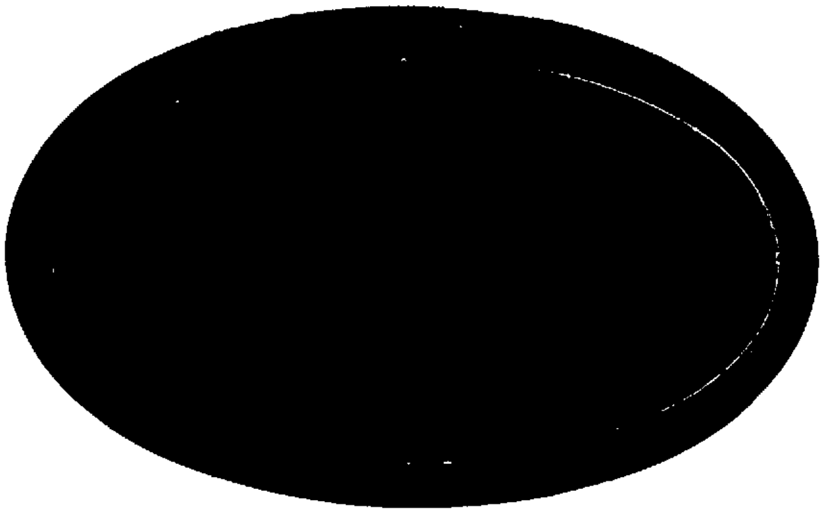
NASA CR-
140278



(NASA-CR-140278) HAL/S-360 COMPILER TEST ACTIVITY REPORT (Intermetrics, Inc.) 68 p CSCL 09B

N74-33675

Unclas
G3/08 48785

INTERMETRICS

HAL/S-360 Compiler

Test Activity Report

IR #86-1

3 July 1974

Prepared by: Carl T. Helmers

Approved by: Fred H. Martin
Dr. Fred H. Martin

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

Preface

This report describes the testing activities which have been performed as a part of the HAL/S-360 compiler development project.

There have been eight HAL/S-360 compiler releases, designated 360-1 through 360-8. 360-1 and 360-2 were early developmental in-house versions of the compiler. They were used for design purposes, and performance was verified largely by visual inspection of produced code. No real execution was planned*.

Releases 360-3 through 360-8 were compilers with successively greater capabilities. These compilers were issued to the Shuttle programming community (viz. Rockwell, Draper, IBM/Houston, Honeywell). Discrepancy reporting began with 360-3**. Formal testing activities, according to plan, have been conducted on all releases from 360-3 forward. Plans and results have been recorded by Intermetrics Shuttle Information Exchange (SIE) Memos. This current publication summarizes all of the 360 test activities and is meant to serve as a final test report.

* Results were discribed in Intermetrics reports issued on 13 April 1973 and 15 June 1973.

** This release was also described by Intermetrics report issued on 28 August 1973.

Table of Contents

1.0 . INTRODUCTION: LEVELS OF TESTING

The testing of HAL/S can be undertaken at several different levels. At the highest level of abstraction, the usefulness of HAL/S as a programming tool can be tested by using it in typical applications programming cases. Such testing has been and continues to be an important source of changes and modifications to the HAL/S Language Specification itself, as well as serving as a compiler-verification tool (see Section 3.1).

At a lower level, given the current language definition, the functional characteristics of the compiler system can be tested. Here the object is to verify that the compiler produces object code which is a faithful translation of the original HAL/S Program source. This functional testing of the compiler and its generated code is performed by a judicious selection of significant test cases which are compiled and then executed. Selection criteria for tests are based on many inputs, ranging from knowledge of the compiler's internal structure to the known "good" results of mathematical built-in functions.

At the lowest level there is the machine oriented testing of a particular compiler's implementation - such as the HAL/S-360 implementation on the IBM 360 or the HAL/S-FC implementation for the AP-101 flight computer. Here the emphasis is placed on the details of the resulting object code, e.g. are internal register assignments correct? is optimization working properly? etc. In a great many instances, problems revealed at the functional level provide indicators of significant areas to test at this level.

All three levels of testing are employed in verifying the HAL/S-360 compiler.

2.0 INITIAL TEST PLAN

The formal program of testing the HAL/S Compiler for the System 360 began just prior to the first operational release of the HAL/S compiler, Release 360-3 (August 1973). Subsequent evolution of the language itself, as well as the continued work on testing, have together resulted in modifications and improvements of the test plan.

This section contains a description of the HAL/S-360 test plan as originally formulated. Section 3, which follows, contains information on additional features incorporated in the test plan during the course of compiler development.

2.1 Test Philosophy and Test Procedure

If it is assumed that the definition of a working compiler is one which produces correct object code for every possible legal input, then clearly, the primary criterion for a compiler test plan is that it examine the code produced for every possible source statement. However, to take this requirement literally would produce a test program with a hopelessly large number of statements; the number of syntactic forms in the language factorial, might be a good first approximation. If this were necessary, it would not be practical to test the compiler at all. A more efficient method of testing than this "black box" approach must be utilized.

The saving feature is that of the huge number of possible HAL/S statements, most are different only to the programmer and not to the compiler. For instance, the "black box" or outside approach dictates that the addition operator, '+', must be tested with all possible combinations of positive and negative operands. This is really not necessary, as examination of the compiler will show that the negative operands, once converted to an internal form, are treated exactly the same as positive operands. In fact, the sign of an operand is not usually known by the compiler. Thus, an "inside" approach to testing may be developed. It is necessary only to test the several cases of this conversion to internal machine representation, and the test of the addition operator is reduced by a factor of four, or more.

This is not an isolated case; in fact, it is the rule rather than the exception. Testing based on a detailed knowledge of the compiler's inner workings eliminates the gross redundancy of the outside, or user, approach. Thus, the test philosophy becomes one of exhaustively testing all meaningfully different inputs to the compiler, where a "meaningful" difference is defined as one that is preserved through the initial passes of the compiler, and causes a different path to be taken in generating object code. This approach may require more time on the design of test cases, but it reduces these cases to a tractable number and further, makes it possible to examine the output code for correctness, in addition to comparing the results of the execution of that code against a predetermined answer.

The general problem of testing the HAL/S-360 compiler can be separated into several categories. The categories are chosen through an analysis of the structure of the compiler, paralleling its logically disjoint sections. Four major headings are identified (see Table 1):

a. the hand-coded portions of the first phase of the compiler which are listed under PASS 1;

b. those language features listed under Discrete which are more basic and self-contained;

c. those under General which have wider implications and which will require more special testing; and

d. System Support Features which cotains those categories which facilitate communication with the environment, and simpligy the writing of programs.

Certain language features, such as declarations, are not listed, but are implicitly tested by their use in the test routines for all categories. These implicit tests, together with the categories listed in Table 1, form a comprehensive set of language elements which must be tested. Based on this organization, the following procedure will be followed to produce all of the necessary tests.

a. Select a category.

b. Determine sub-categories.

c. Classify sub-categories into enumerable vs. special.

d. Produce test matrices for each enumerable sub-category.

e. Generate tests from these matrices.

f. Generate special case tests.

Within each category, sub-categories will be produced by a process similar to that which generated the category itself. These sub-categories tend to split along the lines of statement types, rather than along the lines of language features, as the initial categories did. This type of classification scheme, while somewhat redundant, simplifies the testing process, and increases the probability that no language construct will remain untested.

Next, the sub-categories will be divided into two groups: enumerable and special. The first group will test those parts of the compiler through which several paths may be taken, depending upon the context in which they are used. For instance, the exponentiation routine's behavior varies considerably depending on the arguments it is called with. An example of a "special" sub-category is the test of the multiple assignment processor. This routine always functions in the same way, sequencing through the receiving fields on the left of the assignment statement, evaluating each one, and calling some other routine to do the assignment. The routine that actually does the assignment is tested as one of the "enumerable" sub-categories, as it must do a conversion if the data types involved are not identical.

The tests required for one of these "enumerable" sub-categories are represented by a matrix. The matrices for the Integer-Scalar sub-categories are listed in Appendix A. Refer to the matrix A3: This matrix is accompanied by the expression <column element>**<row element>. The X in the matrix which has been circled indicates that one of the test cases must be a double precision scalar raised to the power of a double precision scalar. In general, an X in the matrix indicates that the label above the X (signified by <column element>) must be raised to the power of the label to the left of it (signified by <row element>). In one of these matrices, an X may be omitted if and only if there is already a case which will take the same path through compilation.

TABLE 1.

Testing Categories

Pass 1	Discrete
error checks	Integer, Scalar
output writer	Vector, Matrix
scanner	Bit
	Character
	Structure
	Flow Control
	Conversions
	I/O

System Support Features	General
macros	library
includes	built-ins
PDS manipulation	Arrayness
files	structure copyness
compool	relational expressions
comsubs	arrayed relations
locks	procedure link and parameter pass
real time	addressability
	register usage
	update, task blocks

Finally, test cases must be generated for the "special" sub-categories. Here the matrix formalism is not applicable. These special cases must be made to trigger specific pathways through the compiler, and their design requires a detailed knowledge of the decision points along the compilation process. Wherever a decision point occurs in the compiler, a test case must be prepared to follow each branch. The results of these special test cases must be checked by an examination of the object code rather than by checking the results of execution, but this is advantageous in that the number of test cases may be reduced by combining several tests into one statement.

2.2 EXAMPLE: INTEGER-SCALAR

To illustrate the testing strategy, one of the major categories from Table 1 will be explained in detail down to actual test cases. The example that has been chosen is Integer-Scalar. The primary reason for selecting this category for first consideration is that it will be so widely used in HAL/S programs; an error here is likely to cause errors in the majority of compiled programs. In addition, an early test of this category may show up widespread errors in the compiler, since the testing of Integer-Scalar operations must involve the use of many other language features. Finally, the Integer-Scalar category is large enough to illustrate many of the problems encountered in the test procedure and point to their solutions.

