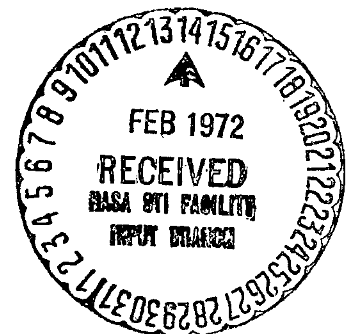
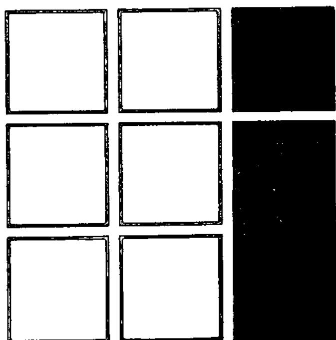




INTERMETRICS

N72-16145

(NASA-CR-115367) CONTINUED ADVANCEMENT OF
THE PROGRAMMING LANGUAGE HAL TO AN
OPERATIONAL STATUS Final Report
(Intermetrics, Inc.) 30 Dec. 1971 142 p
CSCL 09B

G3/08 14476

FAC (NASA CR OR TMX OR AD NUMBER) (CATEGORY)

Reproduced by
NATIONAL TECHNICAL
INFORMATION SERVICE
U S Department of Commerce
Springfield VA 22151

Final Report

CONTINUED ADVANCEMENT OF THE PROGRAMMING LANGUAGE HAL TO AN OPERATIONAL STATUS

NAS 9-11944

December 30, 1971

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

NOTICE

THIS DOCUMENT HAS BEEN REPRODUCED FROM THE BEST COPY FURNISHED US BY THE SPONSORING AGENCY. ALTHOUGH IT IS RECOGNIZED THAT CERTAIN PORTIONS ARE ILLEGIBLE, IT IS BEING RELEASED IN THE INTEREST OF MAKING AVAILABLE AS MUCH INFORMATION AS POSSIBLE.

TABLE OF CONTENTS

	Page
FOREWORD	
1.0 TASK SUMMARY	1
1.1 Task I: Maintenance and Training	1
1.2 Task II: Advanced HAL Development	1
2.0 HAL INSTALLATION AND ON-SITE SUPPORT	3
2.1 Installation	3
2.2 Program Changes and Maintenance Procedures	9
3.0 HAL COURSES	11
3.1 General Description	11
3.2 Course Preparation	11
3.3 Course Outline	12
4.0 NECESSARY MODIFICATIONS AND ADDITIONS	15
4.1 Storage Allocation Problem	15
4.2 Miscellaneous Improvements	16
5.0 HAL TRANSFERABILITY	19
5.1 Technical Approach	19
5.2 Translation of XPL Programs Into HAL	23
5.3 Feasibility Demonstration ("HAL-in-HAL")	35
APPENDIX A. HAL Course Material	41
A.1 Overview	43
A.2 Longer HAL Course	69
APPENDIX B. HAL-in-HAL Detailed Description and Listing	89

FOREWORD

This document represents the final report of a contract for the continued advancement of the programming language HAL to an operational status. The effort was sponsored by the National Aeronautics and Space Administration's Manned Spacecraft Center in Houston, Texas under Contract NAS-9-11944 . It was performed by Intermetrics, Inc., Cambridge, Mass. under the technical direction of Mr. Daniel J. Lickly. The Technical Monitor for NASA/MSC was Mr. John Garman, FS/6.

The publication of this report does not constitute approval by the National Aeronautics and Space Administration of the findings or the conclusions contained therein. It is published only for the exchange and stimulation of ideas.

1.0 TASK SUMMARY

The objectives and time duration of this contract were limited. Essentially, over the summer months of 1971 HAL was to be installed on the 360/75 at MSC, on-site support provided, users trained by class sessions, the compiler updated as necessary and experience gained with the language. The work was divided into two broad areas: maintenance and training, and advanced HAL development.

1.1 Task I: Maintenance and Training

Under this task, Intermetrics established a support systems programmer (Mr. Ronald Kole) at MSC within the Flight Software Section. Mr. Kole succeeded in solving some of the formidable problems associated with running at the RTCC under RTOS and transferred HAL from Intermetrics' 360/65 to MSC's 360/75. In addition, from time to time, necessary modifications originating in Cambridge and in Houston were incorporated at both sites and specifically, a general compiler-version update procedure was developed and implemented at MSC. The most significant modification was a redesign of the storage allocation algorithms, described in detail in Section 4.2 of this report.

Also as part of this task activity, 36 hours of training classes were conducted at MSC and the MIT Draper Laboratory. A total of approximately 50 people attended, including both government and industrial personnel.

Although no modifications were made which affected the HAL Specification or Guide documents, a new complete description of HALMAT, the intermediate code, was issued.

1.2 Task II: Advanced HAL Development

The objective of this development task was to increase the transferability of HAL to another host computer. The approach taken was to demonstrate that the compiler itself could be written in HAL. If this were accomplished then the entire compiler could be compiled on the 360 into FORTRAN and then the FORTRAN moved to almost any other large computer with only minor modifications. Toward this end a portion of the HAL compiler was coded in HAL and demonstrated to work on the 360/75 at the MIT Draper Laboratory. The portion selected exercised the bit and character handling features of HAL and indicates the feasibility of the approach.

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

PRECEDING PAGE BLANK NOT FILMED

2.0 HAL INSTALLATION AND ON-SITE SUPPORT

2.1 Installation

The HAL compiler and associated software were developed at the Harvard Computing Center in Cambridge, Massachusetts using a 360/65 running under OS/360 MVT Release 18. Transferability problems were encountered in establishing a usable HAL system at MSC. These problems may be broken down into two categories: (1) logistics and (2) internal software compatibility. Before the second set could even be recognized,the first had to be solved.

2.1.1 Logistics Problems

The logistics problems are the ones associated with the differences in operating procedures and at the systems' two installations. The preparation and submission of jobs at Harvard were done almost entirely through the Conversational Remote Batch Entry System (CRBE). The Harvard system was equipped with four IBM 1403 printers, each having a full PL/1 character set. The entire HAL system was maintained on a disk pack that was mounted by the operator when needed.

In contrast, the RTCC at MSC uses IBM 360/75 computers running a modified OS MVT which they call RTOS (Real Time Operating System). The RTOS version available during the June-September 1971 installation period corresponded roughly to OS/360 Release 18. The RTCC does not support the CRBE system, so all input submission was done via punched cards. This in itself was something of a problem since some pieces of the compiler and even some updates to the compiler were quite long. Also, the handling of cards and the chance of error in mixing up cards was considered less desirable than an on-line editing/ submitting system like CRBE.

2.1.1.1 RTCC Limited Disk Space. The RTCC also has very limited available disk space. In fact, it is impossible to have a permanently saved disk on the system. Therefore, a reasonable way had to be found to give users access to the HAL compiler. The method settled on was to use a dump/restore tape. When a user wants to run something that is located on the HAL disk (known as HAL001 at the RTCC) he tells the operator of the requirement for this special disk. The operator must "restore" the disk from a tape before the user's actual job can run. This means that he must run a background utility to transfer the contents of a special tape to some existing disk on the

system so that any user programs requesting HAL001 will find the disk mounted. After the user's run finishes, the HAL001 disk is scratched to make room for perhaps some other user's restored disk.

This system works, but it puts all jobs that require HAL001 in a special class that is usually run only during the early morning hours. This obviously precludes any same-day turnaround as can sometimes be obtained with "non-special" device requirements. There is, however, an advantage that is gained indirectly through use of the dump/restore tape. Since it is not any one physical disk that is mounted to satisfy requests for HAL001, it is possible to have serial versions of the disk available as separate dump/restore tapes. The user need only specify which dump/restore tape he wants used to create HAL001. Since one HAL001 is the same to the system as any other, all of his Job Control Language will work no matter which dump/restore tape is used.

The more troublesome aspect of the dump/restore system is the creation of an updated HAL001 and getting that new version put onto a D/R tape. The method of updating is to submit the job that makes the desired changes, requesting that HAL001 be restored in the normal manner, but also requesting that the disk to which the restore will take place be completely erased first. The job is then run as a regular batch job. After the run is completed the operator is requested to "dump" HAL001 to some specified tape. This tape then becomes the updated restore tape.

