF RETURNED EXPRESSION REQUIRED
- PF5 -SEVERITY 1
MATRIX DIMENSIONS OF FUNCTION DISAGREE WITH THOSE OF RETURN EXPRESSION
- PF6 -SEVERITY 1
VECTOR LENGTH OF FUNCTION DISAGREES WITH THAT OF RETURN EXPRESSION
- PF7 -SEVERITY 1
TREE ORGANIZATION OF FUNCTION DOES NOT MATCH THAT OF RETURN EXPRESSION
- PF8 -SEVERITY 1
IN A DEREFERENCING CONTEXT, THE BLOCK NAME ?? MAY NOT
POSSESS SUBSCRIPTING HAVING ARRAYNESS
- PF9 -SEVERITY 1
RETURN EXPRESSION MAY NOT BE A NAME PSEUDO-FUNCTION
OR NULL.
- PL1 -SEVERITY 1
THE FUNCTION ?? HAS BEEN DECLARED BUT NOT DEFINED
- PL2 -SEVERITY 2
?? IS A DUPLICATE LABEL

PL3 -SEVERITY 1
 LABEL CN CLOSE DOES NOT MATCH BLOCK DEFINITION LABEL: ??

PL4 -SEVERITY 1
 FUNCTION LABEL CONFLICT

PL5 -SEVERITY 1
 LABEL ?? IS NOT DEFINED WITHIN THE CURRENT SCOPE

PL6 -SEVERITY 1
 A DEFINITION BLOCK FOR THE PROCEDURE OR TASK ?? IS ABSENT FROM
 THE COMPILATION

PL7 -SEVERITY 1
 USED IN A CALL STATEMENT, PROCEDURE LABEL ?? MAY NOT
 POSSESS ARRAYNESS.

PM1 -SEVERITY 2
 DUPLICATE DEFINITION FOR ??

PM3 -SEVERITY 1
 EARLIER DEFINITION OVERRIDDEN FOR ??

PM4 -SEVERITY 0
 OUTER DEFINITION OVERRIDDEN FOR ??

PD1 -SEVERITY 1
 A ?? DEFINITION MUST BE THE OUTERMOST BLOCK DEFINITION

PP10 -SEVERITY 2
 INLINE FUNCTIONS MAY NOT BE NESTED WITHIN INLINE FUNCTION BLOCKS.

PP11 -SEVERITY 2
 INLINE FUNCTIONS MUST NOT APPEAR IN EXPRESSIONS WHICH ARE
 REQUIRED TO BE COMPILE TIME EVALUABLE - THIS ERROR IS RECOVERABLE

PP2 -SEVERITY 1
 BLOCK DEFINITION IS NOT THE FIRST OUTERMOST BLOCK DEFINITION

PP3 -SEVERITY 1
 A ?? DEFINITION CANNOT BE AN OUTERMOST BLOCK DEFINITION

PP4 -SEVERITY 1
 NO BLOCK DEFINITIONS WERE ENCOUNTERED IN COMPILATION

PP5 -SEVERITY 1
 AN INLINE FUNCTION MAY NOT CONTAIN AN I/O STATEMENT.

PP6 -SEVERITY 1
 AN INLINE FUNCTION MAY NOT CONTAIN A REAL TIME STATEMENT.

PP7 -SEVERITY 1
 AN INLINE FUNCTION MAY NOT CONTAIN A PROCEDURE CALL.

PP8 -SEVERITY 1
 AN INLINE FUNCTION MAY NOT CONTAIN A USER FUNCTION INVOCATION.

PP9 -SEVERITY 1
 AN INLINE FUNCTION MAY NOT CONTAIN A PROCEDURE OR FUNCTION
 DEFINITION BLOCK.

PS1 -SEVERITY 1
EXTERNAL PROCEDURE/FUNCTION TEMPLATE CONTAINS STATEMENT(S)
OTHER THAN DECLARATIONS

PS2 -SEVERITY 1
ONLY PROCEDURES OR FUNCTIONS MAY BE DESIGNATED ??

PS3 -SEVERITY 1
ILLEGAL ACCESS ATTRIBUTE OR BLOCK HEADER.

PS4 -SEVERITY 0
THE ACCESS ATTRIBUTE MAY ONLY BE USED ON THE DEFINITION
OF AN OUTERMOST BLOCK.

PS5 -SEVERITY 1
THE PROGRAM NAMED ?? IS ACCESS CONTROLLED. THE CURRENT
COMPILATION UNIT IS NOT AUTHORIZED TO SCHEDULE THIS PROGRAM.

PS6 -SEVERITY 1
THE PROCEDURE NAMED ?? IS ACCESS CONTROLLED. THE CURRENT
COMPILATION UNIT IS NOT AUTHORIZED TO CALL THIS PROCEDURE.

PS7 -SEVERITY 1
THE FUNCTION NAMED ?? IS ACCESS CONTROLLED.
THE CURRENT COMPILATION UNIT IS NOT AUTHORIZED TO INVOKE
THIS FUNCTION.

PS8 -SEVERITY 1
THE VARIABLE NAMED ?? IS ACCESS CONTROLLED. THE CURRENT
COMPILATION UNIT IS NOT AUTHORIZED TO CHANGE THE VALUE
OF THIS VARIABLE.

PS9 -SEVERITY 1
VARIABLE ?? IS DEFINED WITHIN A COMPOOL BLOCK WHICH IS
ACCESS PROTECTED. THE VARIABLE MAY NOT BE USED BY THIS
COMPILATION UNIT.

PT1 -SEVERITY 1
TASK DEFINITIONS OR DECLARATIONS MAY ONLY APPEAR IN
THE OUTER MOST BLOCK OF A PROGRAM COMPILATION

PU3 -SEVERITY 1
INVOCATIONS IN AN UPDATE BLOCK OF PROCEDURES OR USER FUNCTIONS DEFINED
OUTSIDE THE BLOCK ARE ILLEGAL

P1 -SEVERITY 1
END-OF-FILE AT INVALID POINT IN SOURCE TEXT

P3 -SEVERITY 0
BLOCK SUMMARY TABLE OVERFLOW

P4 -SEVERITY 2
CONFLICTING USE OF ??

P5 -SEVERITY 1
TOO MANY MACRO EXPANSIONS FOR ??