To simplify the task, the Integer-Scalar category was divided into several sub-categories along the lines of statement type (see Table 2). These sub-categories overlap to cover all uses of integers that are not more appropriately covered under one of the other categories listed under General in Table 1.

Appendix A contains appropriate matrices for the enumerable sub-categories of Integer-Scalar. Some matrices indicate many possibilities where as in others only a smaller number of cases

TABLE 2.

Categories Tested Under Integer, Scalar*

Category	Matrix	Comments
1. Assignments	X	All tested due to implicit conversions.
2. Initial Values	X	
3. Comparisons	X	
4. Binary Mathematical Operators	X	Not including ** (exponentiation)
5. Exponentiation	X	Separate category due to required internal handling by compiler.
6. Unary Operators	X	+ and = prefixes.
7. Multiple Assignments		Test of ordering of assignment included.
8. Complex Expressions		Tests of precedence; elementary function, returns, definitions, and element selection implicitly tested.
9. Complex Comparisons		Tests of precedence, combinations with BOOLEAN variables included.

* Note: Certain declares and initialization, (simple I/O and flow control, comment ability, and no argument procedures call) implicitly tested.

will need to be tested. Those cases which are eliminated are done so due to certain properties of the compiler, primarily commutivity. For example, under the operators '+', '-', and '/', the tests form a triangular matrix due to the "bubbling up" (to higher precision) property of the compiler; a scalar of double (SD) precision divided by a scalar of single precision (SS) is essentially the same as SS/SD since in both cases the SS is converted to an SD and then the division is performed. On the other hand, all possible assignments are tested due to implicit conversions. Each matrix will have notes explaining deletions of tests.

2.3 Test Result Criteria

Once the test programs have been run, the results must be verified. This can be done on two levels. The higher level is that of checking produced values vs. expected values. (This level is, in fact, the more important of the two in terms of correctness.) For two reasons the checkout of the HAL compiler will involve both levels. The first is that of completeness through retention of an outside check. The second is that in order to remain compatible with our inside approach, the object deck produced from a HAL program should be examined. Therefore, in a HAL statement such as A=5; where A is a scalar or single precision, one would check that A is indeed equal to 5 and that the literal has been converted to a single precision scalar.

It may be noted that the majority of test cases are simple rather than compound statements and expressions. The motivation behind this is that intermediate results in the evaluation of expressions are usually kept in registers. Thus, they all have the same format, which means that there is no need to mix data types in the complicated expressions, after each combination has been paired and checked in the simple expressions. More complex expressions in the Integer-Scalar category examine such things as order of evaluation of sub-expressions and the correct use of temporaries. Evaluation of these tests will be done by examination of the object code produced rather than by checking the results. Thus, as expressions get more complex, we move from examination of the results of many cases, to a check of the generality of the code produced for a few cases.

3.0 TEST PLAN ADAPTATION

The original test plan, as described in Section 2 above, has been adapted and improved during the course of the HAL/S-360 testing activities. At the time of the final compiler version, Release 360-8, the number of test programs, for instance, had grown to over 100 from an original count of 40 programs. Language design features have also changed during the course of compiler development, and with each release the test cases in the library of test programs have been reviewed and modified to reflect such additions. This section describes the extensions to the original test plan which have been incorporated as of the date of this report.

3.1 Design Tests Incorporated into Formal Testing

In carrying out the HAL/S-360 test plan, Intermetrics took advantage of its knowledge of the internal design and implementation of the compiler. Special tests in order to verify specific constructs and/or algorithms were added to the set of Formal Tests. Some examples are described in this paragraph:

1. Since the compiler's replace machanism sees the replace text as an uninterpreted character string, significant tests involve replace texts of varying length, nested replaces and parameterized replaces.

2. Referencing elements of aggregate data types must be tested in cases where the total size of the data in question is greater than 4096 bytes (on the 360). This is due to the fact that in some cases a subscript expression can be incorporated into the displacement field (12 bits) of a machine instruction.

   For a similar reason, HAL statements which increment an integer variable by more than 4096 must be tested:

   > "I = I + 5;" can be done by a LA instruction (leaving the result in a register) whereas "I = I + 5000;" cannot.

PRECEDING PAGE BLANK NOT FILMED

3. Since the compiler stores multidimensional arrays (and matrices) in row-major order, tests of the form "A$(*,3) = ..." are necessary to verify assignments into array partitions which are not contiguous.

4. The order of evaluation of parts of a statement must be tested in several cases other than expression context. An example is:

   "A$I,I = I+A$3".

5. Since the compiler maintains a separate stack segment for each code block, assignment statements across task, procedure, and program boundaries must be used to verify correct accessing of data at various nest levels.

6. Some computations are done at compile time. Thus, a complete test of exponentiation must include both "I = 3**12;" and "J = 3; I = J**12;". Tests of precedence and certain built-in functions must also be made in this context.

7. Consider the following example:

```
                    T:  program;
                        Declare integer initial (5), A,C;
                        Declare B bit(16) initial (Hex'8005');
/*Test 1*/              A = abs(integer(b));
/*Test 2*/              C = abs(A);
                        Write(6) A,C;
                    Close;
```

   One might expect that Test 2 was the same as Test 1 in the above example, since both statements take the absolute value of a single (default) precision integer. This, however, is not the case, due to the implementation of the abs function and the pecularities of the 360 STH and LPR instructions.

3.2 Total Testing of the HAL/S-360 Compiler - Use Testing

The programs which represent the implementation of the HAL/S-360 test plan are only a portion of the test effort that is in progress. All tests can be included in one of the following four classes:

1. Formal Testing: The tests which are generated according to a test plan.

2. Developmental Testing: The day to day test programs which are used to verify parts of the compiler as they are written.

3. Visual Verification: The examination of the code as it is written to find special cases which have not been accounted for.

4. Compiler Use: Programs which are written in HAL/S to be used in the course of other work.

With respect to the last class, it should be noted that certain types of errors are not easily caught by a systematic test plan. These errors can be found only when a program is presented with one particular set of inputs. Therefore, it is useful to spot check the compiler randomly by subjecting it to a number of application and utility programs. Such programs have been designed and are listed under Category 6 in the tables of Appendix C.

The uses of these programs range from calculations to debugging, to the generation of other test programs. Many of them were run on a daily basis.

This type of program has been responsible for the detection of several errors during the course of the HAL/S development effort. For instance, in release 3 of the compiler, an application program called QUEUES detected one of several discrepancies in compiler operation when a particular combination of name choice and block nesting was compiled. In release 6 of the compiler, the programs KIP and FDA were responsible for the detection of an error triggered by the specific program swapping techniques employed in the real time aspects of these applications. These are not the only examples of such error detection. The technique will continue to prove valuable in the future releases of HAL/S compilers for the flight computer and the 360, if necessary.

3.2.1 PMF and FDI Simulators

Two major HAL/S programming efforts were developed at Intermetrics in support of Rockwell International, viz. a Performance Monitoring Function simulator and a Failure Detection Indication simulator. The HAL/S-360 development has benefitted from this in-house use of the language and compiler. The programs are distinctly different and exercise almost all aspects of HAL/S from mathematics and structures to systems use and real time. They were developed in parallel with HAL/S-360 and were run successfully using several releases of the compiler. Several HAL/S discrepancies were discovered using these programs on developmental versions of the compiler.

3.2.1.1 PMF. The Performance Monitoring Function simulation currently comprises ten external procedures, two Compools, and four programs. The external procedures are display handling primitives which are called by the following programs:

1) Operational Programs

   a) FDA. Executes all Fault Detection and Annunciation functions. Provides data base for SMM, SCM display functions.

   b) KIP. (Keyboard Input Processor) Control center for display processing. Responds to KBD inputs, and maintains control of the CRT level usage by each of the four DEU's.

2) Simulation Support Programs

   a) CREWMAN. Prepares simulated crew KBD inputs, and passes them to KIP. Also processes requests for CRT page print-outs, and provides general simulation control.

   b) EXEC. This is a small routine that does the initial scheduling of all the component programs, to get the simulation going.

The combined programs and procedures contain approximately 1800 HAL/S statements and the Compool data occupies approximately 10,000 360 words. The four programs contain eight task blocks and over twenty internal procedures and functions.

REPRODUCIBILITY OF THE
ORIGINAL PAGE IS POOR

3.2.1.2 FDI. The FDI simulator is designed to evaluate sensor FDI algorithms under realistic Shuttle flight considerations. The following Program Layout taken from the compilation indicates the organization and scope of the simulation.