The problem in this system can occur in many ways. The disk to be restored must be erased first. This is to assure that after the update run, HAL001 contains <u>only</u> the HAL001 files. The dump program that does the disk to tape transfer will copy anything that it finds on the disk, no matter where it originated. So failure to erase the disk, while not really causing any errors in the resulting D/R tape, makes the tape file very long. This also means that subsequent restores from the tape will take a much longer time. All of this degrades performance of the overall system. It is also possible, if an unerased disk is cluttered enough, to overflow one D/R tape which causes even more complications.

Another source of error is the possibility that the operator will forget to dump the disk at all. Since this is an operator controlled utility, the user gets no indication on his output if the disk was actually dumped. The only real way to tell is to submit another job the next day to see if the updates were indeed saved. There is no way of telling if the restore tape file is too big because of a non-erased disk, other than watching the restore take place and guessing whether the tape moves a reasonable distance. Once an oversized tape file has been created it is very difficult to get rid of the unwanted "garbage". Thus, the

dump/restore tape method of supplying users with a HAL system works fairly well from the user's viewpoint (although turnaround is probably adversely affected) but is error prone from the system maintenance point of view.

The real solution to the D/R problem is to eventually catalogue all of the HAL system components on the system library as is done with the other language translators. This could be done now, but such a method is even more difficult to update and maintain and should not be used for anything short of a non-changing, well established version of the system.

2.1.1.2 RTCC Limited Print Facility. One other source of frustration at the RTCC can be identified. The RTCC is basically designed to run space missions and not necessarily to support batch users. The main outputs of mission programs are real-time displays and telemetry. These require little hard copy I/O. The main output of the batch user is printout. Yet the RTCC has only one printer per machine with two others shared between all machines. The 360/65 at Harvard manages to keep four printers busy by itself. So it would seem that the faster model 75 would generate a print backlog with even the maximum of three printers attached. This did indeed seem to be the case at MSC. This print backlog, of course, results in reduced turnaround. The HAL compiler proper output requires a full PL/1 character set. At the RTCC there is only one printer that has the UCS (Universal Character Set) feature that can support a print set other than the standard limited FORTRAN 48 character one. All of the output that requires this special character set is spooled on tapes until such time as there is enough to make the mounting of the special print train worthwhile. That point is reached two or three times a day. So HAL print delays are generally even longer than those caused by the regular print backlog. A small point further slowing down the printing is that the print train that was used for HAL was a special one, known as the Philco train. Some characters used by HAL only appear once on this train. This delay in waiting for the single character to move to the correct print position makes a large reduction in the speed of the printers (a visual estimate would be 20 to 40% slower).

The necessity of mounting the special character set created another potential error situation. If for some reason the class of output was mis-written on the spool tape, or misread, the output might (and has on occasion) be printed on the Fortran print train. It is, of course, then unreadable. The net result of these operation dependent procedures was a 1 run per day situation at best. Some preliminary users experienced even longer delays.

2.1.2 Software Incompatibility

The second class of problems relating to implementing HAL at MSC concern the operating system peculiarities that are present in any system. The most significant difference between RTOS at MSC and OS/360 was the handling of core allocation. One part of the HAL compiler called the Submonitor was significantly affected by the RTOS scheme.

The Submonitor is an assembler language program whose purpose essentially is to provide an I/O interface with the operating system. It also has the task of obtaining a block of core storage into which the actual compiler machine code can be placed for execution. In the original version of the Submonitor at Harvard, the necessary core was requested by means of a GETMAIN macro instruction. The operating system then gave the Submonitor the requested core out of the remaining part of the task's region as specified in the REGION parameter. Thus, it was possible to use the region parameter to determine how much core was made available to run the compiler.

2.1.2.1 <u>OS/360 Core Allocation</u>. The standard OS scheduling algorithms keep jobs in a waiting queue until such time as there is available in the machine, enough core in a contiguous block to satisfy the job's REGION request. Then the job is started and is free to do whatever it wants with its core. This block of contiguous core is reserved for the one task, even if the task uses only a small part of it. Thus, if an adequate REGION parameter is supplied, the Submonitor is guaranteed the availability of the core it needs to run the compiler. The Submonitor, in fact, makes maximum use of the entire available core by using a form of the GETMAIN Macro instruction that gives the operating system a minimum and maximum value of acceptable core regions. The maximum is set very high so that the operating system, in trying to come as close as possible to the maximum requested allocates all of the core remaining in the Submonitor's region.

2.1.2.2 <u>RTOS Core Allocation.</u> RTOS, however, does not make a job wait until the requested REGION is available in a contiguous block. It waits only until the sum of all free core in the system meets or exceeds the requested REGION. In fact, the decision of which jobs to run at any one time is based on a 125% allocation of core on the assumption that not all jobs will require their entire region at the same time. So in the case of the HAL compiler which requires about 4K bytes of core,

the operating system will load and execute the Submonitor as soon as there is 4K contiguous core available and enough other free core to satisfy the rest of the specified REGION parameter (this is usually 300K for HAL). The Submonitor then immediately issues a GETMAIN for the remaining core. This is where the operating system differences cause problems.

Under RTOS, a GETMAIN of the original form used at Harvard causes the operating system to put the Submonitor task into a wait state until the maximum of the minimum/maximum pair is available. For a large maximum this never happens and the job must eventually be cancelled by the operator.

2.1.2.3 Some Solutions. The first attempt to fix this was made by allowing the user to specify in his JCL the minimum and maximum values to be used in the GETMAIN request instead of the Submonitor's default values. This approach will apparently work for smaller programs, but for larger programs, there seems to be some system prescribed limit to the amount of core that can be obtained in this way. The Submonitor was never able to obtain more than 262, 144 (or 2^{18}) bytes of core. This is not enough to run the compiler and another method was needed.

The second method was to change the type of GETMAIN to one that requested a single specific size piece of core. The user was given the ability to specify this number through a keyword in his JCL. This system partially solved the problem. It was possible to obtain the correct amount of core this way, but another problem persisted. Even though RTOS could guarantee the existance of 300K bytes of free core, it could not guarantee how this core was divided up as could the regular OS. It was possible for the Submonitor to go into a wait state while the operating system tried to supply its contiguous core requirement. Under the right circumstances, this might take an hour or more. Whenever a task goes into the wait state, RTOS monitors how long it stays there. After some length of time RTOS begins sending the operator messages informing him of the lack of progress of the task. The operator makes the decision on the length of time the task is allowed to wait. After he gets tired of seeing the periodic messages, he usually cancels the job. The problem with this method is that the operator is never told why the job is waiting; it may be waiting because of some programmer error. He really has no chance to evaluate the situation.

The next step taken toward insuring a successful run was to take the wait for core out of the running Submonitor and put it in the pre-execution allocation. This simply meant doing away with the GETMAIN and giving the Submonitor a built-in

storage area big enough to run HAL. Under this system, the Submonitor became about 300K bytes long. This forced the operating system to find the 300K in a contiguous region before it could even load the Submonitor. As before, it was still possible for the available core to total more than 300K and yet not have 300K contiguous, but now the operator got a message saying "JOBXXX AWAITING A REGION". In this way the operator was informed of the real cause for the delay and was more willing to let the job wait for the core to become available.

Although the probability of getting a job to run has been increased, it is still possible for the job to be cancelled when it was the only user job in the system and was still unable to get its core. This happens when there are "backround" utilities running. Although they are termed "backround", they still compete for core like any other job on the system and it is possible for them to tie up core in such a way that a HAL job will not run even though it appears to be all alone on the system.

One small drawback to this final state of the Submonitor is that it is no longer possible to use the REGION parameter, or any user keywords to limit the size of the available core. This is not considered much of a handicap since the size of the HAL compiler is quite stable and is expected to remain so.

There is an alternate approach to solving the core lockout problem. This would involve a redesign of the HAL compiler structure to give it a scatter loadable attribute. If this were done, the required core would not need to be contiguous; several smaller contiguous areas would be requested. The probability of finding these smaller areas would be greater than the present system. There would, however, still be a finite chance that even these smaller regions would not all be available. The situation is such that the more the core requirement is split up, the more chance there is that the resulting smaller pieces will be found. More pieces of code, however, require much more overhead to maintain. Also, the redesign of the HAL code to allow such a split would be a difficult job that would not further the goal of producing a better overall compiler. The frequency of run failures under the present system is very low and sporadic. For a large, ground-based, batch-oriented system like the RTCC, additional time spent on refinement of HAL running procedures, would be of little value compared to the same time spent on refinement of the actual compiler code.