P6 -SEVERITY 0
PROGRAM LAYOUT TABLE EXCEEDED

PS -SEVERITY 1
THE FOLLOWING SYMBOL IS SYNTACTICALLY ILLEGAL IN THE CONTEXT USED: ??
ERROR RECOVERY MAY CAUSE SUBSEQUENT SPURIOUS ERRORS

166<

D-28

ERROR MESSAGES FOR MAJOR CLASSIFICATION Q
CLASSIFICATION "Q" ERRORS DEAL WITH SHAPING FUNCTIONS

- OA1 -SEVERITY 1
 ARRAYNESS OF SINGLE ARGUMENT OF INTEGER/SCALAR CONVERSION FUNCTION DOES NOT
 MATCH THAT OF EXPRESSION CONTAINING FUNCTION
- OA2 -SEVERITY 1
 ARRAYNESS OF RESULT OF INTEGER/SCALAR CONVERSION FUNCTION IS UNCOMPUTABLE
- OA3 -SEVERITY 1
 SPECIFIED ARRAYNESS OF INTEGER/SCALAR CONVERSION FUNCTION IS INCONSISTENT WITH
 NUMBER OF DATA ELEMENTS SUPPLIED IN ARGUMENT LIST
- OA4 -SEVERITY 1
 ARRAYNESS OF RESULT OF INTEGER/SCALAR CONVERSION FUNCTION DOES NOT MATCH THAT
 OF EXPRESSION CONTAINING FUNCTION
- OD1 -SEVERITY 1
 DIMENSIONS OF VECTOR/MATRIX CONVERSION FUNCTION DO NOT AGREE WITH THE NUMBER
 OF DATA ELEMENTS SUPPLIED IN THE ARGUMENT LIST
- OD2 -SEVERITY 1
 BIT OR CHARACTER CONVERSION FUNCTION MAY ONLY HAVE ONE ARGUMENT
- OS1 -SEVERITY 1
 COLONS AND SEMICOLONS MAY NOT APPEAR IN SUBSCRIPT OF CONVERSION FUNCTIONS
- OS10 -SEVERITY 1
 BIT OR CHARACTER CONVERSION FUNCTION MAY ONLY HAVE ONE SUBSCRIPT
- OS11 -SEVERITY 1
 SUBBIT CONVERSION FUNCTION MAY ONLY HAVE ONE SUBSCRIPT
- OS12 -SEVERITY 1
 COLONS AND SEMICOLONS MAY NOT APPEAR IN THE SUBSCRIPT OF A SUBBIT
 PSEUDO-VARIABLE
- OS13 -SEVERITY 1
 SUBSCRIPT OF A SUBBIT PSEUDO-VARIABLE MAY NOT CONTAIN A PRECISION
 MODIFIER
- OS2 -SEVERITY 1
 MATRIX CONVERSION FUNCTION DOES NOT HAVE TWO SUBSCRIPTS
- OS3 -SEVERITY 1
 VECTOR CONVERSION FUNCTION DOES NOT HAVE ONE SUBSCRIPT
- OS4 -SEVERITY 1
 INTEGER OR SCALAR CONVERSION FUNCTION HAS MORE THAN MAXIMUM PERMITTED
 NUMBER OF SUBSCRIPTS
- OS5 -SEVERITY 1
 SUBSCRIPT OF ARITHMETIC CONVERSION FUNCTION IS NOT A SINGLE INDEX
- OS6 -SEVERITY 1
 SUBSCRIPT OF ARITHMETIC CONVERSION FUNCTION MAY NOT CONTAIN # VALUES
- OS7 -SEVERITY 1
 SUBSCRIPT OF ARITHMETIC CONVERSION FUNCTION MUST BE AN UNARRAYED INTEGER/SCALAR
 EXPRESSION COMPUTABLE AT COMPILE TIME

- OS8 -SEVERITY 1
VALUE OF SUBSCRIPT OF ARITHMETIC CONVERSION FUNCTION LIES OUTSIDE LEGAL RANGE
- OS9 -SEVERITY 1
SUBSCRIPT OF BIT OR CHARACTER CONVERSION FUNCTION MAY NOT CONTAIN A PRECISION QUALIFIER
- OX1 -SEVERITY 1
CONVERSION FUNCTIONS MAY NOT HAVE ARGUMENTS OF STRUCTURE TYPE
- OX2 -SEVERITY 1
MATRIX/VECTOR CONVERSION FUNCTIONS MAY NOT HAVE ARGUMENTS OF BIT TYPE
- OX3 -SEVERITY 1
MATRIX/VECTOR CONVERSION FUNCTIONS MAY NOT HAVE ARGUMENTS OF CHARACTER TYPE
- OX4 -SEVERITY 1
MATRIX OR VECTOR ARGUMENT IS ILLEGAL IN BIT OR CHARACTER CONVERSION FUNCTION
- OX5 -SEVERITY 1
CHARACTER CONVERSION FUNCTION WITH RADIX DOES NOT HAVE ARGUMENT OF BIT TYPE
- OX6 -SEVERITY 1
BIT CONVERSION FUNCTION WITH RADIX DOES NOT HAVE ARGUMENT OF CHARACTER TYPE
- OX7 -SEVERITY 1
IN AN ASSIGNMENT CONTEXT THE ARGUMENT OF A SUBBIT PSEUDO-VARIABLE MAY NOT ITSELF BE A SUBBIT PSEUDO-VARIABLE
- OX8 -SEVERITY 1
ARGUMENT OF ILLEGAL TYPE IN SUBBIT PSEUDO-VARIABLE
- OX9 -SEVERITY 1
THE ARGUMENT OF A SUBBIT PSEUDO-VARIABLE MAY NOT BE A NAME PSEUDO-FUNCTION.

ERROR MESSAGES FOR MAJOR CLASSIFICATION R
CLASSIFICATION "R" ERRORS ARE RELATED TO REAL-TIME STATEMENT ERRORS

RE1 -SEVERITY 1
ILLEGAL FORM OR VALUE OF ON ERROR SUBSCRIPT

RE2 -SEVERITY 1
ILLEGAL FORM OR VALUE OF SEND ERROR SUBSCRIPT

RE3 -SEVERITY 1
TOO MANY ON ERROR STATEMENTS ACTIVE

RT1 -SEVERITY 1
SCHEDULE STATEMENT CONTAINS AN ILLEGAL FORM OF ?? TIMING EXPRESSION

RT10 -SEVERITY 1
AN UNLATCHED EVENT MAY NOT BE SET OR RESET

RT11 -SEVERITY 1
USED IN A REAL TIME STATEMENT OR AS A PROCESS EVENT, LABEL
?? MAY NOT POSSESS ARRAYNESS.