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

   1    STATEUPDATE: EXTERNAL PROCEDURE;
        SELECTSTATE: EXTERNAL PROCEDURE;
   2
        REALDATA: EXTERNAL COMPOOL;
   3    ORTHO: EXTERNAL FUNCTION;
        COMPDATA: EXTERNAL COMPOOL;
   4
        OUTPUTDATA: EXTERNAL PROCEDURE;
   5    VEHICLEDYNAMICS: EXTERNAL PROCEDURE;
        IMUMODEL: EXTERNAL PROCEDURE;
   6
        NAVSENSORS: EXTERNAL PROCEDURE;
   7    IMUFDI: EXTERNAL PROCEDURE;
        NAVFILTER: EXTERNAL PROCEDURE;
   8
        GUIDANCE: EXTERNAL PROCEDURE;
   9    SHUTTLE_FDISIM: PROGRAM;
            BASICDATA: PROCEDURE;
  10
            CASEDATA: PROCEDURE;
  11        REALWORLD: PROCEDURE;
            COMPUTER: PROCEDURE;
  12
            TIMESTEP: PROCEDURE;
```

The program contains over 1400 HAL/S statements and also includes 40 internal procedures and functions.

3.2.2 Boost Trajectory Generator

This program simulates a boost-into-orbit trajectory. It is typical of guidance and control programming using all the mathematical features (integer, scalar, vector, matrix) available in HAL/S as well as arrays for formulating and manipulating tables of data.

3.2.3 Other Testing

Testing of the HAL/S compiler releases was also conducted at three other contractor installations:

Draper Laboratory
IBM/Houston
Rockwell/Downey

These contractors reported discrepancies (Discrepancy Reports) to NASA/JSC and Intermetrics. All reports were evaluated by Intermetrics and errors fixed in subsequent releases.

3.3 Mathematical Function Precision Testing

The incorporation of precision tests into the testing of HAL/S-360 mathematical function routines was begun with Release 8 of the HAL/S 360 compiler. The purpose of this testing is to verify the mathematical function algorithms as implemented, for internal consistency and for consistency with known values of the functions.

The initial approach to this form of testing was to analyze the algorithms used, looking for likely sources of error in the calculation and for input ranges where large errors can be expected. The purpose of this analysis was to find a set of worst case test points for each function which would then be compared with the exact mathematical values of the functions. For example, it was expected that the errors would be larger near singularities. There was some success with the method as applied for example to the Sine routine, testing multiples of π for an exact zero value, thus analyzing the results of the finite precision of the constant π used in making the principal angle corrections. This type of testing is continuing with more results to be expected at a later date.

Following the initial testing of mathematical routines as outlined above, it was decided to check the accuracy of the approach --- were the largest relative errors at the predicted points based on analysis of the algorithms? To perform this consistency checking, it was decided to use the double precision versions of routines as the "standard" for comparing single precision results and calculating relative error as a function of the input argument range. Preliminary results indicate that for ranges of the argument which result in errors on the order of magnitude of the last digit, the error is not related to the "most likely" points determined by analysis. The source of this error is attributed to truncation as opposed to fundamental characteristics of the routine itself. For larger errors, the programs doing the testing were modified to isolate the regions of larger error and examine such regions with a finer granularity. Using this approach, it was found, for example, that the HAL/S single precision TANGENT function yields regions of arguments with errors three orders of magnitude larger than the largest IBM FORTRAN relative error in the equivalent routine, indicating a probable error in the HAL/S runtime routines. In order to extend this approach to double precision routines, it is necessary to obtain comparison standards for triple precision versions of the same functions. Work is now progressing on the definition of such extended precision algorithms for use in checking out the HAL/S double precision routines.

All the function range testing being applied on the 360 version of the compiler can and will be carried over to the flight computer compiler. This work can be accomplished due to the identical floating point data representations and the fact that identical algorithms (although differing in machine representation) are used for both computers.

4.0 TEST RESULTS SUMMARIES

The HAL/S-360 testing activities have been carried out for each release of the HAL/S compiler. This section summarizes the results of tests, by release.

4.1 Formal Testing of Release 360-3

The test plans, as described above, were first applied to Release 360-7. A set of test categories and sub-categories was generated for this purpose. Test programs were written and exercised on 360-3 covering all features expressed in the plan. In the tables of Appendix C, the categories (and sub-categories) and the corresponding test programs (in caps) are indicated. (Appendix C shows the current list of programs).

In all, forty (40) test programs were written for categories A, B, C, D. These programs contained approximately 4,000 individual logic and language feature tests for Release 360-3. The bulk of source text, object code, and test results printout precluded their publication for distribution, however, these volumes are on file at Intermetrics, i.e:

1. The original source program
2. The formatted compiler listing
3. The object code listing produced by the compiler
4. The printout of the test routine's execution

4.2 Formal Testing of Release 360-4

Release 360-4 was tested using a sub-set of the tests defined for 360-3 (see Appendix C) and a new additional set of tests. The tables and paragraphs below identify and summarize the purpose of these tests. In all, these programs represented in excess of 2000 individual logic and language feature tests.

PRECEDING PAGE BLANK NOT FILMED

4.2.1 Sub-Set of Test Plan

A. Discrete

1. Integer-Scalar

   HALTEST
   OPERTEST
   EXPONTEST

2. Vector-Matrix

   MATVECT_COMPARE
   MATVECT_PARTITIONING
   MATVECT_EXPRESSIONS

3. Bit

   BITTEST

4. Character

   CHARACTERTEST
   CHARTEST
   TEST2

6. Arrays

   ARRAY_ELEMENT_SELECT

7. Flow Control

   FLOWTEST

B. System Support Features

4. Compool & 5. Comsubs

   CSUB
   CPOOL
   PROG

•D. General

1. Library & 4. Procedures

EXPRESSIONS_AS_ARGUMENTS

4. Procedures

PARM
ECCHO

6. Conversion

VECFEON

F. Special Tests

SCRUNCH
ED

4.3.2 New Formal Tests for Release 360-4

1. CONFLICT_TEST examines part of the compiler's error detection facility. It is a test of the use of factored attributes which disagree with non-factored attributes in the same declaration: e.g. "Declare vector(3), I,J,K integer, L;". A test is also made of the compiler's conflict-detection algorithm; i.e. "Declare integer, A,B, C integer double, D;" must not be flagged as an error.

This category of testing (error detection) is made difficult by the fact that the compiler, having found one error, may announce further spurious errors as a result of incomplete error recovery. Further, the discovery of one error may cause the occurrence of further errors to go undetected.

2. LOG_SQRT is a test of two of the built-in mathematical functions. It produces a table of square roots and logorithms which may be checked against any standard reference. In addition, the inverses of the built-in functions are used within the program for self checking and to determine the loss of precision involved in the operations. Two other tests, <u>SIN</u> and <u>SINH</u> are essentially the same, except that they test the trigonometric functions.