This potential lockout problem is not peculiar to HAL. It can happen to any job on the system whose core requirements are of the same size and nature as HAL's.

2.2 Program Changes and Maintenance Procedures

During the June-September 1971 time period many changes were made to both the HAL Submonitor and the compiler itself. Most of the changes made to the Submonitor are detailed in 2.1 above. These changes were basic in nature and took a large part of the summer to research and implement properly at MSC. In addition, new compiler versions were developed at Intermetrics and were sent to Houston on magnetic tape where copies of all of the files on the tape were put on MSC-owned tapes. The required files were then transferred from tape to the HAL001 disk and the disk then dumped to a dump/restore tape as described above.

2.2.1 Updates

A three tape dump/restore system was established to maintain the integrity of the system. One tape, called the system tape was the only one available to users. It always contained the most recent released version of the compiler. Thus, users only had to have this tape number to run HAL compilation. The two other tapes were development tapes. They were used in an alternating manner to build and checkout a new release. The alternation was necessary to provide a backup in case of some failure to make a good update. To make an update, the newest version development tape was used to restore HAL001. The update was made to the disk and then HAL001 was dumped to the alternate development tape. Even if the dump was not done, or if the update was unsuccessful, the original restore tape was still intact.

Once a version was considered ready for release, the development tape on which it resided was simply copied onto the system tape. Users specifying the system tape number as the HAL001 dump/restore automatically got the new release.

In addition to updates originating in Cambridge, some changes were made to the compiler at MSC. Small changes were communicated to Cambridge directly by long distance through the CRBE system at Harvard. In the case of larger updates, tapes were exchanged.

2.2.2 Summary of Changes Made At MSC

a) Research and implement the changes to the Submonitor to allow a more reliable core allocation.

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

b) Fix numerous small bugs found during checkout both in the HAL code itself and in the HAL run time library.

c) Partially implement and lay the ground work for a more complete listing generator as detailed in the HAL Guide. This involved providing additional functions in the Submonitor to allow the HAL compiler to set a maximum number of lines per page of listing and to dynamically request the line number of the current line on the page. These new functions helped to lay the framework for the ability to control completely the layout of the HAL listing. The listing was changed to the extent that the statement and line numbers were made available and the format of the format of the printed source code changed to increase readability.

3.0 HAL COURSES

3.1 General Description

Intermetrics personnel prepared and conducted three HAL langauge courses during the contract period. The material was designed for two types of audiences: 1) those seeking a broad "brush" overview of HAL, 2) those intending an indepth exposure to HAL. Two 15-hour sessions (2 1/2 days each) were given at the Manned Spacecraft Center in Houston. The first, primarily for NASA personnel and the second, for industrial contractors and other government agencies with an interest in higher order languages. For each session, the first three hours were devoted to the HAL overview; however, the overview itself was considered an integral part of the longer course.

A special third session was also conducted at the MIT Draper Laboratory, for Laboratory personnel and local industrial contractors. Because of the familiarity of these personnel with MIT's MAC language and certain similarities between HAL and MAC, an effective 1-day, 6-hour course was held. The course consisted of the overview, with elaborations and discussions, followed by a rapid presentation of the salient features of HAL.

In general, the participating students at MSC and MIT were highly motivated to learn HAL and always attempted the place HAL in perspective with respect to Shuttle applications. As a result, many provocative questions were asked and in some circumstances material discussed in class was fed back into the HAL design.

3.2 Course Preparation

The HAL courses were prepared with two objectives in mind: an overview, and a detailed study. For the overview, a balanced presentation of most of the important features and rationale incorporated into the HAL Specification Document (MSC-#01846) was designed. The purpose here was to illustrate how HAL satisfied, for the most part, the requirements imposed on a programming language for the Shuttle. Toward this end readability, vector-matrix arithmetic, data management, systems programming, real-time control and software reliability were emphasized.

The material was presented in vu-graph form and included numerous "Shuttle-like" application examples and commentary which included Intermetrics' experience with Apollo software

development. Particular attention was paid during the overview to indicate which HAL features would not be included in the first implementation for the IBM 360/75 at MSC. This was especially true for the descriptions of real-time control, controlled data sharing, and the error recovery features.

The longer HAL course was designed as an in-depth study of HAL and the overview served as an excellent orientation. This part of the course was based closely on the HAL Guide (MSC #01848)and the material was a combination of vu-graphs, references, to the Guide text and blackboard work. Only those features actually intended for implementation on the first 360/75 version were covered. (This specifically excluded real-time control, etc.)

Levels of increasing detail were presented, first with a set of vu-graphs covering all of the language features of HAL; i.e., operations, declarations, indexing, control, etc. followed by a careful tour through selected portions of the Guide. The Guide work illustrated usage, described many examples and motivated class discussions (and, in fact, contributed to subsequent corrections to the Guide). The technique of repeating subject material in levels of increasing detail; i.e. from overview to construct description to Guide with examples, proved to be an effective method of rapid assimilation and study.

In addition to text material and lecture, each student was provided with a HAL problem set as a homeword exercise. Unfortunately, few found the time to actually address these problems out of class. However, during the last class session, prepared problem answers were distributed and each problem was carefully "talked-through". Actual runs on the 360/75 by the students were contemplated during the course preparation, but 360 turn-around time within the RTCC facility was not consistent with the 2 1/2 day course duration.

An outline of the HAL course material is presented in the next section and the vu-graphs for both the overview and the longer course are collected in Appendix .

3.3 Course Outline

3.3.1 Overview (vu-graph material)

1. Higher order language motivation and capabilities.

2. Salient features of HAL

3. Data types

4. Program organization and structure

5. HAL Statements

6. Specific Examples

7. Real-time control, including data sharing and error recovery

8. Summary

3.3.2 Longer Course (Vu-graph Material)

1. Data Operations

2. Data Declarations

3. Indexing: partitions and use of subscripts

4. Control and branching mechanisms

5. Name scope rules

3.3.3 Longer Course (Guide Material)

1. Two-dimensional input-output format

2. HAL_M (HAL Mathematical Subset)

   a. Data and declarations
   b. Arithmetic expressions
   c. Assignment statements
   d. User-defined functions (SCALAR, VECTOR, MATRIX)
   e. IF Statements
   f. Illustrative problems - I
   g. Subscripts
   h. DO Statements
   i. Illustrative problems - II
   j. Subroutines; i.e., HAL PROCEDURES

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

k. Illustrative problems - III

l. Name scope

m. I/O Facilities

n. Illustrative problems - IV

3. Integer and Bit String Data

4. Structures

5. Bit and Character String Manipulations

6. Subscript facilities: complete

7. Implicit conversion of mixed data types

8. User-defined functions: complete

9. Array processing

10. Shaping functions: complete

11. REPLACE and DEFAULT Statements

12. "Talk-through" of problem set

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

4.0 NECESSARY MODIFICATIONS AND ADDITIONS

Redesign work was undertaken to increase the scope and capabilities of the HAL compiler and to promote its transferability to other computers. The first step was to redesign the variable storage philosophy and mechanization. Extensive design sessions were conducted to develop a suitable memory storage allocation system that would support the most general future goals of HAL, especially transferability (see 4.1 for more detail). This included the techniques necessary to support the calling of separately compiled HAL programs and the sharing of their data through a COMPOOL. This capability, in some form, is vital to the production of a multipass compiler.

During this time items were also dealt with that were either incomplete or had been newly defined. Thus, certain "holes" in HAL's capabilities were filled in. In addition, a number of shortcomings which had been uncovered were remedied.

4.1 Storage Allocation Problem

Certain storage allocation problems encountered during the implementation of some of the more advanced features of HAL in Phase II of the compiler (Fortran code generation) had necessitated basic conceptual changes in the allocatin algorithms in the compiler.