RT14 -SEVERITY 1
?? IS A NAME LABEL TYPE AND MAY NOT THEREFORE BE USED AS
A PROCESS EVENT

RT2 -SEVERITY 1
WHILE EXPRESSION MAY NOT BE A TIMING EXPRESSION

RT3 -SEVERITY 1
SCHEDULE STATEMENT CONTAINS AN ILLEGAL FORM OF ?? EVENT EXPRESSION

RT4 -SEVERITY 1
?? STATEMENT CONTAINS ILLEGAL PRIORITY EXPRESSION

RT5 -SEVERITY 1
SCHEDULE STATEMENT CONTAINS DUPLICATED AT/IN/ON EXPRESSIONS

RT6 -SEVERITY 1
WAIT STATEMENT CONTAINS ILLEGAL FORM OF ?? EXPRESSION

RT7 -SEVERITY 1
EVENT MUST BE SIGNALLED ON/OFF OR ITS BINARY EQUIVALENT

RT8 -SEVERITY 1
AN ARRAYED EVENT MAY NOT BE SIGNALLED

RT9 -SEVERITY 1
USED IN A REAL TIME STATEMENT OR AS A PROCESS EVENT, LABEL
?? MUST BE A PROGRAM OR TASK.

RU1 -SEVERITY 1
SIGNAL STATEMENTS ARE THE ONLY REAL-TIME STATEMENTS WHICH MAY APPEAR INSIDE
AN UPDATE BLOCK

ERROR MESSAGES FOR MAJOR CLASSIFICATION S
CLASSIFICATION "S" ERRORS INDICATE INCORRECT SUBSCRIPT USAGE

SC1 -SEVERITY 1
?? HAS TOO MANY STRUCTURE SUBSCRIPTS

SC2 -SEVERITY 1
?? HAS TOO MANY ARRAY SUBSCRIPTS

SC3 -SEVERITY 1
?? HAS TOO FEW ARRAY SUBSCRIPTS

SC4 -SEVERITY 1
?? HAS TOO MANY COMPONENT SUBSCRIPTS

SC5 -SEVERITY 1
?? HAS TOO FEW COMPONENT SUBSCRIPTS

SP1 -SEVERITY 1
SUBSCRIPTING CONTAINS MORE THAN ONE LIST OF STRUCTURE SUBSCRIPTS

SP2 -SEVERITY 1
SUBSCRIPTING CONTAINS MORE THAN ONE LIST OF ARRAY SUBSCRIPTS

SP3 -SEVERITY 1
SUBSCRIPT CONTAINS LEADING COLON, OR A COLON PRECEDED BY A SEMICOLON,
COLON, OR COMMA

SP4 -SEVERITY 1
SUBSCRIPT CONTAINS LEADING SEMICOLON, OR A SEMICOLON PRECEDED BY A SEMICOLON,
COLON, OR COMMA

SP5 -SEVERITY 1
SUBSCRIPT CONTAINS A LEADING COMMA, OR A COMMA PRECEDED BY A SEMICOLON, COLON,
OR COMMA

SP6 -SEVERITY 1
SUBSCRIPT IS EMPTY OR CONTAINS A TRAILING COMMA

SQ1 -SEVERITY 1
?? IS OF INCORRECT TYPE TO POSSESS A PRECISION QUALIFIER

SQ2 -SEVERITY 1
SUBSCRIPTED VARIABLE ?? MAY NOT POSSESS A PRECISION MODIFIER

SQ3 -SEVERITY 1
?? IS IN AN ASSIGNMENT CONTEXT AND THEREFOPE MAY NOT POSSESS A
PRECISION QUALIFIER

SR1 -SEVERITY 1
SIZE OF PARTITION IN A SUBSCRIPT OF ?? WAS UNKNOWN

SR2 -SEVERITY 1
SIZE OF PARTITION IN SUBSCRIPT OF ?? IS EITHER LESS THAN 2 OR
PRODUCED AN INDEX VALUE GREATER THAN THE MAXIMUM ALLOWABLE

SR3 -SEVERITY 1
THE VALUE OF A SUBSCRIPT OF ?? WAS GREATER THAN THE CORRESPONDING DIMENSION

SR4 -SEVERITY 1
THE VALUE OF A SUBSCRIPT OF ?? WAS LESS THAN 1

170<

SR5 -SEVERITY 1
?? CONTAINED AN ILLEGAL # SUBSCRIPT

SR6 -SEVERITY 1
INDEX VALUE IN SUBSCRIPT OF ?? IS UNKNOWN

SS1 -SEVERITY 1
IN SUBSCRIPT OF ?? ONLY TRAILING ASTERISKS MAY BE OMITTED

ST1 -SEVERITY 1
A SUBSCRIPT OF ?? WAS NOT OF INTEGER OR SCALAR TYPE

SV1 -SEVERITY 1
SUBSCRIPTING OF ?? IS ILLEGAL IN CONTEXT OF USE AS AN ASSIGN ARGUMENT

SV2 -SEVERITY 0
USER SUPPLIED OVERPUNCH NOT CONSISTENT WITH SUBSCRIPTING FOR VARIABLE. ??

SV3 -SEVERITY 1
?? MAY NOT POSSESS SUBSCRIPTS

ERROR MESSAGES FOR MAJOR CLASSIFICATION T
CLASSIFICATION "T" ERRORS DEAL WITH INPUT/OUTPUT STATEMENTS

- TC1 -SEVERITY 1
ARGUMENT OF I/O CONTROL FUNCTION MUST BE OF UNARRAYED INTEGER OR SCALAR TYPE
- TD1 -SEVERITY 1
I/O DEVICE NUMBER IS NOT IN RANGE 0 THROUGH ??
- TD2 -SEVERITY 1
RECORD ADDRESS IS NOT AN UNARRAYED INTEGER OR SCALAR
- T1 -SEVERITY 1
VARIABLE IN READALL IS NOT OF CHARACTER TYPE
- T2 -SEVERITY 1
VARIABLE IN READ MAY NOT BE OF EVENT TYPE
- T3 -SEVERITY 1
VARIABLE IN READ/READALL MAY NOT BE A SUBBIT PSEUDO-VARIABLE
- T4 -SEVERITY 1
A FILE STATEMENT MAY NOT READ INTO A SUBBIT PSEUDO-VARIABLE
- T5 -SEVERITY 1
AN I/O STATEMENT MAY NOT CONTAIN A NAME PSEUDO-FUNCTION
OR NULL.
- T6 -SEVERITY 0
I/O STATEMENT CONTAINS A STRUCTURE WHOSE TEMPLATE CONTAINS
AT LEAST ONE TERMINAL NODE WITH THE NAME ATTRIBUTE.
|
- T7 -SEVERITY 1
IN FILE STATEMENT VARIABLE TO BE READ POSSESSES
AN ILLEGAL SUBSCRIPT.
- T8 -SEVERITY 1
IN FILE STATEMENT VARIABLE TO BE READ INTO MAY ONLY
POSSESS MULTIPLE COPIES IF IT IS A MAJOR STRUCTURE
NAME.
|