3. REMEMBER_REGS tests the compiler's optimization of register usage. This program checks the following features:

a) That register contents are not assumed to remain intact across statement boundaries when a labeled statement or when flow control statements are encountered.

b) That when a statement has too many variables for the number of registers available, the variables in registers that must be re-used are properly stored into memory, and

c) That these variables which have been replaced are not assumed to remain in the registers in subsequent statements.

4. VECTOR_SHAPETEST was written specifically to insure that the discrepancies found in the vector shaping function at the time of Release 360-3 had been corrected. It tests nested shaping functions, and the use of a vector shaping function in the argument list to a procedure or function. The errors which were found in Release 360-3 were corrected.

5. SPLAT is a test of the use of the operator "#" to specify repetition in an initial or constant list in declarations. The program insures that nested repetition lists, such as "3#(14.5,2#(3,4),5)" are correctly interpreted, and that repeated lists with elements that require some conversion are properly handled. An example of this latter case is: "Declare I ARRAY (6) INTEGER INITIAL (2#(3,SQRT(8),4));".

6. LONG is a test of the EXIT and REPEAT statements used inside an iterative DO loop which has a length of more than 4096 object bytes. The program checks that addressability is maintained when an EXIT or REPEAT causes a branch to a section of code which cannot be addressed from the current base register.

7. The programs PROD, SUM, CEILING, FLOOR, MOD, and REMAINDER test the built-in functions of the same names. Several related errors were found in these library routines. For instance, CEILING (4.0) returned 5. This error was introduced in the single to double precision conversion which introduces non-zero low order bits, forcing the result in the next higher integer. These errors were corrected in Release 360-5.

4.3 Formal Testing of Release 360-5 and 360-6

All previously generated tests were run using releases 5 and 6. In addition, new test programs were written to augment the previous test plan. In all, by the release of 360-6, eighty-eight test programs had been written for categories A, B, C, and D. These test programs contain approximately 8,000 individual logic and language feature tests.

As a result of the tests above being conducted, nineteen compiler errors were discovered:

1. Pass two of the HAL/S compiler failed to generate proper code for shaping functions applied to bit strings of length 16 arguments.

2. One matrix exponentiation library routine was improperly coded, causing spurious errors and abends. This error was detected by MATRIX_EXPONENTIATION.

3. Assumed length passing in the library routine of CTOB caused erroneous runtime results. The error was discovered by BSHAPE.

4. The use of the CSHAPE test program determined that improper coding in the library routine called by the HAL/S language form $\text{CHARACTER}_{\text{@DEC}}$ caused results to be proper if and only if a factor of ten was calculated into the original argument.

5. The replace macro facility improperly interpreted macros nested within macros for the case in which a parameter to the outer macro was itself an argument of a nested macro. The error was detected by SUBBIT_ON_INTANDBIT.

6. Registers 0 and 1 were found not to be preserved in process swapping. Such preservation is necessary for the proper functioning of the code generated by pass two of the compiler because pass two assumes that these registers remain unchanged across statement boundaries.

7. Passing a bad argument to the library routine CTOI caused abends during runtime instead of the generation of the proper error message.

8. Character subscripting in an expression using the TO # subscript form caused overwriting of the calculated length value. Thus, erroneous results occured at run-time.

9. Structure input and output caused bad code to be generated for the case in which the structure had a multiple copy specification.

10. The length of a literal character string was incorrectly calculated in the compiler.

11. During automatic template generation, multiple replaces in compools caused replacement of the several subsequent macro statements by the first macro string, regardless of the form of the later statements.

12. Subscripts which select one copy of a multiple copy structure result in incorrect code: the base to which the calculated index is added is greater than the address of the first copy of the structure.

13. The default value of the "speed" parameter was set to five million rather than five hundred thousand as specified.

14. The INCLUDE function worked incorrectly when a block template being generated was being matched with a previously created template.

15. The MAX or MIN functions did not work with arguments with a * array size.

16. If two compool templates used in the same compilation unit possessed identically named identifiers, no error was detected.

17. There was an output writer problem in printing a qualified structure name in a subscript.

18. If the dot product of two vectors of unequal length was taken, the Phase I error message was garbled.

19. The multiplication of unsubscripted variables by argument less functions sometimes produced bad object code.

20. If a program contained an identifier T then occurrences of the superscript form of transpose caused spurious cross references to appear in the symbol table entry for T.

21. In some circumstances, use of the SIZE function with arguments of * array size produced bad object code.

All of these errors were corrected for Release 7 of the compiler.

4.4 Formal Testing of Release 360-7

The standard library (with additions) of formal tests were re-run on Release 7. Additional tests were generated, covering new HAL/S language features incorporated into Release 360-7. A total of 4 discrepancies were reported during development of 360-7, with additional testing after release yielding 3 more.

4.5 Formal Testing of Release 360-8

The standard library (with additions) of formal tests was run using the release 8 version of the compiler. Additional tests generated for release 8 were used to test out new features and to add to the repertoire of previous tests. The new test programs are:

Structures:

STRUCT_IO_COMP

STRUCT_ACCESS

STRUCT_ERR

STRUCT_PARM

Arrays:

ARRAY_PARTITION

ARRAY_PARM

Replace Macros:

REPL_TEST

REPL_ERR

System Language Features:

INLN_FUNCTIONS

TEMP_TEST

As of the time of the release, this testing had revealed one discrepancy in the operation of the compiler occurring when a structure function used in an assignment statement. The final list of formal test programs is included as Appendix C of this document.

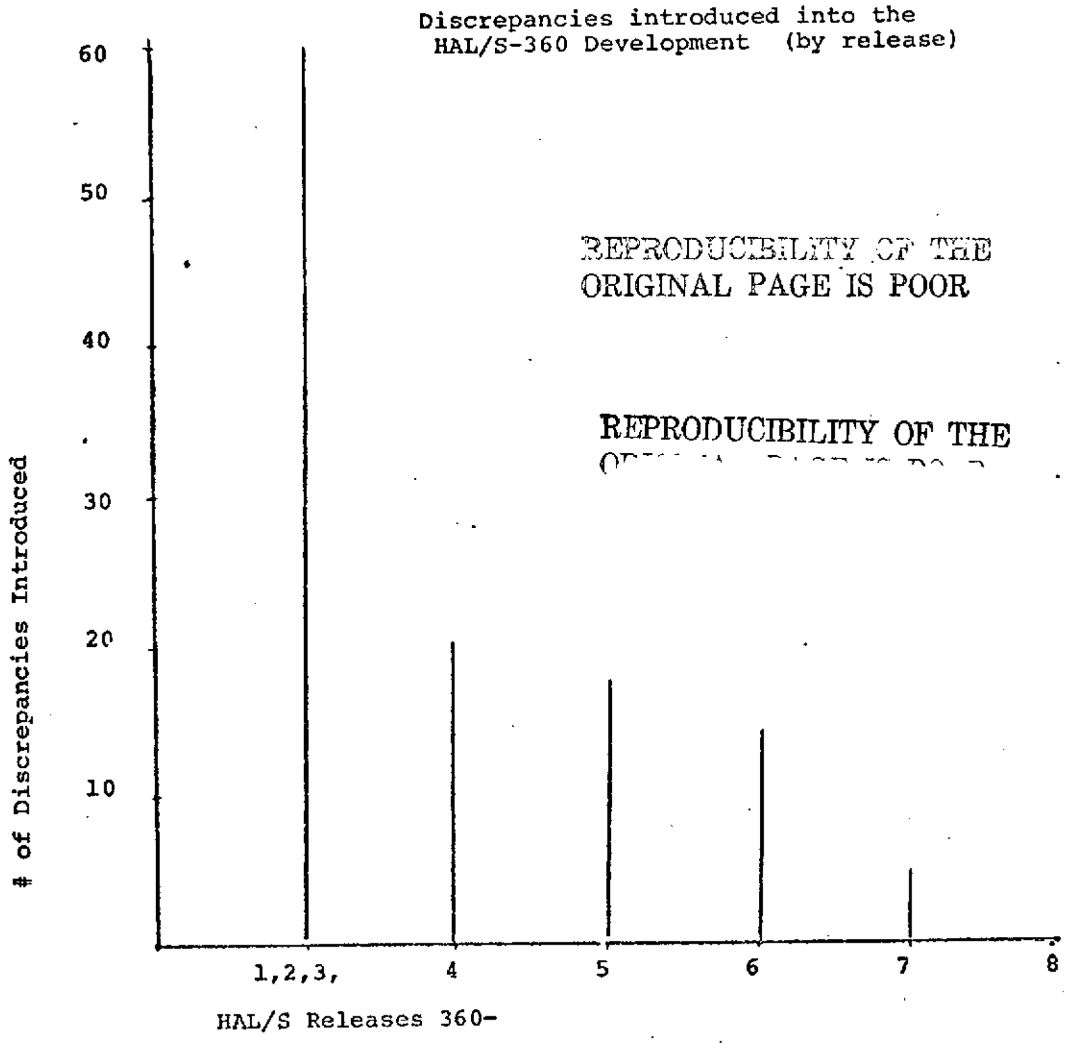
Also, the precision testing of mathematical function subroutines in the run-time library was begun with release 8 of the compiler. A later memo will document results of this test activity.

4.6 Discrepancy Reports