In the original version of the algorithms, temporary storage required for partial numerical results was allocated when needed during the code generation of a HAL statement, and freed-up again not later than at the end of the statement. This caused two major difficulties. Firstly, when temporary storage was required to hold the value of an argument in a procedure or user function invocation, special "unfreeable" temporary storage had to be used to prevent it from possibly being reallocated in the body of the procedure or user function. Secondly, in HAL statements containing user function invocations, (possibly nested), temporary storage allocated for partial results before the invocation code was generated had also to be masked "unfreeable" for the same reason. Other more subtle considerations finally made a complete restructuring of the algorithm essential.

The idea of providing completely dynamic storage allocation of execution time was rejected as requiring too many basic changes in the mode of operation of Phase II of the compiler. Instead a static scheme similar in some respects to the original

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

was adopted. In this scheme temporary storage for the program, and for each procedure or user function are allocated within mutually non-overlapping segments. A program or subprogram may have one segment or several non-contiguous segments of varying sizes dedicated for its use, depending on its requirements. The sum total of all segments constitutes a single continuous area of storage (except possibly for word boundary alignments). At the microscopic level within the bounds of a segment, storage is allocated and freed exactly as it was in the original allocation scheme.

Under this scheme, no temporary storage need be marked "unfreeable" no matter to what use it is put. Furthermore, the scheme has resulted in considerable simplification and unification of other storage allocation mechanisms in operations at code generation time.

4.2 Miscellaneous Improvements

1. Arraynesses:

    * reorganization of the mechanism controlling the utilization of statement arraynesses, especially with regard to utilization by arrayed subscripts of arrayed variables, and by the arguments of user functions.

    * implementation of the arrayed subscripted variable as an input or assign argument in a function or procedure call.

2. Cosmetics and Statistics:

    * generation of Phase II timing information, improvement of error message format, generation of statistics on certain critical parameters of Phase II operations

    * introduction of toggle directives to control Phase II and subsequent Fortran IV operation.

3. Shaping Functions:

    * introduction of a limited range of shaping and conversion functions: INTEGER, SCALAR, MATRIX, and VECTOR (no arrayed arguments or results).

4. Program Calling:

    * setting-up operating mechanisms for calling independent (i.e. separately compiled) HAL programs to "any" nest level, non-recursive

* creation of mechanisms for saving HAL programs in an object library.

5. I/O Routines

   * first, implementation of full-scale HAL READ/WRITE statements fixed, uni-channel input and output, fixed record length (printer and punch only).

6. Bit Strings:

   * fundamental bit string operators were implemented. Included were terminal and array subscripting and the AND, OR, and NOT operations. Bit strings are limited to not more than 32 bits; they have been implemented in full-word, half-word, and byte form.

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

PRECEDING PAGE BLANK NOT FILMED

5.0 HAL TRANSFERABILITY

5.1 Technical Approach

5.1.1 Background

The quest for easy transfer of operational programs from one computer to another has occupied the minds of many men since the early days of computer technology. The importance of this capability has grown considerably as the computer explosion has populated our society with countless kinds and types of computers with ever decreasing and more attractive price tags, and yet soaring software costs through higher programmer salaries has made conversion more difficult due to the huge investment in operational software for existing computers. The solutions to the programming transferability problems can be categorized into one of the following types:

1. Hardware emulators - In order to maintain compatability many modern computers have included hardware or micro-program features that permit them to simulate other (usually older) computers. Thus, existing programs can still be executed.

2. Software translators - A program is developed that will take programs that were written for machine X and translate them into equivalent programs for machine Y. This approach has been limited since the technique is seldom 100% successful, even when the two computers are almost identical.

3. Higher level languages - If programming is confined to high level languages, hopefully machine independent, and a translator or compiler is used to produce the actual machine code, then it should theoretically be possible to feed the same higher level source statements into a translator to another brand of computer and produce a program that performs functionally equivalent tasks. The difficulty here is whether the language and the interpretations given it by compiler writers are truly machine independent.

5.1.2 Level of Transfer

In the design of the HAL compiler system for the 360 implementation, Fortran was adopted as the output language from the code generator. A principal reason expressed for the

somewhat unusual procedure was to promote machine transferability. Fortran IV is the most widely used programming language and ANSI Fortran IV purports to be defined in a machine independent way. Production of the HAL code generator was initiated with the avowed intent of producing ANSI standard Fortran IV. If this could have been rigidly adherred to, transferability would have been automatically produced at the lowest level. The output of Pass 2 would be suitable for submission to any Fortran compiler. As it is, there exists some 360 specific Fortran output and some assembly language subroutines, but the job required to take the Fortran output of the HAL compiler and move to another computer is a minor one. Figure 5.1 depicts the steps in the HAL compilation process. The Fortran output of Pass 2 may be physically moved (in card or tape form) to another computer facility.

Contrast this transferability with the proposed system for construction of a HAL code generator for a flight computer. (See Figure 5.2) In this case, a new Pass 2 is required, the same output of Pass 1 (HALMAT) is used. This is the traditional approach. Every time that HAL is desired for a different target computer, another version of Pass 2 is required. This is a <u>mid-level</u> transfer.

However, neither of the above approaches will satisfy the needs of another general purpose computer facility. The reason is that they are only partial transfers. Although they produce code for another computer, the compiler itself still must run on the initial computer, the IBM 360 in this case. This is poor operationally. It means that a user must submit his HAL source program to the 360 for compilation and then take the object program to the other computer for execution. (This approach is perfectly adequate for a flight computer where the usual mode of operation is via simulation on his general purpose computer. Besides, the flight computer is usually of such limited size that compilation on it is not possible even if one were physically available.)

A total transfer is needed for implementation on another large commercial computer. It requires that the entire system be transferred, "lock, stock and barrel". Then the user can compile and execute on the new facility with no further need of the 360. This is a more demanding requirement since it necessitates moving the entire compiler to a different computer complex. The result is a <u>high level</u> transfer or complete conversion.

5.1.3 Method of Attack

There are three avenues of approach that might be followed to achieve a compiler transfer. They are:

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

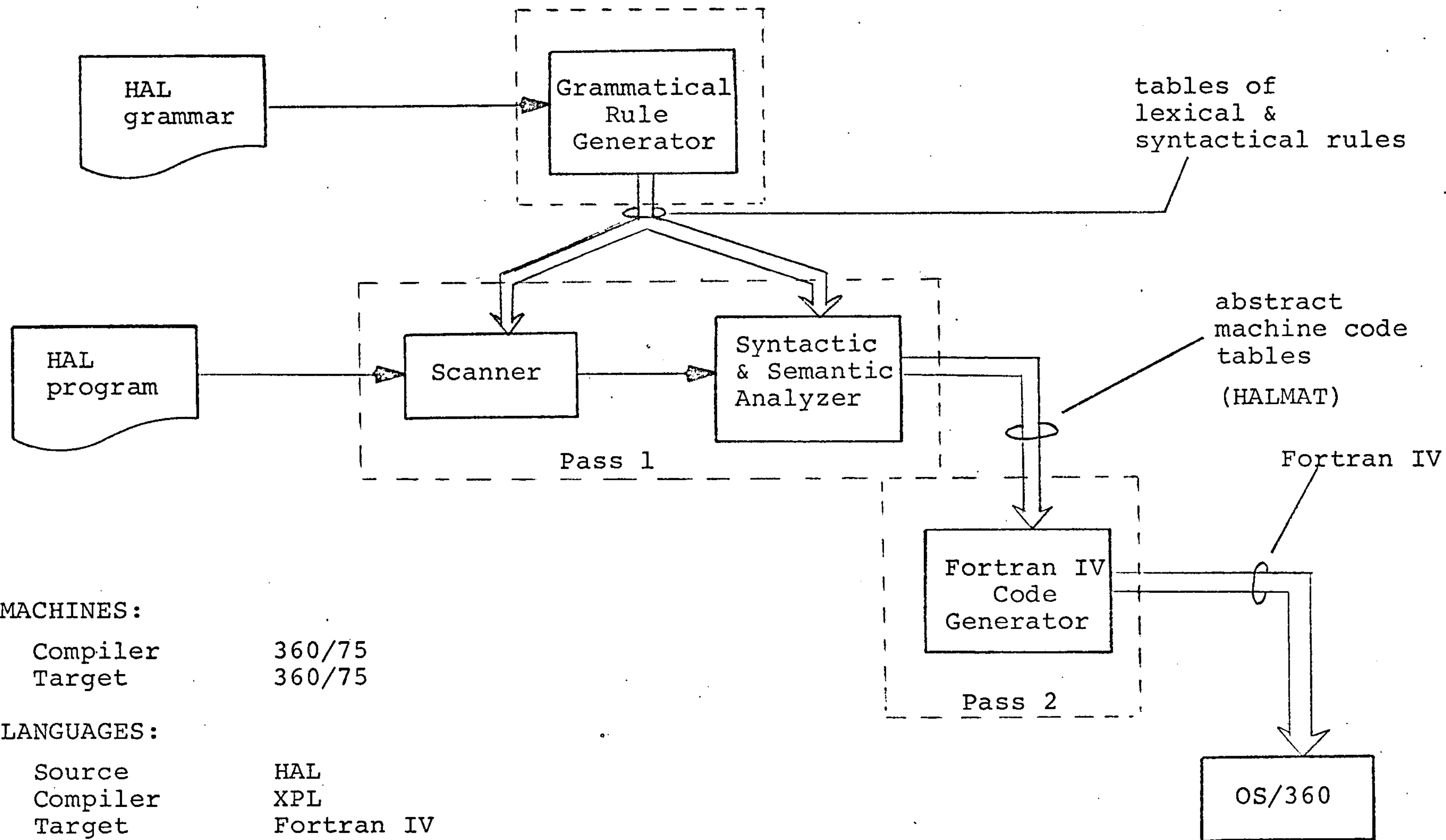


Fig. 5.1 Construction of the HAL Compiler System

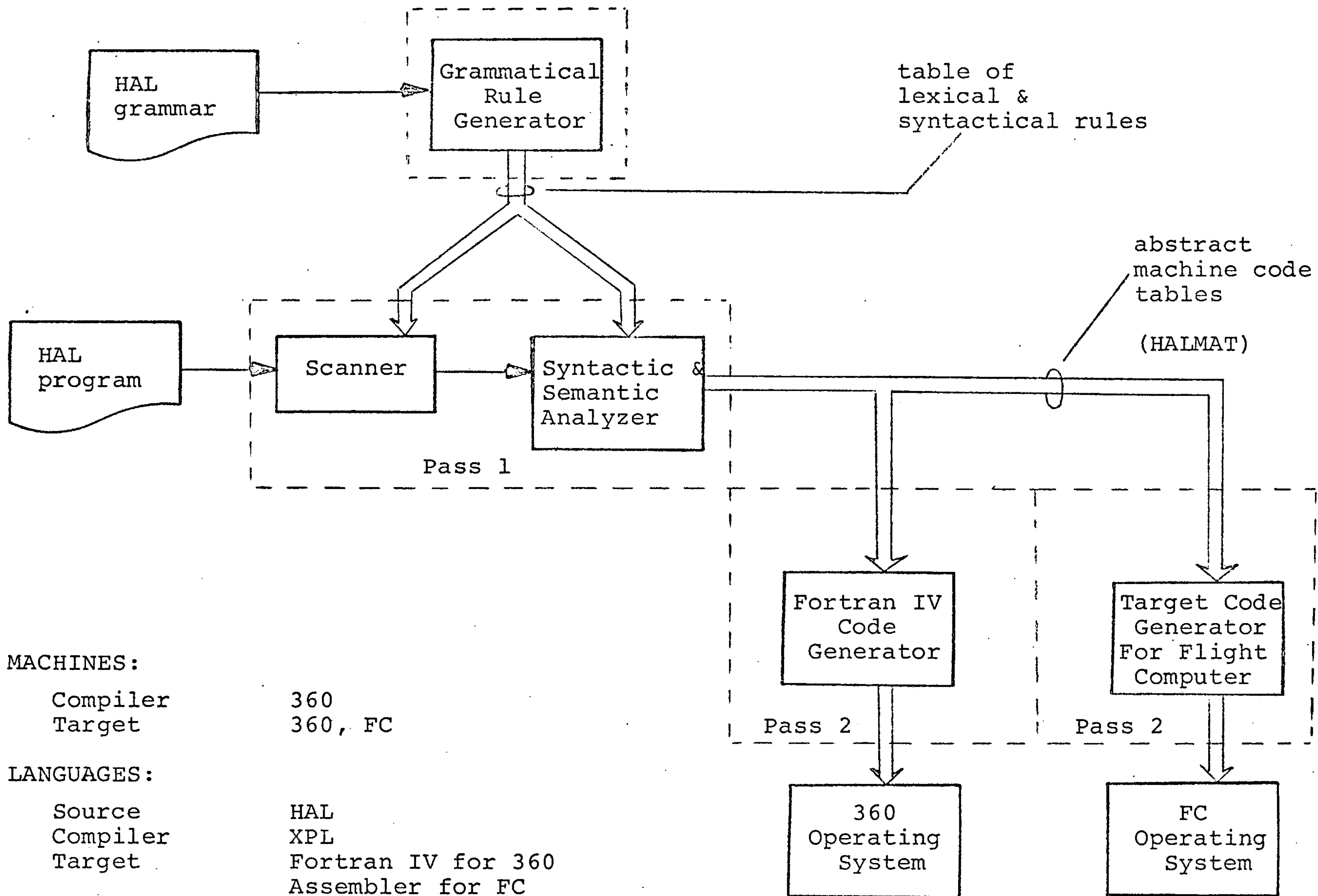


Fig. 5.2 Proposed Construction of HAL Compiler for Flight Computers

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

1. Reprogram the HAL compiler for Brand X: This technique looks at the process as a one-of-a-kind step and selects whatever seems most appropriate for machine X, be it assembly language or whatever. Then the job is done. This is the brute force approach and has no generality whatsoever.

2. Reprogram the XPL compiler for Brand X: HAL is written in XPL, a simplified subset of PL/1. Thus, it would be relatively easy to transfer to another computer that supports PL/1; however, there are few that do. But we could transfer XPL to another computer. Since XPL is itself written in XPL, the transfer could be accomplished by a mid-level transfer. (A new code generator on the 360 that produced code for Brand X would allow a version of XPL to be compiled that would execute on Brand X.) However, this approach also lacks generality; each new computer requires another code generator, itself not an easy task.

3. Reprogram HAL into a language more widely supported: If HAL could be rewritten in a language that was universally supported, than transfer problems would be minimized. The most widely used language is Fortran. And since Fortran is now produced by HAL, an interesting variation of this technique is immediately suggested. If HAL was rewritten in HAL and compiled on the current HAL compiler then the result would be Fortran source cards that would be suitable for compilation on any computer with a Fortran compiler. Thus, the transfer of HAL to almost any large scale computer could be achieved by minor changes to the Fortran output (chiefly in the area of data types and declarations) and the recoding of machine-dependent library routines. But the latter must be done anyway if HAL is to execute on Brand X; even the low-level transfer needed it. The extra task is the effort needed to rewrite HAL in HAL. But having done it once, it would not need to be done again to affect other transfers. The generality of this approach resolved the issue in its favor.

5.2 Translation of XPL Programs Into HAL

5.2.1 Introduction

This is a brief discussion of the methods used when translating a program from XPL into HAL. It is intended to provide a useful guide to a process which requires a considerable amount of analysis and judgement on the part of the individuals performing the work. This end is achieved by presenting the

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

essentials of language differences and by discussing examples of coding economies possible through the use of HAL. Explicit illustrations demonstrate the translation of several XPL constructs into HAL.

Translation of some form was necessary since it was intended that a copy of the HAL compiler be implemented on the Univac 1108. There were two general strategies available to Intermetrics as alternate means to affect this implementation. As one possibility, we could have rewritten or modified the XPL system to implement it on the 1108, then we would have been able to recompile the original XPL source code of HAL on the 1108. This approach lacks generality and involves the difficulties of emitting executable and efficient low level code for a machine with extant high level software. As a second alternative, we could rewrite the HAL compiler in a source language which maps via an existing processor onto a target language recognized by existing 1108 software.

This latter course was chosen, using HAL itself as the source language, and using the HAL/360 compiler as the mapping onto Fortran IV, a target language understood by the 1108 (as well as other large scale computers). This course procides a large amount of generality, and also proves to be easiest to carry out because of HAL's many high level features and the convenient degree of similarity between HAL and XPL.