ERROR MESSAGES FOR MAJOR CLASSIFICATION U
CLASSIFICATION "U" ERRORS DEAL WITH UPDATE BLOCKS

- UI1 -SEVERITY 1
 ?? IS LOCKED: IT MAY ONLY APPEAR IN UPDATE BLOCKS
 OR ASSIGN ARGUMENT LISTS,
- UI2 -SEVERITY 1
 UPDATE BLOCK DEFINITION MAY NOT APPEAR INSIDE AN UPDATE BLOCK
- UP1 -SEVERITY 1
 THE PROCEDURE, TASK, OR PROGRAM ?? MAY NOT BE INVOKED WITHIN THE
 CURRENT UPDATE BLOCK
- UP2 -SEVERITY 1
 UPDATE BLOCKS MAY NOT CONTAIN RETURN STATEMENTS
- UP3 -SEVERITY 1
 THE FUNCTION ?? MAY NOT BE INVOKED WITHIN THE CURRENT UPDATE BLOCK
- UT1 -SEVERITY 1
 I/O STATEMENTS ARE ILLEGAL INSIDE UPDATE BLOCKS

.....

ERROR MESSAGES FOR MAJOR CLASSIFICATION V
CLASSIFICATION "V" ERRORS ARE RELATED TO COMPILE-TIME VARIABLE ERRORS

- VA1 -SEVERITY 1
COMPILE TIME INTEGER/SCALAR ADDITION FAILED
- VA2 -SEVERITY 1
COMPILE-TIME INTEGER/SCALAR SUBTRACTION FAILED
- VA3 -SEVERITY 1
COMPILE TIME INTEGER/SCALAR MULTIPLICATION FAILED
- VA4 -SEVERITY 1
COMPILE-TIME INTEGER/SCALAR DIVISION FAILED
- VA5 -SEVERITY 1
COMPILE TIME INTEGER/SCALAR EXPONENTIATION FAILED
- VC1 -SEVERITY 0
COMPILE-TIME CATENATION PRODUCED TOO LONG A CHARACTER STRING -
TRUNCATED TO 255 CHARACTERS
- VE1 -SEVERITY 1
AN EXPRESSION NOT COMPUTABLE AT COMPILE-TIME HAS BEEN USED IN A CONTEXT WHERE
A VALUE MUST BE KNOWN
- VF1 -SEVERITY 1
COMPILE-TIME EVALUATION OF ?? FUNCTION FAILED

ERROR MESSAGES FOR MAJOR CLASSIFICATION X
CLASSIFICATION "X" ERRORS DEAL WITH IMPLEMENTATION DEPENDENT FEATURES

- XA1 -SEVERITY 0
 ONLY ONE PROGRAM IDENTIFICATION DIRECTIVE IS ALLOWED IN A COMPILATION
 UNIT.
- XA2 -SEVERITY 0
 THE PROGRAM IDENTIFICATION DIRECTIVE DOES NOT CONTAIN A VALID IDENTIFICATION.
 THE DIRECTIVE MUST BE OF THE FORM:
 D PROGRAM ID=<ID>
- XA3 -SEVERITY 0
 A PROGRAM IDENTIFICATION DIRECTIVE MUST APPEAR FOLLOWING ANY EXTERNAL
 TEMPLATES AND PRIOR TO THE BEGINNING OF THE PRIMARY UNIT OF COMPILATION.
 THE CURRENT DIRECTIVE IS OUT OF PLACE AND WILL NOT BE PROCESSED.
- XD1 -SEVERITY 0
 UNINTELLIGIBLE INFORMATION IN DEVICE DIRECTIVE
- XD2 -SEVERITY 0
 DUPLICATE DEVICE DIRECTIVE FOR CHANNEL ??
- XD3 -SEVERITY 0
 DEVICE DIRECTIVE DOES NOT CONTAIN A VALID CHANNEL INDICATION
- XD4 -SEVERITY 0
 CHANNEL NUMBERS MUST BE IN RANGE 0 TO 9
- XI1 -SEVERITY 0
 NESTED INCLUDE DIRECTIVES NOT ALLOWED
- XI2 -SEVERITY 0
 INCLUDE DIRECTIVE DOES NOT CONTAIN A NAME
- XI3 -SEVERITY 0
 ?? NOT IN INCLUDE LIBRARY
- XO1 -SEVERITY 0
 D CARD CONTAINS UNKNOWN DIRECTIVE
- XV1 -SEVERITY 1
 LAST LINE OF TEMPLATE LIBRARY MEMBER ?? IS NOT A VALID
 VERSION DIRECTIVE