Appendix D lists all reported discrepancies as of June 30, 1974. These discrepancies were analyzed and entered onto the work sheets included in Appendix E. The figures below summarize this information. Figure 4-1 plots the number of discrepancies introduced into each released version of the compiler as a result of new code and/or the alternation of existing (good) code. The first three HAL/S-360 releases have been grouped together and are treated as the original release. The drop-off in introducing errors is marked with only two resulting from 360-7. The trend is indicative of the maturity of both the compiler itself and programmers at Intermetrics developing it. Changes can now be introduced with minimum risk in causing additional errors.

Figure 4-2 is interesting in that it attempts to convey a degree of difficulty in finding introduced errors. The latency value 0, 1, 2, etc., means that the error

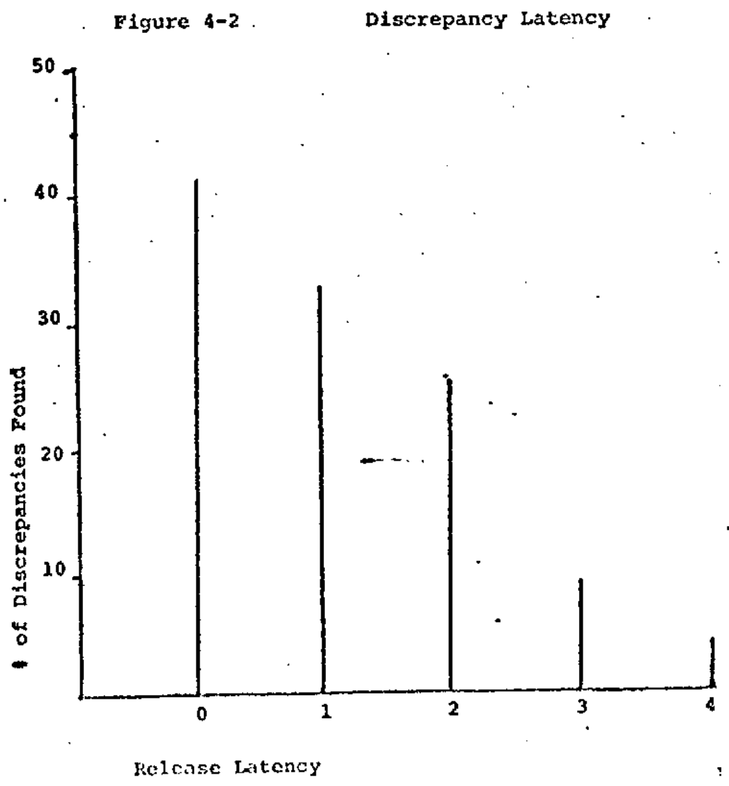
Figure 4-1



REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR

REPRODUCIBILITY OF THE

Figure 4-2

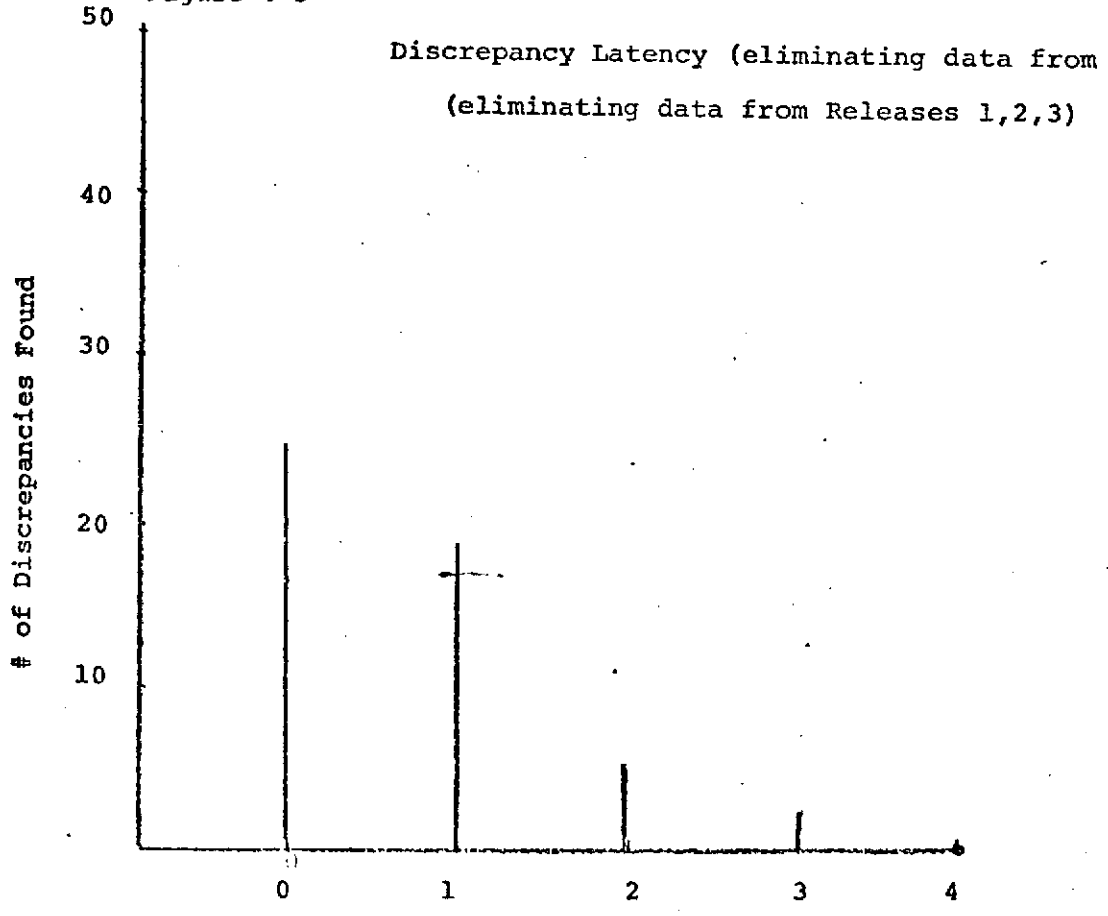


was introduced in one release and found in the same release(0), the next(1), or the next again(2), etc. Thus for example, 42 of the descrepancies were introduced and then discovered in the same HAL/S-360 release. The average latency value is 1.1 and indicates that for the development so far, it takes, on the average, one additional release with its attendent testing, use, verification, etc., to find an introduced error.

Figure 4-3 presents latency data again, this time with the first three HAL/S releases eliminated. Discounting this start-up "transient" it would appear that the total verification effort was some what better, in that errors were found, on the average, a little sooner. Of course this result might be expected as the test cases became more voluninous and the programmers gained more insight into the vulnerable areas of the compiler.

The full story on HAL/S-360 development is, at this writing, incomplete. Release 360-8 has just been release and data will continue to be gathered. At a subsequent date an addendum to this report will be issued in which it is hoped that the reliability of the compiler and the effectiveness of the total test and verification efforts may be judged.

Figure 4-3



APPENDIX A

Matrices of Tests

<row element> means the label of the row in which an 'X' appears.

<column element> means the label of he column in which an 'X' appears.

List of Tables

A.1 Assignments with Implicit Conversions

	Sc(s)	Sc(d)	I(s)	I(d)	Lit(scalar)	Lit (integer)
Sc(s)	X	X	X	X	X	X
Sc(d)	X	X	X	X	X	X
I(s)	X	X	X	X	X	X
I(d)	X	X	X	X	X	X
Lit(s)						
Lit(I)						

<column element> = <row element>

A.2 Operator Test

	Sc(s)	Sc(d)	I(s)	I(d)	Lit(scalar)	Lit (integer)
Sc(s)	X					
Sc(d)	X	X				
I(s)	X	X	X			
I(d)	X	X	X	X		
Lit(s)	X	X	X	X	X	
Lit(I)	X	X	X	X	X	X

<column element> <operator> <row element> where <operator>

:: = + | -|<times>|/

A.3 Exponentiation Test

	Sc(s)	Sc(d)	I(s)	I(d)	Lit(scalar)	Lit (integer)
S[redacted](s)	X	X	X	X	X	X
S[redacted](d)	X	X	X	X	X	X
[redacted]s)	X	X	X	X	X	X
[redacted]d)	X	X	X	X	X	X
[redacted]t(s)	X	X	X	X	X	X
[redacted]t(I)	X	X	X	X	X	X

<column element>**<row element>

An exhaustive test was necessary because the compiler makes a more elaborate treatment of exponentiation than the other operators:

<literal>**<literal> is reduced to <literal> at compile time.

Exponentiation is done with an in-line code substitution.

A.4 Unary Operator (-)

	Sc(s)	Sc(d)	I(s)	I(d)	Lit(scalar)	Lit (integer)
Sc(s)	X					[redacted]
Sc(d)	X	X	X			[redacted]
I(s)			X			[redacted]
I(d)				X		[redacted]
Lit(s)	X	X	X			[redacted]
Lit(I)		X	X	X		[redacted]

<column element> = - <row element>

The cases above include a few of the possible conversions. Conversions were tested thoroughly elsewhere; their inclusion here is just to test that the compiler recognizes the need for a conversion.

A.5 Comparisons

	Sc(s)	Sc(d)	I(s)	I(d)	Lit(scalar)	Lit (integer)
Sc(s)	X					
Sc(d)	X	X				
I(s)	X	X	X			
I(d)	X	X	X	X		
Lit(s)	X	X	X	X		
Lit(I)	X	X	X	X		

If <column element> = <row element> then

Once one operator has been tried against all possibilities, the others need only one test case. The code produced will be similar to the following:

CLC	dependent on data types
BC	M-dependent only on operator

Thus, a test of one operator will ensure that the correct comparison is done, and the test of the others is only to ensure that the correct mask is set up in the BC instruction.

A.6 Initial

	Sc(s)	Sc(d)	I(s)	I(d)	Lit(scalar)	Lit (integer
Sc(s)					X	X
Sc(d)					X	X
I(s)					X	X
I(d)					X	X
Lit(s)						
Lit(I)						

Declare <row element><attributes> initial (<column element>);

APPENDIX B: Summary of Categories (Initial Listing)

B.1 DISCRETE

INTEGER-SCALAR
VECTOR-MATRIX
BIT
CHARACTER
STRUCTURE
ARRAYS
FLOW CONTROL
I/O

B.2 SYSTEM SUPPORT FEATURES

MACROS (REPLACE)
INCLUDES
FILES
COMPOOLS
COMSUBS
REAL TIME
OPTIONS IN JCL
DEVICE DIRECTIVES

B.3 PASS 1

ERROR CHECKS
OUTPUT WRITER
SCANNER
DATA DECLARATIONS
SCOPING RULES

B.4 GENERAL

LIBRARY & BUILT-INS
ARRAY-STRUCTURE INTERACTION
RELATIONAL EXPRESSIONS
PROCEDURE LINK & PARAMETER PASS
ADDRESSABILITY & REGISTER USAGE
CONVERSIONS
SHAPING FUNCTIONS

B.5 NEW FEATURES

NAME Facility
INLINE FUNCTION BLOCKS
PERCENT MACROS (% MACROS)
STAND-ALONE DEBUGGING LANGUAGE

APPENDIX B: Detailed Breakdown of Categories

B.1 DISCRETE

B.1.1. Integer-Scalar

Assignments
Initial Values
Comparisons
Infix Mathematical Operators
Exponentiation
Prefix Operators
Multiple Assignments
Complex Expressions
Complex Comparisons

B.1.2. Vector-Matrix

Comparisons
Assignments
Initial Values
Conversions
Plus, Minus
Multiply (All Types), Divide
Exponentiation
Unary Operators
Element Selection
Partitioning
Complex Expressions
Complex Comparisons
Multiple Assignments
Built-In Functions

B.1.3. <u>Bit</u>

Assignments
Initial Values
Comparisons
Conversions
Concatenation
And, Or Operators
Not Operator
Element Selection
Complex Expressions
Complex Comparisons
Multiple Assignments
SUBBIT
Partitioning

B.1.4. <u>Character</u>

Assignments
Initial Values
Conversions
Comparisons
Concatenation
Element Selection
Complex Expressions
Complex Comparisons
Multiple Assignments
Built-In Functions
Partitioning

B.1.5. Structures

Qualified Name Reduction (assign into)
Qualified Name Reduction (assign from)
Template Matching
Initialization
Comparisons
As Parameter or Return
Multiple Copies
Dense & Aligned
Copy Selection
Multiple Assignments
Minor Structures
Qualified vs. Unqualified
Complex Subscripting
Assignments

B.1.6 Arrays

Element Selection
Partitioning
Initialization
Mathematical Infix Operators
Comparisons
Parameter Passing
Dense vs. Aligned
Assignments
Multiple Assignments
Prefix Operators
Built-In Functions

B.1.7. Flow Control

IF <bit> THEN
IF <arrayed bit exp> THEN...
IF <bit> THEN ELSE
IF <bit or relational expression> THEN
DO...END GROUP
Discrete DO FOR
Iterative DO FOR
DO UNTIL <bit>
UNTIL <bit or relational expression>
DO WHILE (<bit>)
DO WHILE (<expression>) .
DO Case
DO Case...Else
GOTO
EXIT
REPEAT
DO FOR...BY...TO
Discrete DO FOR WHILE
Iterative DO FOR WHILE
Discrete DO FOR UNTIL
Iterative DO FOR UNTIL
DO Case <scalar>
Nested Conditionals
Nested DO's
Conversion in DO Case
Null Statement

B.1.8. I/O

Implicit Positional Control

Explicit Positional Control

(Tabl, Skip...)

Conversions

Record Selection

Null Field, <;> in read statement

Field Width

Generation of Device Control Characters

B.2. SYSTEM SUPPORT FEATURES

B.2.1. MACROS (Replace)

Substitution of Arguments

Nested Macros

Scoping Conflicts

Argument Name as Argument

B.2.2. Includes

Selection of Correct Data Set Member

Text Included at Correct Point

Text Introduced Unchanged

Multiple Includes

Include Within Include (Error)

B.2.3. Files