The two-dimensional input scanner employed in Pass 1 of the HAL compiler was chosen as an initial goal. If a program as complex as this worked satisfactorily once debugged, we could be fairly certain that no part of the compiler would create a problem. The translation strategies and methods described in this document were devised in the process of successfully rewriting the input scanner. As an added bonus of this choice of translation strategy, the use of HAL as a source language proved to be exceptionally helpful in the process of debugging the current HAL/360 compiler. Quite a number of bugs which were invisible prior to this large scale application were exposed and repaired in the process.

5.2.2 Methodology

5.2.2.1 Variables. Variable declarations differ somewhat between XPL and HAL. Each individual DECLARE statement must be examined for possible changes.

BIT variables are declared identically in both XPL and HAL. The length specification is also identical in the two languages.

Ex. DECLARE X BIT(8);

FIXED variables in XPL are functionally identical to INTEGER variables in HAL. Therefore, mere substitution of the word INTEGER for FIXED is all that is necessary to make the language change. The REPLACE facility in HAL is the simplest method of substitution. Note here that any XPL variable names which correspond to HAL reserved words must be changed or augmented; i.e., the XPL identifier VECTOR could become VECTOR1 in HAL. Note also that the break characters @, #, and $ are not legal identifier break characters in HAL and XPL identifiers using them must be replaced by legal HAL identifiers.

CHARACTER variables have somewhat different properties in XPL and HAL. In XPL, character variables are implicitly varying with a maximum length of 256 characters. However, VARYING character strings in HAL are currently limited to a length of 255. Thus, the general substitution rule for character declares is to change:

```
DECLARE ALPHA CHARACTER;                              (XPL)
```

to

```
DECLARE ALPHA CHARACTER (255) VARYING;                (HAL)
```

In cases where a string is known to have a maximum length considerably less than 255 characters, it may be declared as such. Also, if a string is to be of fixed length (as with an initial unchanging value), the VARYING attribute should also be omitted.

Factored declarations in XPL and HAL are also implemented differently and involve a complete rewriting of the statements. For example:

```
DECLARE (I,J,K) FIXED, L BIT(8);                      (XPL)
```

becomes:

```
DECLARE INTEGER, I, J, K;                             (HAL)
DECLARE L BIT(8);
```

A word of caution is necessary at this point. XPL initializes all FIXED and BIT variables to "0" and all character strings to null strings unless otherwise specified by the INITIAL modifier. Any variable not explicitly initialized in HAL will have unpredictable contents. When in doubt as to whether the program itself initializes variables, include an INITIAL(0) specification

on the DECLARE statement (INITIAL('') for character strings).

The LITERALLY attribute in XPL is used to perform macro substitution for identifiers. The REPLACE statement in HAL performs the same function. The statement:

```
DECLARE FOREVER LITERALLY 'WHILE "1"';            (XPL)
```

becomes

```
REPLACE FOREVER BY 'WHILE TRUE';                  (HAL)
```

5.2.2.2 Arrays. When transferring array declarations and specifications from XPL to HAL there are a number of ground rules to follow. First, XPL subscripts start at 0 and the dimension specified is the highest allowable subscript. Therefore, an XPL array declared with an arrayness of 99 actually consists of 100 elements and must be declared as such in HAL, since all HAL subscripts start at 1 for an array. Thus,

```
DECLARE ABLE(99) FIXED;                           (XPL)
```

becomes

```
DECLARE ABLE ARRAY(100) INTEGER;                  (HAL)
```

The word ARRAY must be supplied in HAL in array declarations.

Frequently in XPL the name of an array appears without a subscript. This means an implied reference to the 0th element of the array. However, in HAL, an array name without an explicit subscript implies reference to the entire array, not the first element. Therefore, for conversion, all such occurrences of non-subscripted array names must be translated with the explicit subscript of 1. Thus, the following XPL segment:

```
DECLARE ARR (9) FIXED, B CHARACTER;

B = ARR;
```

becomes in HAL:

DECLARE ARR ARRAY (9) INTEGER,

B CHARACTER (255) VARYING;

B = ARR_1;

In general, unless the 0th element of an XPL array is known not to be used, indexing expresions must be augmented by adding one to the original subscript expression, and not by changing the computation of indices in other statements. This is especially true when array references are made using Boolean values of 0 and 1 as switches for referring to one of two array elements. Thus, the XPL sequence:

```
IX = IY + IZ;

VALUE = ABLE (IX);
```

should become in HAL:

$$IX = IY + IZ;$$

$$VALUE = ABLE_{IX + 1};$$

as opposed to the HAL sequence:

$$IX = IY + IZ + 1;$$

$$VALUE = ABLE_{IX};$$

as the latter form could possibly change the operational characteristics of the program.

Finally, XPL allows the specification of a subscript on a variable which is not declared as an array. This allows certain machine dependent coding "tricks" to be performed. Consider the following XPL sequence:

```
DECLARE INDEX FIXED, INDEXTAB (199) BIT (8);

    DO I = 1 TO 50;

        INDEX (I) = 0;

        END;
```

This program in effect zero's out INDEXTAB with 50 references, rather than the 200 required to clear the individual INDEXTAB elements. This sequence is illegal in HAL and may be coded as follows in HAL:

```
INDEXTAB = 0;
```

where the non-subscripted version of the name implies setting the array to zero.

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

An extra step is involved when translating statements utilizing arrays of character strings or bit strings. A colon (:) must follow the array element subscript, to distinguish it from the individual character or bit subscript which is the default in HAL for these types of variables. It may be used following any array subscript, but is required in the above named instances to prevent ambiguity.

Ex:

```
DECLARE A CHARACTER(5),
        B ARRAY(10) CHARACTER(10);
```

$B_{1\ TO\ 5} = A;$

$B_{1\ TO\ 5:} = A;$

In the first statement, characters 1 to 5 of all ten array elements of B are set to the value of the characters in A; in the second statement, the first five array elements of B are set to the value of A (padded with blanks to make the total length ten).

5.2.2.3 Built-in Functions. The XPL functions ADDR and INLINE are not available in HAL, and because of the machine independence of the language no corresponding functions exist. In the HAL compiler, fortunately, most such functions are used to manipulate data types not existing in XPL but which do exist in HAL.

The SUBSTR and BYTE functions in XPL are replaced with character string subscript notation in HAL. Examples of both forms of BYTE and SUBSTR substitution follow:

a. BYTE (CHAR_STRING) (XPL)

becomes

$CHAR_STRING_{1}$ (HAL)

b. BYTE (CHAR_STRING, N) (XPL)

becomes

$CHAR_STRING_{N+1}$ (HAL)

c. SUBSTR (CHAR_STRING, START) (XPL)

becomes

$CHAR_STRING_{START+1\ \ TO\ \#}$ (HAL)

d. SUBSTR (CHAR_STRING, START, N) (XPL)

becomes

$CHAR_STRING_{N\ AT\ START+1}$ (HAL)

The functions SHL and SHR are used for doing word manipulation in XPL. For positive arguments, the SHL function may be replaced by multiplication by the appropriate power of two. The SHR function is more complex as integer division is not allowed in HAL. Since SHR is normally used to isolate a field of a packed word, the BIT shaping function can be used to achieve the same results.

For example, the XPL sequence:

```
DECLARE (ENTRY, PART) BIT (16),
        WORD FIXED;
  PART = SHR (ENTRY, 4);
  PART = SHR (WORD, 16);
```

becomes in HAL:

```
DECLARE BIT (16), ENTRY, PART;
DECLARE WORD INTEGER;
```

PART = $ENTRY_{1\ TO\ 12}$;

PART = $BIT_{1\ TO\ 16}$ (WORD);

In XPL, it is legal to assign the result of a relational expression to a BIT type variable. This is illegal in HAL. Thus, the statement

TEST = A > B; (XPL)

must become:

IF A > B THEN TEST = TRUE; ELSE TEST = FALSE; (HAL)

5.2.2.4 Constants. The following constant conversion rules apply:

"ABDF" or "(4)ABDF"	becomes	HEX'ABDF'
"(1)11010"	becomes	BIN'11010'
"(3)70346"	becomes	OCT'70346'

The quartal constant "(2)20312" must be converted to either BIN'1000110110' or HEX'236'.