PHASE II ERRORS

Compiler Limits

SEVERITY 2
DATA STORAGE CAPACITY EXCEEDED

SEVERITY 2
INDIRECT STACK OVERFLOW

SEVERITY 2
CHARACTER LITERAL BUFFER OVERFLOW

SEVERITY 2
CONSTANT TABLE OVERFLOW

SEVERITY 1
TOO MANY EXTERNAL NAMES

SEVERITY 1
STORAGE DESCRIPTOR STACK OVERFLOW

SEVERITY 1
EXCEEDED TEMPORARY SPACE

SEVERITY 2
STATEMENT LABELS ALL IN USE

SEVERITY 2
SUBPROGRAM STACK OVERFLOW

SEVERITY 1
EXCEEDED ON ERROR STACK SIZE

SEVERITY 1
TOO MANY UNIQUE OPERANDS IN EVENT EXPRESSION

SEVERITY 1
EXCEEDED ARGUMENT STACK SIZE

SEVERITY 1
EXCEEDED SHAPING FUNCTION DIM STACK

SEVERITY 1
EVENT EXPRESSION TOO LONG

SEVERITY 1
FLOW LABEL TABLE OVERFLOW

User Errors

SEVERITY 1
SIZE CONFLICT ON VECTOR/MATRIX PARAMETER #N

SEVERITY 1
ASSIGN PARAMETER NOT SYMBOL

SEVERITY 1
DATA TYPE CONFLICT ON PARAMETER #N

SEVERITY 1
ARRAYNESS CONFLICT ON PARAMETER #N

SEVERITY 1
ARRAY SIZE CONFLICT ON PARAMETER #N

SEVERITY 1
NOT ASSIGN PARAMETER

SEVERITY 1
STATEMENT CONTAINS PHASE I ERROR

SEVERITY 1
UNIMPLEMENTED FEATURE OF HAL/S CALLED FOR

SEVERITY 1
MALFORMED TEMPLATE, WALK INHIBITED

SEVERITY 1
STRUCTURE TEMPLATES DO NOT MATCH

SEVERITY 1
STAR SIZE ARRAY TEMPORARY NOT ALLOWED

SEVERITY 1
STRUCTURE COPYNESS CONFLICT ON PARAMETER #N

SEVERITY 1
STRUCTURE COPY SIZE CONFLICT ON PARAMETER #N

SEVERITY 1
REFERENCE TO UNDEFINED PROCEDURE OR FUNCTION

SEVERITY 1
INCORRECT NUMBER OF ARGUMENTS TO ??

SEVERITY 1
ILLEGAL NONHAL FUNCTION TYPE

SEVERITY 1
DO FOR ENDING INCOMPLETE DUE TO PREVIOUS ERROR RECOVERY

SEVERITY 1
ARRAYNESS CONFLICT

SEVERITY 1
COPYNESS CONFLICT

SEVERITY 1
ARRAYNESS INCONSISTENT WITH STATEMENT

SEVERITY 1
NON-ARRAYED ARGUMENT TO ARRAY FUNCTION

SEVERITY 1
INVALID ARGUMENT TYPE FOR ARRAY FUNCTION

SEVERITY 1
ILLEGAL ARGUMENT TO SIZE FUNCTION

SEVERITY 0
INITIAL STRING TOO LONG

SEVERITY 0
INITIAL VALUE TOO LARGE

SEVERITY 1 @Q
INITIALIZATION DATA TYPE MISMATCH

SEVERITY 1
NULL ONLY LEGAL NAME CONSTANT

SEVERITY 1
ILLEGAL NAME PARAMETER #N

SEVERITY 1
NAME PROCEDURE/FUNCTION MAY NOT HAVE ARGUMENTS

SEVERITY 1
NOT NAME PARAMETER

SEVERITY 0
TEMPORARY ?? NOT ADDRESSABLE

SEVERITY 0
?? NOT ADDRESSABLE

SEVERITY 1
SUBBIT SUBSCRIPT OUT OF RANGE

SEVERITY 1
CANNOT RETURN VALUE FROM NON-FUNCTION

Compiler Errors

SEVERITY 1
LITERAL PROCESSING FAILURE

SEVERITY 0
INDEX STACK USAGE INCONSISTENT

SEVERITY 0
UNMATCHED DO CASE ENDING

SEVERITY 0
UNMATCHED CASE LABEL

SEVERITY 0
UNMATCHED DO WHILE ENDING

SEVERITY 0
UNMATCHED DO FOR ENDING

SEVERITY 1
LEVEL MISMATCH ON PROC/FUNC/IO ARGUMENT

SEVERITY 1
LEVEL MISMATCH ON SHAPING FUNCTION ARGUMENT

SEVERITY 1
STRUCTURE NODE SIZE CONFLICT

SEVERITY 1
ARRAY LEVEL MISMATCH

SEVERITY 1
ARRAY ENDING MISMATCH

SEVERITY 1
MISMATCHED CLOSING

SEVERITY 1
LEVEL MISMATCH ON PROC/FUNC CALL

APPENDIX E.

Execution-time Errors

The following tables indicate runtime error conditions which may occur during execution of a HAL/S-360 program. The tables list any standard fixups performed by the runtime system. The form of the system action taken is indicated by the following code:

U	UNLIMITED
L	LIMITED
T	TERMINATE

Error Number	Message	Explanation	Standard Fixup	SYSTEM Action
1	EXPONENT OVERFLOW	the exponent of a scalar or of a vector/matrix element has overflowed	the result is set to the maximum value representable on the machine	L
2	EXPONENT UNDERFLOW	the exponent of a scalar or of a vector/matrix element has underflowed	the result is set to zero	U
3	SCALAR DIVISION BY ZERO	a scalar division by zero has occurred	the result is set to the maximum value representable on the machine	L
4	EXPONENTIATION OF ZERO TO POWER ≤ 0	a negative or zero power was specified	the result is set to zero	U
5	SQUARE ROOT HAS ARG < 0		the result is the square root of the absolute value of the argument	U

181A-2

182 < E-3

Error Number	Message	Explanation	Standard Fixup	SYSTEM Action
6	EXP FUNCTION HAS ARG > 174.673		the result is set to the maximum value representable on the machine	L
7	LOG FUNCTION HAS ARG < = 0.		if the argument was zero then the result is set to the maximum representable negative value, else it is set to the log of the absolute value of the arg.	L
8	SIN OR COS FUNCTION HAS $ \text{ARG} \left\{ \begin{array}{l} 2.621\text{E}5 \\ 1.126\text{E}15 \end{array} \right\} \text{ PI}$	the two figures are for single and double precision arguments respectively	the result is set to $\frac{\sqrt{2}}{2}$	L
9	SIN OR COSH FUNCTION HAS ARG > 175,366		the result is set to the maximum value representable	L
10	ARCSIN OR ARCCOS FUNCTION HAS $ \text{ARG} > 1$		the result is set to zero	L

Error Number	Message	Explanation	Standard Fixup	SYSTEM Action
11	TAN FUNCTION HAS $ ARG > \left\{ \begin{array}{l} 2.621E5 \\ 1.126E15 \end{array} \right\} \pi$	the two figures are for single and double precision respectively	the result is set to one	L
12	TAN FUNCTION TOO CLOSE TO SINGULARITY	the argument is too close to an odd multiple of $\pi/2$	the result is set to the maximum representable value	L
13	CASE VARIABLE OUT OF RANGE	the value of the case variable is either <1 or greater than the number of cases and there was no ELSE clause	the do case statement is ignored	L
14	CLOSE REACHED ON FUNCTION	no return statement was encountered prior to reaching the close of the function	none: IGNORE not allowed	T
15	SCALAR TOO LARGE FOR INTEGER CONVERSION		the result is set to the maximum representable value	L