Association of Channel Number with Data Set or Device

Opening and Closing Files

Standard Input File (SYSIN)

B.2.4. COMPOOLS

Template Declaration
Compilation of COMPOOL
Access of Contained Data
Storing Into COMPOOL
Access Rights
Multiple COMPOOLS
Locked Data

B.2.5. COMSUBS

External Procedure Declaration
External Function Declaration
External Procedure Reference
External Function Reference
Procedure Compilation
Function Compilation

B.2.6. Real Time Control

SCHEDULE
TERMINATE
WAIT
CANCEL
SIGNAL
ON ERROR
UPDATE PRIORITY
PRIO BUILTIN FUNCTION
SEND ERROR
RUNTIME
Events
Independent Processes
Process Events
Latched Events
Locked Timing
Update Blocks
Event Expressions
Tasks

B.3. PASS 1

B.3.1. Error Checks

Set off every error message.

Check that syntactically correct but semantically invalid statements produce errors.

Check recognition of syntactically invalid forms.

Check code production inhibition.

B.3.2. Output Writer

Single to multiple line conversion.

Automatic variable annotation

Symbol table dump

X-ref dump

Error summary

B.3.3. Scanner

Declaration Processing	Separate Category
Replace Processing	Separate Category
Symbol Table Entry	
Token Creation	
Multi-line Input	

B.3.4. Data Declarations

Conflicting Attributes

Factored Declares

Constant Attribute

Automatic Attribute

B.3.5. Scoping Rules

- DECLARE VARIABLES W/ SAME NAME.

REPLACE SEVERAL IDENTIFIERS OF THE SAME NAME IN DIFFERENT SCOPES.

STRUCT TEMPLATES WITH SAME NAME IN DIFFERENT SCOPES.

DECLARATION LOOPS

TEMPORARY LINKAGES

EXTREMELY DEEP NESTING

B.4. GENERAL

B.4.1. Library & Built-Ins

Matrix-Matrix

Matrix-Vector

Matrix-Scalar (Scalar-Matrix)

Vector-Vector

Vector-Matrix

Vector-Scalar

Scalar-Scalar

Matrix

Vector

Scalar

Integer

Character

Array

No Argument Functions

Use in Expressions

Use in Output Statement

Use as Subscript

B.4.2. Array-Structure Interaction

Assignment Comparison Multiple Assign Operators	Use with mixed structure and array.
Dense vs. Aligned	Ascertain that conversion still works.

B.4.3. Relational Expressions

Multiple Occurrence of &, |, ¬

Multiple Operators with Parentheses

IF..., THEN IF...THEN IF...

With Arrayed Relations (=, ¬=)

With Structured Comparisons

With Function Calls

B.4.4. Procedure Link & Parameter Pass

Linking to External Procedures

Passing Simple Parameters of 6 Data Types

Passing Multiple Parameters of Mixed Type

Assign Lists

Linking to External Functions

Passing Single Parameter of Each Data Type

Passing Multiple Parameters

Test Return of Six Types

Test 'X'-length Parameter

Implicit Conversions

Expressions as Arguments

Subscripted Variables in Assigns

Use of Arrays and Structures as Parameters, Assigns, and Returns

B.4.5. Addressability & Register Usage

Data

single array > 4096 bytes

many single elements totalling > 4096 bytes

many structures > 4096 each

random access indexing

computed access where optimization applies

Procedure

length > 1600 instructions

branching top-to-bottom, bottom-to-top...

DO case

DO WHILE

long iterative DO loop

repeat

large number of literals

automatic storage

register usage under all adverse conditions above

B.4.6. Conversion

a. Shaping functions covered separately except with single arg and radix

b. Implicit conversions not covered under individual data types:

int-bit

bit-int

char to int

char to scalar

char to bit

bit to char

c. @single, @double

B.4.7. Shaping Functions

single argument
arrayed arguments
radix
all levels of resultant subscripting
use with argument of size
unknown at compile-time
nested shaping fus

APPENDIX C: Test Categories & Test Programs (Listing current as of Release 8)

REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR

C.1. DISCRETE

C.1.1. Integer-Scalar

Assignments	HALTEST*
Initial Values	HALTEST*
Comparisons	HALTEST*
Infix Mathematical Operators	OPERTEST*
Exponentiation	EXPONTEST*
Prefix Operators	HALTEST*
Multiple Assignments	HALTEST* Multiple assignments to data types of differing type or precision will result in loss of precision.
Complex Expressions	HALTEST*
Complex Comparisons	HALTEST*

C.1.2. Vector-Matrix

Comparisons	MATVECT_COMPRE
Assignments	MATVECT_ELEMENT_SELECTION,* MATVECT_PARTITIONING,* MATVECT_EXPRESSIONS*
Initial Values	Same as above
Conversions	Implicitly tested in all Vector-Matrix routines.
Plus, Minus	MIXED_TYPE_TEST
Multiply (all Types) Divide	MIXED_TYPE_TEST
Exponentiation	MATRIX_EXPONENTIATION
Unary Operators	MATVECT_UNARY_COMPARE*
Element Selection	MATVECT_ELEMENT_SELECTION*
Partitioning	MATVECT_PARTITIONING*
Complex Expressions	MATVECT_EXPRESSIONS*
Complex Comparisons	MATVECT_UNARY_COMPARE*
Multiple Assignments	Multiple assignments to data types of differing precisions will result in loss of precision.
Built-in Functions	MATVECT_UNARY_COMPARE*

C.1.3. Bit

Assignments	NASNTEST
Initial Values	ANDTEST,* ORTEST,* NASNTEST,* CONCTEST,
Comparisons	COMPTEST
Conversions	Precision conversion implicitly tested in all tests.
Concatentation	CONCTEST
And, Or Operators	ANDTEST,* ORTEST*
Not Operator	NASNTEST*

Element Selection	BITTEST*
Complex Expressions	BITTEST*
Complex Comparisons	BITTEST*
Multiple Assignments	BITTEST*
SUBBIT	SUBBIT_ON_INTANDBIT SUBBIT_ON_SCALANDCHAR
Partitioning	BITTEST,* BIT_CHARACTER_SUBSCA

C.1.4. Character

Assignments	CHARACTERTEST*
Initial Values	CHARACTERTEST*
Conversions	CHARTEST*
Comparisons	CHARTEST,* CHARACTERTEST*
Concatenation	CHARACTERTEST,* CHARTEST*
Element Selection	TEST2, CINP, CH
Complex Expressions	CHARTEST*
Complex Comparisons	CHARTEST*
Multiple Assignments	CHARTEST*
Built-In Functions	CH, CHARTEST*
Partitioning	CHARTEST,* CH, TEST2, CINP, BIT_CHARACTER_SUBSCR

C.1.5. Structures

Assignments	No explicit tests
Comparisons	STRUCT_IO_COMP
Initial Values	STRUCTURE_TEMPLATE
Terminal Accessing	STRUCT_ACCESS
Element Selection	STRUCT_ACCESS
Partitioning	STRUCT_ACCESS
I/O	STRUCT_IO_COMP
Dense vs. Aligned	STRUCT_IO_COMP

C.1.6. Arrays

Arrayed Statements	ARRAY_BIT_CHAR_TEST ARRAY_PREFIX ARRAYASSIGN
Subscripting	ARRAY_PARTITION
Element_Selection	ARRAY_ELEM_SELECT
Comparisons	ARRAY_COMPARISON_TEST
Arrayed Subscripting	ARRAY_SUBSCR

C.1.7. Flow Control

Flow Control	FLOWTEST*, TEST2
I/O	Extensively Tested***

* TEST source published in previous memo.
** See Section 3 of this report.
*** Means that features are tested by most runs during development and use of HAL/S compiler.

REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR

C.2. SYSTEM SUPPORT FEATURES

C.2.1. MACROS (Replace)	REPL_TEST
C.2.2. Includes	Extensively tested INCLUDE_TEST
C.2.3. COMPOOLScompools	All subcategories covered by CSUB, COMPOOL, PROG
C.2.4. Comsubs	All subcategories covered by CSUB, COMPOOL, PROG
C.2.5. Real Time Control	Tested by PMF Simulator

C.3 PASS 1

C.3.1. Error Checks	CBLTIN EXPONENTIATION_ERROR SUBBIT_ERROR_TEST CH EVEN_TEST UNFACTORED_ERROR_TEST ASSIGN_CONTEXT_ERROR TERMINAL_SS_BIT_CHAR SUBBIT_IN_READ STRUCT_SUBBIT_IN_ASSIGN BLOCKSTRUCT_TEST NAME_PROG_ERR NAME_ERR STRUCT_ERR REPL_ERR
C.3.2. Output Writer	Extensively Tested
C.3.3. Scanner	Extensively Tested
C.3.4. Data Declarations	
Conflicting Attributes	CONFLICT_TEST
Factored Declares	Implicitly tested in many routines
Constant Attribute	CONSTANT_TEST
Automatic Attribute	Development Tests
C.3.5. Scoping Rules	Implicitly tested

C.4. GENERAL

C.4.1. Library and Built-Ins

Integer & Scalar	COS_TEST, COSH_TEST, LOG_SORT Values of results have been compared with a CRC tables and were found to be correct within the accuracy of the tables MAXIMUMPRODSUM MODTEST SIGN_SIGNUM_TEST DIVANDREMAINDERTEST CEILINGFLOOR_TEST EXPRESSION_AS_ARGUMENTS