The use of the BYTE function, notably BYTE('C'), to allow use of the internal representation of the character as a numeric quantity is accomplished in HAL by stating BIT('C'), or INTEGER (BIT('C')) where implicit Bit-to-Integer conversion may not take place.

5.2.2.5 Procedures. In XPL, all subroutines and functions are declared as PROCEDURE's. The RETURN statement may or may not pass back a value. If an XPL PROCEDURE which returns a value is called by the CALL statement, the returned value is ignored. In HAL, there are two classes of routines: PROCEDURE's and FUNCTION's. A PROCEDURE does not allow a value to be returned in the RETURN statement, whereas a FUNCTION demands that a value be returned. Thus, XPL PROCEDURE's that return values must be declared as FUNCTION's in HAL. Any such FUNCTIONS invoked by the CALL statement in XPL must be changed to the form:

```
DUMMY_VARIABLE = FUNCTION_NAME(X);
```

where the dummy variable is some unused name in the HAL program with the mode of the called function.

Also, in XPL, all formal parameters are call-by-value parameters. This presents a problem in HAL because, 1) PROCEDURE and FUNCTION parameters may not be assigned values within HAL programs, unlike XPL which freely allows such assignments, 2) the alternative in HAL, the ASSIGN list, is treated as a list of call-by-reference parameters, where assignments to such parameters are passed back to the calling program, whereas in XPL, parameter assignments do not reflect back to the calling program. Therefore, in all FUNCTIONS and PROCEDURES where assignments to formal parameters are made, a procedure prologue must be coded to assign the formal parameter (with an augmented name) to a local variable with the same declared properties with the original parameter name. Thus, the following XPL program segment:

```
ALPHA: PROCEDURE (BETA, GAMMA);
    DECLARE BETA FIXED, GAMMA CHARACTER;
          .
          .     BETA = BETA + 1;
          .
    END ALPHA;
```

becomes in HAL:

```
ALPHA: PROCEDURE (BETA_PRIME, GAMMA);
    DECLARE INTEGER, BETA, BETA_PRIME;
    DECLARE GAMMA CHARACTER (255) VARYING;
    BETA = BETA_PRIME;
          .
          .     BETA = BETA + 1;
          .
    CLOSE ALPHA;
```

when BETA is used as an assigned variable in the procedure, whereas GAMMA is not. The HAL compiler itself can be used to detect such occurrences, since assignments to parameters will be flagged as errors, significantly reducing the amount of program scanning necessary.

Note: Notice that the word CLOSE was used on the last line of the sample rather than the standard END. When closing a function or procedure in HAL, the word CLOSE is substituted for END. END is only used to signify the end of the DO loop or a DO case.

5.2.2.6 DO Statements. The DO case statement in HAL is similar to that in XPL, the only difference being that the first group of statements are executed when the DO case argument is equal to 0 in XPL and the first group of statements in HAL are executed when this argument is equal to 1. The following XPL sequence:

```
DO CASE I;
DO;/*CASE0*/
    ...
END;/*CASE0*/
    ...
END;/*OF DO CASE*/
```

translates to the following HAL sequence:

```
DO CASE I + 1;
DO;/*CASE1*/
    ...
END;/*OF CASE 1*/
    ...
END;/*OF DO CASE*/
```

When translating a DO case group from XPL to HAL, 1 must be added to the argument of the DO case statement rather than to change the value of the variable itself. DO case statements are translated in this manner to preclude the possibility of causing errors elsewhere in the compiler. It is not really possible to be certain that the change of the variable's value might not cause problems elsewhere. The looping statement:

```
DO IX = 1 to 10;                    (XPL)
```

simply becomes:

```
DO FOR IX = 1 to 10;                (HAL)
```

The word FOR is required to distinguish this type of DO statement from the DO CASE or DO WHILE statements.

5.2.2.7 INPUT/OUTPUT. The primary input/output statements in XPL are the INPUT and OUTPUT pseudo-variables. To read a card image, the following statement is used:

```
CARD_IMAGE = INPUT;
```

Similarly, to write a line the following statement is used:

```
OUTPUT = NEXT_OUTPUT_LINE;
```

Both pseudo-variables are character string type and imply a new input/output record on each occurrence.

The corresponding HAL statements to read the same card image and print the same line are as follows:

```
READALL(5) CARD_IMAGE;

WRITE(6) NEXT_OUTPUT_LINE;
```

Note that READALL, not READ, is used for input, as this forces reading an entire card image. READ into a character variable stops at any legal input delimiter.

5.2.2.8 Format of HAL File. The format of the input file is for the most part free of conventions. The only exception to this is that Column 1 may only be used to contain special letters. The following letters may appear in Column 1: C, D, E, M, and S.

These letters specify what type of line is contained on that current image. The letter C is to specify that the following text is to be treated as a comment and not actually compiled. D is used only for special compiler directives such as an INCLUDE file specified on this line. The letter E constitutes an exponent line which is part of the multi-line input format which HAL offers. M specifies that the following is the main line of the multi-line input, and S specifies a subscript line again which is part of the multi-line input. When using the single line format of HAL input, the M may be omitted from the line as long as text begins in Column 2 or after. The letter M is assumed on all lines which do not contain a character in Column 1. The above exception is the only one which pertains to the format of a HAL program.

Long and complicated HAL statements may be continued over as many cards as necessary just as in XPL. Certain equations which are broken up into several steps in the XPL version may be condensed into one large equation in HAL, resulting in a savings of temporary variables. (This is because XPL limits the number of expression temporaries in a statement to three registers. HAL has no such restriction.)

5.2.3 Debugging

The debugging procedure can be made quite simple by the use of various options which may be specified when compiling a HAL program. One may specify toggles on comment lines in HAL, which produce an identifier trace, a listing of the HALMAT code produced, and a list of the Fortran produced from Phase 2.

When errors occur, it is easy to trace the problem by consulting the three listings as mentioned above. Also, a check of the cross-reference listing produced greatly speeds debugging time since it is possible to determine in which statement a variable is either referenced, declared, or set. When a new section of code is added, a toggle can be set in a comment line at the beginning of the HAL program, which disables the call to Phase 2 of the HAL compiler. This is done to save computer time, since Phase 1 could perform a syntax check. When all syntax errors are eliminated, Phase 2 could then be called and Fortran output could be produced and subsequently compiled by the Fortran compiler.

It is, of course, much easier to debug a higher level language program than to debug assembly code, since ideas are clearly specified by the code being read, whereas in assembly language the intent is not always quite clear. In fact, when translating the in-line code it was sometimes necessary to speak to the person who had originally coded that section before a clear understanding could be gotten in order that the translation could be performed.

5.2.4 Conclusions

At the time of this writing, some HAL features are still unimplemented. Because of this, certain sections of the translated code have as of now not been tried or debugged. However, that code which has been debugged and executed seems to prove that HAL is a language with which a large compiler can be easily written and debugged. The fact that HAL implements floating point arithmetic also eliminated a great deal of the complicated code necessary in the original XPL version. This fact alone made readability of the final copy much easier than the complicated in-line code which appears in the corresponding sections of the original copy.

5.3 Feasibility Demonstration ("HAL-in-HAL")

5.3.1 Objective

The HAL-in-HAL program was written as an experiment to prove whether or not the HAL language was suitable for writing translator systems, as well as aerospace applications. The program consists of a rewrite of the two-dimensional read routines originally coded in XPL for the HAL/360 compiler, utilizing the conversion techniques outlined in Section 5.2 above. These routines represent a full test of the character and bit manipulation facilities normally required for translator and system coding.

5.3.2 Test Program Description

The test program consists of an elementary scanning routine which utilizes the STREAM procedure for receiving its character-by-character input. STREAM converts the two-dimensional HAL input cards into the corresponding one-line format which is required by the scanner and subsequently the lexical analyzer. The test scanner repeatedly calls STREAM building-up identifier and numeric strings as tokens, as well as treating any special character as an automatic token. These are printed out as they are encountered. The test scanner is concurrently building-up an output line image which is a reflection of the input character received from STREAM. Whenever a semi-colon (;) is encountered, the current statement line, along with its corresponding over-punch markers, is printed, showing what the one-line format of the HAL statements looks like. A question mark (?) is used to indicate the end of the input stream for the purposes of this test. See Figure 5.3 for a flow chart of the test scanner. (Program listing - Statements 544-582 in Appendix B.)