100-E-4

184 E-5

Error Number	Message	Explanation	Standard Fixup	SYSTEM Action
16	INTEGER DIVISION BY ZERO	DIV OPERATOR HAS ZERO DIVISOR	The result is set to the maximum representable value	L
17	ILLEGAL CHARACTER SUBSCRIPT	Character component subscripting out-of-bounds	The out-of-bounds subscript(s) set to first or last character	U
18	BAD LENGTH IN LJUST OR RJUST	The length is less than the string length	Truncation to the specified length occurs on the left (RJUST) or right (LJUST)	U
19	MOD DOMAIN ERROR	In $A \text{ mod } B$ $B=0$ and $A<0$	Returns A (negative)	L
20	CHARACTER TO SCALAR CONVERSION	The string was not in standard internal format for integers or scalars.	The result is zero.	U

Error Number	Message	Explanation	Standard Fixup	SYSTEM Action
21	CHARACTER TO SCALAR CONVERSION DURING INPUT	Same as above.	The variable is left unchanged.	U
22	CHARACTER TO INTEGER CONVERSION	The string was not in standard internal format for integers	The result is zero.	U
23	CHARACTER TO INTEGER CONVERSION DURING INPUT	Same as above.	The variable is left unchanged	U
24	NEGATIVE BASE IN EXPONENTIATION	$A**B$ where $A < 0$	The result is $ A **B$	L
25	VECTOR/MATRIX DIVISION BY ZERO		The result is the original vector/matrix	L

105 < E-6

Error Number	MESSAGE	EXPLANATION	STANDARD FIXUP	SYSTEM Action
26	ILLEGAL BIT STRING DURING INPUT	Character other than blank, zeros or ones	Variables left unchanged	L
27	ARG OF INVERSE IS SINGULAR		The result is the identity matrix	L
28	ARG OF UNIT FUNCTION IS NULL VECTOR	Every component of the vector was zero in value.	The result is a vector all of whose components is zero.	L
29	ILLEGAL BIT STRING		Returns zero	L
30	Illegal SUBBIT subscript	value of routine SUBBIT subscript exceeded bit length	Subscript takes value of exceeded limit	L

185 < E-7

Error Number	Message	Explanation	Standard Fixup	SYSTEM Action
31	SYNAD ERROR: xxx	A transmission error was detected in a READ, READALL, or FILE stmt. SYNAD message "xxx" describes the error.	The block is accepted as is (ERROPT=ACC)	L
32	MISSING DD CARD - xxx	User did not provide the specified CHANNELn or FILEn DD card.	I/O on the channel is ignored	T
33	PRINT ON INPUT CHANNEL N or INPUT ON PRINT CHANNEL N	I/O was attempted on specified channel in print mode as well as read/readall mode.	The channel remains in the original mode, I/O in the new mode is ignored.	T
34	ILLEGAL SKIP COUNT ON CHANNEL n	The number of skips is negative.	Skip (0) is assumed	L
35	MARGIN VIOLATION ON CHANNEL n	A TAB or COLUMN I/O control function was specified which forced the device mechanism off the left or right margin.	The horizontal position is reset to either column one (left margin error) or just off the right-hand margin. On an I/O transfer this latter causes an immediate skip to the next line.	L

Error Number	Message	Explanation	Standard Fixup	SYSTEM Action
36	ILLEGAL PAGE COUNT ON CHANNEL n	A PAGE I/O control function with negative argument was specified	The PAGE command is ignored.	L
37	ILLEGAL LINE COUNT ON CHANNEL n	In a LINE I/O control function on an "unpaged" channel an argument less than the current line number was specified, or in the case of print mode a value greater than the number of lines per page was specified.	In the first case the LINE function is ignored. In the second, the effect is PAGE (1).	L
38	ILLEGAL NUMERIC FIELD ON CHANNEL n	An invalid character was found while reading a numeric field (Valid characters: 0-9, -, +, ., E, B, H)	The field with the invalid character(s) is skipped. The variable remains unchanged as if a null field were encountered.	L
39	ILLEGAL BIT OR CHARACTER STRING	In READ mode, character/bit strings must be delimited by apostrophes, with included apostrophes doubled.	The field is treated as a numeric field with regard to separators, and is skipped. The character/bit variable remains unchanged as if a null field were encountered.	L

E-9 1008 <

189 < E-10

Error Number	Message	Explanation	Standard Fixup	SYSTEM Action
40 through 49	END OF FILE ON CHANNEL n	Error 40+n is signalled if the end of file is reached while reading on channel n.	The remainder of the I/O statement is ignored. A further read on that channel will close and reopen the file at line 1 again.	T
50	ERROR IN HAL SOURCE		Continue	T
51	PROCESS NOT SCHEDULED DUE TO UNTIL/WHILE: program or task name	A SCHEDULE statement with the UNTIL <arith exp> or WHILE <event exp> was executed, and the time was already passed, or the event expression was false.	The SCHEDULE statement is ignored.	U
52	PRIORITY NOT UPDATED, PROCESS NOT ON QUEUE: program or task name	The specified task or program is not a currently scheduled process.	The update request is ignored.	U
53	PROCESS NOT TERMINATED: NOT ON QUEUE: program or task name	Same as above.	The terminate request is ignored.	U

ASD<E-11

Error Number	Message	Explanation	Standard Fixup	SYSTEM Action
54	PROCESS NOT CANCELLED, NOT ON QUEUE: program or task name	Same as above.	The cancel request is ignored.	U
55	PROCESS NOT SCHEDULED - ALREADY ACTIVE: program or task name	The specified program or task is currently an active process	The SCHEDULE statement is ignored.	U
56	PROCESS DEADLOCK	All remaining processes are in a wait state for some condition other than time.	None (IGNORE or GOTO action not allowed)	T
57	"REPEAT EVERY" program or task OVERLAP: name	The process was scheduled with the REPEAT EVERY option, and the current cycle lasted longer than the EVERY value.	(GOTO ACTION not allowed) The cycle which was supposed to start is skipped. The timing stays "in phase".	L
58	PROCESS NOT TERMINATED: NOT A DEPENDENT: program or task name	The specified program or task is not dependent on the running process.	The terminate request is ignored.	L

E-12

191 <

Error Number	Message	Explanation	Standard Fixup	SYSTEM Action
59	ARCCOSH function has argument < 1		Return 0	U
60	ARCTANH function has ARG ≥ 1		Return 0	U
61	ARCCOTH function has ARG ≤ 1		Return 0	U
62	BIT@OCT - INVALID character		Return 0	L
63	BIT@HEX - INVALID character		Return 0	L
64	No Space in File	Attempt to add a new block to Type II or III file failed.	The I/O operation is ignored.	T

E-13 1984

Error Num.	Message	Explanation	Standard Fixup	System Action
65	Block Length Mismatch	Block in memory and block on file indicated by FILE stmt have different lengths	See Sec.7	U
66	Block Number out of range	Block number specified in FILE stmt is out of legal range for partitioned file	The I/O operation is ignored	T
- - -				
70 thru 79	BLOCK bbbbbbbb NOT FOUND	Error 70+n is signalled if a Type II or III file input request indicates a legal but non-existent block.	The I/O operation is ignored	L

APPENDIX F.