Character	CH, CINP, CHARACTERTEST*, CBLTIN
Array	SIZE_TEST
No Argument Functions	Only RUNTIME Explicitly tested
Use in Expressions	Tested in the complex expressions subcategory of each category
Use in Output Statement	Development Tests
Use in Subscript	Tested as part of the general subscripting test for each datatype

C.4.2. Array-Structure Interaction	Development Tests
C.4.3. Relational Expressions	Subcategories covered by parts of FLOWTEST, HALTEST, CHARTEST, BITTEST, MATVECT_COMPARE, ARRAYCOMPAREISONTEST

C.4.4. Procedure Link & Parameter Pass

Linking to External Procedures	CSUB, PROG
Passing Simple Parameters of 6 Data Types	Development Tests
Passing Multiple Parameters of 6 Data Types	PARM, TEST2, T4, ECCHO
Assign Lists	PARM, ECCHO, T3
Linking to External Functions	Extensively tested by both FDI and PMF Simulators*

* See Section 3 of this report.

Passing Single Parameter of Each Data Type	Tested in the complex expressions subcategory of each datatype
Passing Multiple Parameters	PARM, T4
Test Return of Six Types	Tested in the complex expressions subcategory of each datatype
Test '*'-length Parameter	SIZE_TEST
Implicit Conversions	Used in all tests
Expressions as Arguments	EXPRESSIONS_AS_ARGUMENTS
Subscripted Variables in Assigns	SUBSCRIPT_ASSIGN_PARTITION
Use of Arrays and Structures as Parameters, Assigns, and Returns	STRUCT_PARM ARRAY_PARM

C.4.5. Addressability and Register Usage — Extensively Tested

C.4.6. Conversion

Integer-Scalar Shaping Functions	INTEGER_SCALAR_SHAPE, ISSHAPE_ON_MISC
Vector-Matrix Shaping Functions	VECFUN, VECTORSHAPE_TEST, MSHAPE

C.4.7. Shaping Functions

SUBBIT	SUBBIT_ON_SCALANDCHAR SUBBIT_ON_INTANDBIT
Bit Shaping Functions	BSHAPE
Character Shaping Functions	CSHAPE

C.5. NEW FEATURES

C.5.1. System HAL

NAME	NAME_PROG
REPLACE MACROS	REPLACE_BUGs*
% MACROS	Developmental Tests
TEMPORARY	TEMP_TEST
IN-LINE FUNCTIONS	INLIN_FUNCTIONS

* Set of routines to test stacking, parameters, macros-inside-macros, etc.

C.5.2. Error Library

An extensive error check library has been developed. Some programs in this library are designed to trigger the error message and error recovery mechanisms within the compiler. These tests verify whether the appropriate error messages are being generated and whether the error recovery mechanism responds in a fashion which allows subsequent statements within the compilation to be processed despite the fact that no code will be generated by the particular run. This allows the user to review the majority of his current syntactic or semantic errors before resubmitting the job. Other programs in the error check library are designed to trigger runtime errors. These tests verify the proper generation of an error message, and the possible trace information associated with the message. In addition they check the GO TO, IGNORE, and SYSTEM options available with the ON ERROR statement.

C.6. SPECIAL TESTS

- INT_TO_BIT, ED, QUEUES, PARENTHESIS, CHECKER, DEBUG, SCRUNCH, ARITH, TESTGEN, BTESTGEN, KIP, FORMAT, PMSPOOL, FDA
- FDI-Simulator
  - 2 Compools
  - 13 COMSUBS
  - 39 internal functions and procedures
  - Over 2000 HAL/S statements
- PMF-Simulator
  - 1 Compool
  - 4 programs } real time
  - 8 tasks }
  - 20 internal functions and procedure
  - Over 2000 HAL/S statements
- Math Function Precision Tests

APPENDIX D

HAL/S Discrepancy Report Log

HAL/S DISCREPANCY REPORTS

Number	Source		Title	Intermetrics Evaluation	Date Fixed	Fixed for Release	Comment
	Who?	Date Rec.					
1	CSDL	9/05/73	Replace Text Identifiers	confirmed bug	8/15/73	360-3A	
2	CSDL	9/05/73	Initialization of Integers	"	8/15/73	360-3A	
3	CSDL	9/05/73	Bit Function Writes	"	8/15/73	360-3A	
4	RI	9/17/73	Character Initialization	"	9/05/73	360-4	
5	CSDL	9/18/73	Trailing Under-scores in identi-fiers	"	9/25/73	360-4	
6	CSDL	9/18/73	END Labels	"	9/01/73	360-4	
7	CSDL	9/18/73	READ Statements	"	9/27/73	360-4	
8	IBM	10/04/73	Half-word sign bit	"	10/04/73	360-5	
9	IBM	10/04/73	Bit catenation	"	10/04/73	360-5	
10	CSDL	10/10/73	Replace Statements	"	10/14/73	360-5	
11	CSDL	10/18/73	Precedence Inconsis.	"	10/20/73	360-5	
12	RI	10/24/73	Arith. Functions	"	-	360-4	
13	RI	10/24/73	DO END Label	See DR #6			
14	IBM	10/29/73	Matrix Shape Error	Not available	in 360-4		

HAL/S DISCREPANCY REPORTS

Number	Source		Title	Intermetrics Evaluation	Date Fixed	Fixed for Release	Comment
	Who?	Date Rec.					
15	IBM	10/29/73	IF Statement	Not a bug			
16	IBM	10/29/73	Error cleanup after INITIAL	Confirmed bug	10/29/73	360-5	
17	IBM	10/29/73	END Lables	See DR #6			
18	IBM	10/29/73	EOF Looping	Confirmed bug		360-5	
19	IBM	10/29/73	Comparison failure	"		360-5	
20	IBM	10/29/73	ARCCOS Results	"		360-5	
21	I^2	10/08/73	Argument Count	"	10/12/73	360-5	
22	I^2	10/08/73	Array Loop Gener.	"	10/12/73	360-5	
23	I^2	10/12/73	Literal in Built-in	"	10/12/73	360-5	
24	I^2	10/15/73	Init. of Bit Var.	"	10/17/73	360-5	
25	I^2	10/17/73	Use of RO	"	10/17/73	360-5	
26	I^2	10/20/73	Shaping Functions	"	10/28/73	360-5	
27	I^2	10/28/73	Single Precision Literals	"	10/28/73	360-5	
28	IBM	10/22/73	Writer Formatting Error	"	11/1/73	360-5	
29	IBM	10/29/73	INITIAL/AUTOMATIC on Assign Var.	"	11/1/73	360-5	
30	IBM	10/30/73	ARCSIN/ARCTAN problems	"	11/11/73	360-5	

HAL/S DISCREPANCY REPORTS

Number	Source		Title	Intermetrics Evaluation	Date Fixed	Fixed for Release	Comment
	Who?	Date Rec.					
31	IBM	10/31/73	ELEMENT Selection from bit string	Confirmed bug	11/6/73	360-5	
32	IBM	10/31/73	Bit Partitioning	"	11/6/73	360-5	
33	IBM	11/1/73	Bit Partitioning and #	"	11/3/73	360-5	
34	CSDL	11/3/73	Inconsistency in Array Compares	-See DR #22			
35	I^2	11/3/73	Structure Assign Parms	Confirmed bug	10/6/73	360-5	
36	IBM	11/13/73	IF/ELSE Indentation	"	11/13/73	360-5	
37	IBM	11/13/73	Invalid Opcode Generation.	"	11/10/73	360-5	
38	IBM	11/28/73	Illegal FP Reg. Alloication	Side effect of #39			
39	IBM	11/28/73	Invalid Opcode Generation	Confirmed bug	11/15/73	360-5	
40	I^2 d	11/28/73	NONHAL Procs & Funcs	"	11/28/73	360-6	
41	I^2 d	11/28/73	Array Args of Functions	"	12/1/73	360-6	
42	I^2	11/27/73	Dynamic FP Range	"	11/28/73	360-6	
43	I^2	11/29/73	Arrayed Statement as case of DO CASE	"	12/1/73	360-6	

HAL/S DISCREPANCY REPORTS

Number	Source		Title	Intermetrics Evaluation	Date Fixed	Fixed for Release	Comment
	Who?	Date Rec.					
44	I^2	12/3/73	Repeat Group in Shaping	Confirmed bug	12/4/73	360-6	
45	I^2 d	12/6/73	$CHAR_{@DEC}$ conversions	"	12/7/73	360-6	
46	I^2 d	12/6/73	Exponentiation	"	12/7/73	360-6	
47	I^2	12/7/73	Abend in Shaping	"	12/8/73	360-6	
48	CSDL	12/15/73	Comment & Directive Cards	Not Error			Documentation Problem
49	CSDL	12/15/73	# in Subscript	Confirmed	12/20/73	360-6	
50	CSDL	12/15/73	Replace & Formatting	Not Error			
51	CSDL	12/15/73	Bit I/O	Confirmed	12/20/73	360-6	
52	I^2	12/13/73	CONSTANT Character Strings	"	1/2/74	360-6	
53	I^2	12/20/73	Bit Formal Parameters	"	12/20/73	360-6	
54	I^2	12/19/73	Functions in Arrayed Starts	"	12/19/73	360-6	
55	I^2	12/19/73	Exclusive Functions	"	12/19/73	360-6	

d: Reported by Draper Laboratory

REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR

HAL/S DISCREPANCY REPORTS

Number	Source		Title	Intermetrics Evaluation	Date Fixed	Fixed for Release	Comment
	Who?	Date Rec.					
56	I^2	12/19/73	Structure I/O	Confirmed	12/19/73	360-6	
57	I^2	12/21/73	Replace Text in Templates	"	12/21/73	360-6	
58	RI	12/20/73	Exceptions -Output Writer with REPLACE				
59	I^2	1/02/74	Bits in Range 9-16 bits	Confirmed	1/02/74	360-6	
60	I^2	1/02/74	Bit(16) Vars. and Assignments	"	1/02/74	360-6	
61	I^2	1/02/74	Structure Parameters	"	1/02/74	360-6	
62	CSDL	12/09/73	Precision	"	1/02/74	360-6	