USEF ABEND CODES

User Abend Codes During Execution

- 0 Missing DD card for message channel. The HAL/S channel specified for writing error and trace messages had no assigned DD card. (This may also appear as SYSTEM ABEND CODE 0).
- 1 HAL/S error with SYSTEM=T. A run time error causing abnormal termination occurred. A specific message preceding the ABEND information explains the error.
- 2 HAL/S error with SYSTEM=L and error count exceeded. A run time error occurred and the specified maximum error count for that error was thereby exceeded.
- 3 Invalid error recursion. An error condition arose while processing a run time error.
- 4 unused
- 5 unused
- 6 I/O mode conflict. Input asked for when I/O processor expected output or vice versa.
- 7 Illegal I/O channel or mode. I/O asked for on illegal channel, or I/O mode was illegal.
- 9 Program interrupt in non-HAL/S environment. Registers 12, 15 not set as expected on program interrupt.
- 10 Program interrupt from convert-to-binary (CVB) instruction.
- 11 Illegal HAL/S error number.
- 12 Too many events in event expression (>6)
- 13 Invalid event expression

User Abend Codes During Compilation

100	Unable to open one of the files: PROGRAM, SYSIN, or SYSPRINT
200	Unexpected end of file while reading in the XPL program
300	Synad error while reading in the XPL program
400	XPL program won't fit in the amount of memory available
500	Invalid service code from the XPL program
600	Printed-page limit exceeded
700	Linked programs specified different size common areas
800	Synad error on output file
900	Invalid output file specified
1000	Synad error on input file
1100	Linking process overlaid common string area
1200	End of file error on input file
1300	Impossible to move the common strings up during linking
1400	Invalid input file specified
1500	Unknown request by 'MONITOR' func
1600	Unknown do in 'MONITOR' request
1700	Directory error on PDS
1800	Synad error on output PDS file
1900	Invalid member name specified
2000	Synad error on direct access file
2100	Attempt to read from an input PDS without issuing the "FIND" MONITOR request first
2200	End of file error on direct access file
2300	Invalid member to be found

2400 Synad error on PDS input file
2500 File blocking specification error
3000 MON#9/10 error or misaligned #5
4000 XPL program called exit to force an abend (and a
 possible core dump)

APPENDIX G

List of HAL/S-360 runtime library routines.

These names should not be used to name any user-written, NONHAL routines.

```
ARCCOS ALIAS(ARCSIN)
ARCSINH ALIAS(ARCCOSH)
ARCTAN
ARCTANH ALIAS(ARCCOTH)
BAKTRACE
BIN ALIAS(CTOB)
BOUT ALIAS(BTOC)
CANCEL ALIAS(CANCELT)
CINDEX
CINP
CLJUSTV
CLOKTIME
CLOSEHAL
COLUMN ALIAS(TAB)
COUTP
CPAS
CPASP
CRJUSTV
CSHAPQ
CSLD ALIAS(CPSLD,CPSLDP,CPSST,CPSSTP,CSLDP,
          CSST,CSSTP)
CTOX ALIAS(CTOO)
CTRIMV
DARCCOS ALIAS(DARCSIN)
DARCSINH ALIAS(DARCCOSH)
DARCTAN
DARCTANH ALIAS(DARCCOTH)
DATE
DEXP
DISPATCH ALIAS(DISPACHS,DISPACHT,DISPACHW)
DLOG
DSIN ALIAS(DCOS)
DSINH ALIAS(DCOSH)
DSL D ALIAS(DSST)
DSQRT
DTAN
DTANH
DTOTHEE
EIN ALIAS(CTOD,CTOE,DIN)
EOUT ALIAS(DOUT,DTOC,ETOC)
ERRORMON
ERRORSUM
ERRTAB
ETOTHEE
EVENTENQ ALIAS(EVENTPRO)
EXCLUDE ALIAS(ALLOW)
```

EXECTRCE
 EXP
 FINDFIX ALIAS(GETADDR,GETSIZE)
 GETPTR
 HALDUMP ALIAS(HSLOCATE)
 HALSIM
 HALSTART
 HALSYS
 HDREAD ALIAS(HDOPEN,HDWRITE)
 HSDUMP
 IIN ALIAS(CTOI,CTOK,HIN)
 INPUT ALIAS(CIN,SKIPIN)
 IOINIT ALIAS(MSGIOINT)
 IOUT ALIAS(ITOC)
 ITOTHEI ALIAS(DTOTHEI,ETOTHEI)
 LINE
 LOG
 MM6DN
 MM6D3
 MM6SN
 MM6S3
 MOMSTACK
 MV6DN
 MV6SN
 M1DNP
 M1DSNP
 M1SDNP
 M1SNP
 M11DN
 M11SN
 M12DN
 M12D3 ALIAS(M14D3)
 M12SN
 M12S3 ALIAS(M14S3)
 M13DN
 M13D3
 M13SN
 M13S3
 M14DN
 M14SN
 M15DN
 M15SN
 M16DNP
 M16SNP
 M17DN
 M17SN
 M20DNP
 M20SNP
 M21DNP
 M21SNP
 NDX2PTR
 OUTPUT ALIAS(COUT,FLUSH,HALPRINT,SKIPOUT)