63	CSDL	12/09/73	Addition Failure	Confirmed			eliminated in re-release of 360-6
64	CSDL	12/09/73	Array I/O	Part Confirmed			
65	CSDL	12/09/73	Macro-text Listings	Confirmed	12/10/73	360-6	
66	CSDL	12/09/73	Vector Shaping	Confirmed	12/10/73	360-6	
67	CSDL	12/09/73	Overflow	See DR #46			
68	CSDL	12/09/73	Overflow(2)	Confirmed			
69	CSDL	12/09/73	Character @ DEC	Confirmed	12/20/73	360-6	
70	CSDL	11/22/73	Procedure Parmeters	Confirmed	1/15/74	360-6	
71	CSDL	11/24/73	Arrayness Conflict	Confirmed	1/15/74	360-6	
72	CSDL	11/20/73	Link Step OC5	Confirmed	1/15/74	360-6	
73	CSDL	11/20/73	Phase II Error in Arrayed Args.	Part Confirmed	1/15/74	360-6	
74	CSDL	11/24/73	Indirect Stack Over-	Confirmed	1/15/74	360-6	

HAL/S DISCREPANCY REPORTS

Number	Source		Title	Intermetrics Evaluation	Date Fixed	Fixed for Release	Comment
	Who?	Date Rec.					
75	CSDL	11/16/73	Addressability	Compiler Restriction	1/15/74	360-6	
76	CSDL	12/7/73	Args of NONHAL Procedures	Not Error			
77	CSDL	12/21/73	Rounding	Confirmed	1/10/74	360-6	
78	CSDL	1/16/74	SUBBIT pseudo variable	Confirmed	1/30/74	360-6	
79	IBM	12/10/73	Abend 1/2 word BIT OPS	See DR #59, 60			
80	IBM	12/12/73	1/2 word bit compare	See DR #59, 60			
81	IBM	1/8/74	Variable Dump of Constraints	Not Error			
82	IBM	11/16/73	Statement # Trace	Not Error			
83	IBM	11/19/73	Trace Diagnostic Problem	?			Trace Package changed. Bug non-existent in Version 5.
84	CSDL	1/29/74	Catalog Procedure Error	Confirmed		360-6	
85	CSDL	1/29/74	Execution with Compiler Error	Confirmed	1/30/74	360-6	
86	CSDL	1/29/74	E&M's split over page	Confirmed	1/30/74	360-6	
87	CSDL	1/30/74	Block Summaries	Confirmed	1/30/74	360-6	
88	CSDL	1/29/74	Failure of MAX-MIN	Confirmed	3/3/74	360-7	
89	CSDL	1/29/74	RG9 Message Wording	Confirmed	1/30/74	360-6	

HAL/S DISCREPANCY REPORTS

Number	Source		Title	Intermetrics Evaluation	Date Fixed	Fixed for Release	Comment
	Who?	Date Rec.					
90	CSDL	1/29/74	Unlatched Event Initialization	Not Error			} documentation problems
91	CSDL	1/29/74	ODD Function	Not Error			
92	CSDL	1/29/74	Compile time OC5 Dumps	Not Error			
93	CSDL	2/12/74	Random Error Messages	Not Error			
94	CSDL	2/18/74	Writing Structures	Confirmed	1/30/74	360-6	
95	CSDL	2/18/74	Indexing Error	Confirmed	1/30/74	360-6	
96	CSDL	2/18/74	Double Explicit Conversion	Confirmed	2/12/74	360-6	
97	CSDL	2/18/74	Unlatched Event Initialization	See DR #90			
98	CSDL	2/18/74	Inter-Process Boolean	Not Error			
99	CSDL	2/18/74	Multiply Defined Compools vars.	Confirmed	1/30/74	360-7	
100	I^2	3/5/74	INTEGER/SCALAR shaping	Confirmed	3/12/74	360-7	
101	CSDL	3/20/74	Functions as Multipliers	Confirmed	3/15/74	360-7	
102	CSDL	3/20/74	Vector Length in dot product	Confirmed	3/15/74	360-7	
	CSDL	3/20/74	Built-ins in inline Functions	Cofnirmed	3/15/74	360-7	

HAL/S DISCREPANCY REPORTS

Number	Source		Title	Intermetrics Evaluation	Date Fixed	Fixed for Release	Comment
	Who?	Date Rec.					
104	CSDL	3/20/74	Inlines in Replaces	Confirmed	3/15/74	360-7	
105	I^2	3/5/74	Argument of SIZE	Confirmed	3/12/74	360-7	
106	I^2	3/7/74	UPDATE BLOCKS & DO CASE	Confirmed	3/15/74	360-7	
107	I^2	3/7/74	Output Writer Subscript Problem	Confirmed			
108	CSDL	3/20/74	Symbol Table Loop	Confirmed	3/18/74	360-7	
109	CSDL	3/20/74	DECLARE Tasks	Confirmed	4/08/74	360-8	LCR Required
110	CSDL	3/20/74	Single Prec. Explicit Conversion	Confirmed			
111	CSDL	3/20/74	DO CASE Error	See DR #106			
112	CSDL	3/27/74	HAL/S Initialization	Confirmed	3/25/74	360-7	
113	CSDL	3/27/74	READ-ALL Structures	Confirmed	3/25/74	360-7	
114	CSDL	3/27/74	READ-ALL Skip	Confirmed	3/25/74	360-7	
115	CSDL	3/29/74	Infinite Loop in GO TO	Confirmed	3/25/74	360-7	
116	CSDL	4/05/74	Init. of Var. with NULL String	Confirmed		360-8	
117	IBM	4/24/74	Garbage in Macro Expansion	Confirmed		360-8	
118	IBM	4/24/74	Precision of Mixed Prec. Divide	Confirmed		360-8	

REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR

HAL/S DISCREPANCY REPORTS

Number	Source		Title	Intermetrics Evaluation	Date Fixed	Fixed for Release	Comment
	Who?	Date Rec.					
119	I^2	4/18/74	Range of Error Numbers	Confirmed		360-8	
120	I^2	4/24/74	Source Macro's & Templates	Confirmed		360-8	
121	I^2	5/02/74	Transpose Problem			360-8	
122	CSDL	5/02/74	Nested SUBBITS	Spec. Problem			
123	CSDL	5/02/74	AP-101 Timing Anomalies	Confirmed		360-8	
124	CSDL	5/02/74	DEVICE Directive	Confirmed		360-8	
125	CSDL	5/13/74	Assignment of BIT Literals	Confirmed	5/19/74	360-8	
126	CSDL	5/13/74	Reading Structures	Confirmed	5/19/74	360-8	
127	CSDL	5/13/74	Size Function	Confirmed	5/19/74	360-8	
128	CSDL	5/13/74	Var. names length 32	Confirmed	5/19/74	360-8	
129	I^2	6/1/74	Phase I abend in macro expan.	Confirmed	5/12/74	360-8	
130	I^2	6/1/74	Replace macros in symbol table	Confirmed	6/12/74	360-8	
131	I^2	6/1/74	Literals & Constants	Confirmed	6/12/74	360-8	
132	I^2	5/16/74	Names of programs & tasks	Confirmed	6/5/74	360-8	

HAL/S DISCREPANCY REPORTS

Number	Source		Title	Intermetrics Evaluation	Date Fixed	Fixed for Release	Comment
	Who?	Date Rec.					
133	IBM	5/15/74	Simulation Vector Table	ICD Discrepancy			
134	IBM	6/1/74	OCI Abend in 02	Confirmed	6/12/74	360-8	
135	IBM	6/12/74	Bad code for DO CASE	"	6/19/74	360-8	
136	IBM	6/1/74	Bad printout of structure terminals	"	6/19/74	360-8	
137	I^2	6/19/74	'<label>' for '('				

Appendix E

REPRODUCIBILITY OF THE ORIGINAL PAGE IS POOR

Discrepancy Worksheet

Discrepancy Number	Disposition					
	360-1,2,3	360-4	360-5	360-6	360-7	360-8
1	✓ O X					
2	O X					
3	✓ O X					
4	✓ O	X				
5	✓ O X					
6	✓ O X					
7	✓ O	X				
8	✓ O	O	X			
9	✓ O		X			
10		✓ O X				
11	✓	O	X			
12	✓ O	X				
13	–					
14	–					
15	–					
16	✓ O	O	X			
17	–					
18		✓ O	X			
19	✓ O		X			
20	✓ O		X			
21	✓	O	X			
22	✓	O	X			
23	✓	O	X			
24	✓	O	X			
25	✓	O	X			
26	✓	O	X			
27	✓	O	X			
28	✓ O		X			
29	✓	O	X			
30	✓ O		X			

PRECEDING PAGE BLANK NOT FILMED

Legend:

✓:	introduced	X:	fixed in
O:	found in		

Discrepancy Number	Disposition					
	360-1,2,3	360 -4	360 -5	360 -6	360 -7	360 -8
•						
31	✓ O		X			
32	✓ O		X			
33	✓	O	X			
34	–					
35		✓ O	X			
36	✓	O	X			
37	✓	O	X			
38	–					
39		✓ O	X			
40			✓ O	X		
41	✓		O	X		
42	✓		O	X		
43	✓		O	X		
44			✓ O	X		
45		✓	O	X		
46	✓		O	X		
47		✓	O	X		
48	–					
49	✓	O	X			
50	–					
51		✓ O	X			
52	✓		O	X		
53	✓		O	X		
54		✓	O	X		
55			✓ O	X		
56			✓ O	X		
57	–		✓ O	X		
58	–					
59	✓		O	X		
60	✓		O	X		

Discrepancy Number	Disposition					
	360-1,2,3	360-4	360-5	360-6	360-7	360-8
61		✓	O	X		
62	✓		O	X		
63			✓ O	X		
64		✓	O	X		
65			✓ O	X		
66			✓ O	X		
67	–					
68	✓		O	X		
69		✓	O	X		
70	✓		O	X		
71	✓		O	X		
72			✓ O	X		
73	✓		O	X		
74			✓ O	X		
75	✓		O	X		
76	–					
77	✓		O	X		
78		✓	O	X		
79	–					
80	–					
81	–					
82	–					
83	–					
84			✓ O	X		
85	✓		O	X		
86	←		✓ O	X		
87			✓ O	X		
88		✓	O		X	
89			✓ O	X		
90	–					

Discrepancy Number	Disposition					
	360-1,2,3	360 -4	360 -5	360 -6	360 -7	360 -8
91	–					
92	–					
93	–					
94			✓ O	X		
95	✓		O	X		
96		✓	O	X		
97	–					
98	–					
99	✓		O		X	
100		✓		O	X	
101	✓			O	X	
102	✓			O	X	
103				O	X	
104				O	X	
105		✓		O	X	
106	✓			O	X	
107	✓			O	X	
108			✓	O	X	
109				✓ O		
110	✓			O	X	
111	–					
112	✓			O	X	
113		✓		O	X	
114	✓			O	X	
115		✓		O	X	
116				✓	O	X
117				✓	O	X
118	✓				O	X
119					✓ O	X
120				✓	O	X

Discrepancy Number	Disposition					
	360-1,2,3	360-4	360-5	360-6	360-7	360-8
121		✓			O	X
122	–					
123					✓ O	X
124	–					
125	✓				O	X
126				✓	O	X
127				✓	O	X
128				✓	O	X
129				✓	O	X
130			✓		O	X
131		✓			O	X
132				✓	O	X
133	–					
134	✓				O	X
135				✓	O	X
136	✓				O	X
137	–					
138						
139						
140						
141						
142						
143						
144						
145						
146						
147	–					
148						
149						
150						