PAGE
PAGER ALIAS(FILETAB,NUMOFFLS,NUMOFPGS,PAGETAB)
PROGINT
QSHAPQ
RANDOM ALIAS(GETSEED,RANDOMG,SETSEED)
READBUFF ALIAS(CMPLBIT)
READPAGE ALIAS(PDSFILE)
SCHEDULE
SDLDDUMMY ALIAS(SDATRAP,SDETRAP,SDFTRAP,SDINIT,SDNTRAP,
SDSTRAP,SDTTRAP,SDWTRAP)
SDLSTACK
SET ALIAS(RESET)
SIGNAL
SIN ALIAS(COS)
~~SINH~~ ALIAS(COSH)
SKIP
SQRT
STMTRACE
SVBTOC
SVDTOC
SVETOC
SVITOC
SVPMSG
SVSTOP
SVTDEQ
SVTENQ
SVTIME
SYMBFILL
TAN
TANH
TENSTBL
TERMIN
TERMINT
TERMPCB
TIMENQ ALIAS(TIMECANC,TIMEINT)
UPPRIO ALIAS(UPPRIOT)
VM6DN
VM6SN
VO6DN ALIAS(VO6D3)
VO6SN ALIAS(VO6S3)
VV6DN
VV6SN
V16DNP
V16SNP
V9D3 ALIAS(V10DN,V10D3,V9DN)
V9S3 ALIAS(V10SN,V10S3,V9SN)
WAIT ALIAS(WAITUNTL)
WAITDEP
WAITFOR
WHERE ALIAS(WHERES,WHERESP)
XTOC ALIAS(KTOC,OTOC)

APPENDIX H.

Compiler Directives

The following compiler directives have been defined for the HAL/S-360 compiler.

- a) The **DEVICE** directive has the form:

```
D DEVICE CHANNEL=n <option>
```

This directive sets the mode of the specified channel (referred to via the CHANNELn DD card) to the mode indicated by the <option>. The <option> may be "PAGED", "UNPAGED", or null (in which case UNPAGED is assumed).

- b) The **INCLUDE** directive has the form:

```
D INCLUDE <name> <option>
```

This directive names a member of an include library as defined in section 6.2.7. The <option> may be "NOLIST" or null. The "NOLIST" option indicates that the included text is not to be listed.

- c) The **PROGRAM** directive has the form:

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

This directive provides a Program Identification Name to be used by the compiler to determine access rights to controlled resources as described in Section 6.2.8.

APPENDIX I

The HALLINK Program

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 LINKED step in the program generation process. This program is known as HALLINK.

Temporary work areas and general register 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). Persons using the standard JCL procedures listed in Appendix C need not be concerned with the existence of the HALLINK step.

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.

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 XXXXXXXX(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.

The following topics describe the various input options and output return codes produced by HALLINK.

HALLINK Output

HALLINK produces a series of object decks and directives to the Link Editor. They are described in the sequence that they are generated. User-control of HALLINK action is described later in an options list.

- 1.) INCLUDE <SYSLIB> (HALSTART)

<SYSLIB> is the name of auto-call library and defaults to SYSLIB.

This card may be suppressed. See option list

description to follow.

2) HALMAP CSECT

The content is detailed in a later paragraph. This object deck may be suppressed. See the options list.

3) Stack object decks for each PROGRAM and TASK.

4) INCLUDE <TEMPLOAD> (<MEMBER>)

<TEMPLOAD> defaults to TEMPLOAD.
<MEMBER> defaults to TEMPNAME.

5) ENTRY HALSTART

6) NAME <MEMBER>(R)

Produced only if <MEMBER> was not TEMPNAME.

HALLINK Option

Parameters may be passed to HALLINK. The JCL for this is PARM.LKED = 'link parms/HALLINK parms'.

The slash is optional if no HALLINK parameters are passed.

HALLINK parameters are coded as numbers, 0-9.

Code	Significance
0	Suppress printing of call tree and recursion diagnostics.
1	Suppress INCLUDE <SYSLIB> (HALSTART)
2	Suppress HALMAP CSECT. If a HALMAP CSECT exists in the input load module, i.e. a module created by HALLINK is reprocessed, this option is ignored.
3	Recursion ignored. Recursive PROGRAMS and TASKS receive stack size of 32760 (plus interrupt time).
4	Pass Link Editor parameters to second Link Editor only.

Attempt to open DCB with DDNAME LINKLIB. If successful, the Link Editor and HALLKED will be retrieved from this library or system library. STEPLIB will not be used.

6-9

Reserved.

HALMAP CSECT

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

- 1) Number of Process Control Blocks (PCBs) required by the Realtime 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	# of entries in pointer table
2	2	# 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.

HALLINK Return Codes

Return Codes

0,4,8,12,16	As defined by Link Editor.
1,5,9,13,17	Return code of n corresponds to Link Editor return code of (n-1) and recursive calls detected in load module and HALLINK option 3 was specified.
100	Recursive calls and option 3 not specified.
104	Insufficient space for tables. Rerun in larger partition. Current implementation requests 32K, 16K, 8K, and 4k, in that order, stopping when a request is satisfied.
108	Unable to open STACKOBJ or TEMPLOAD. If SYSPRINT cannot be opened, HALLINK option 0 assumed.
124	User specified a member name on TEMPLOAD card. TEMPLOAD must not have a member name specified.
128	I/O error reading TEMPLOAD directory.

Appendix J

Block Location and Search Algorithms for Type II and III

The location of a block in a Type II or III is determined by the following algorithm:

- 1) A starting location for the search is determined by the formula $B \text{ MOD } \text{DCBREL}$, where B is the block number and DCBREL is the number of blocks in a Type II file with OPTCD=R or the number of tracks in a Type II or Type III without OPTCD=R (the default). For example, if B=10 and a Type III file has 7 tracks, then the starting location is $10 \text{ MOD } 7$, or 3. This number is interpreted as the relative block or track in the file, again depending on OPTCD. In this case the starting location is relative track 3 of a seven track file, or the fourth track (the first track is relative track 0).
- 2) If OPTCD=E is specified, the search starts at the beginning of the track determined above and continues for as many tracks (or blocks) as the number specified in the LIMCT parameter. The search is for available space for a new block or for an existing block. If OPTCD=E is not specified, the search stops at the end of the track as determined in 1). If OPTCD=E is specified but LIMCT is not, then LIMCT defaults to the maximum, causing the entire file to be searched. (If the end of the file is reached, the search continues from the beginning of the file.)
- 3) If the search fails, the result is either ERROR 64 - NO SPACE IN FILE, or ERROR 70-79 - BLOCK bbbbbbbbbb NOT FOUND, depending on whether an output or an input file statement was being executed.

Conclusions:

- 1) If the range of B is uniformly sparse (say B=100, 200, etc) then the uniform increment (in this case 100) should be prime with respect to DCBREL.
- 2) Available space in an existing file may be increased without increasing the size of the file by utilizing OPTCD=E and/or an increased LIMCT.

205<

J-1