5.3.3 Results

The HAL-in-HAL experiment has proved conclusively that HAL can be used successfully as a compiler implementation tool. Although HAL has no machine dependent features, (which frequently are designed into implementation languages), this experiment has proved that such features are not a requirement for compiler implementation, but rather merely a convenience item to circumvent known code generation inadequacies in the compiler. The HAL implementation is concise, readily followed, and understandable (even more so than the XPL version of the same program).

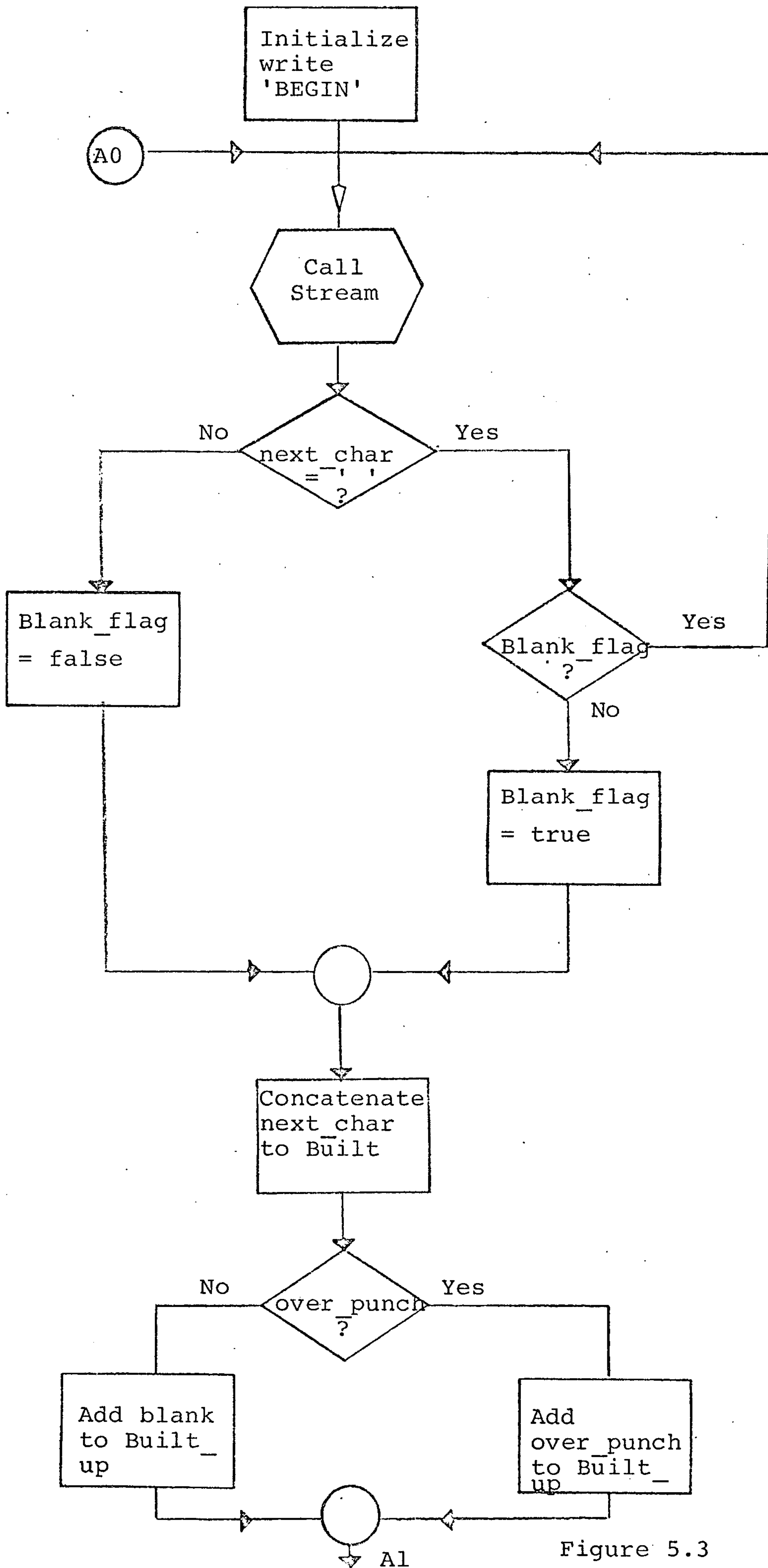


Figure 5.3

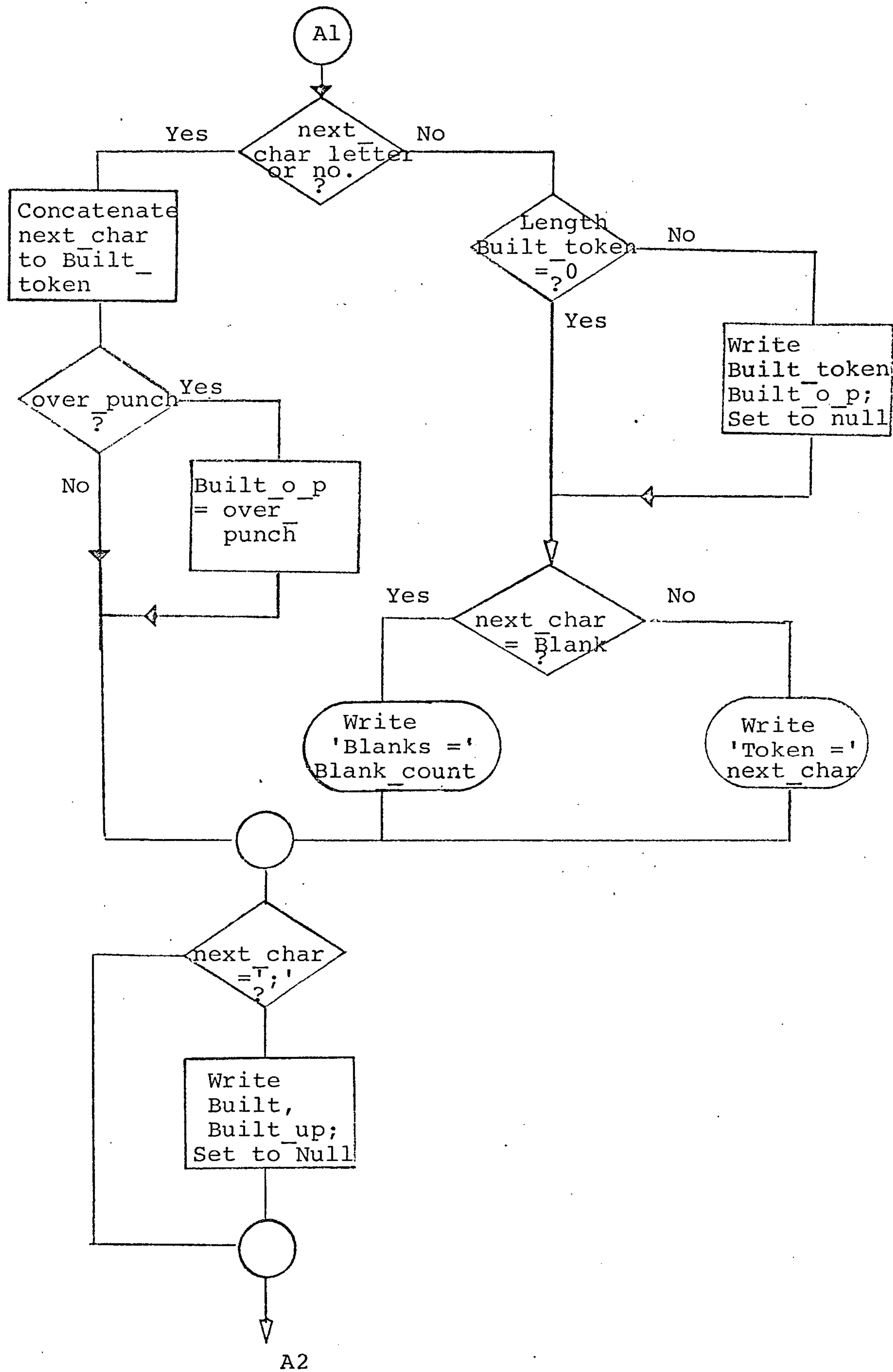


Figure 5.3 (Cont.)

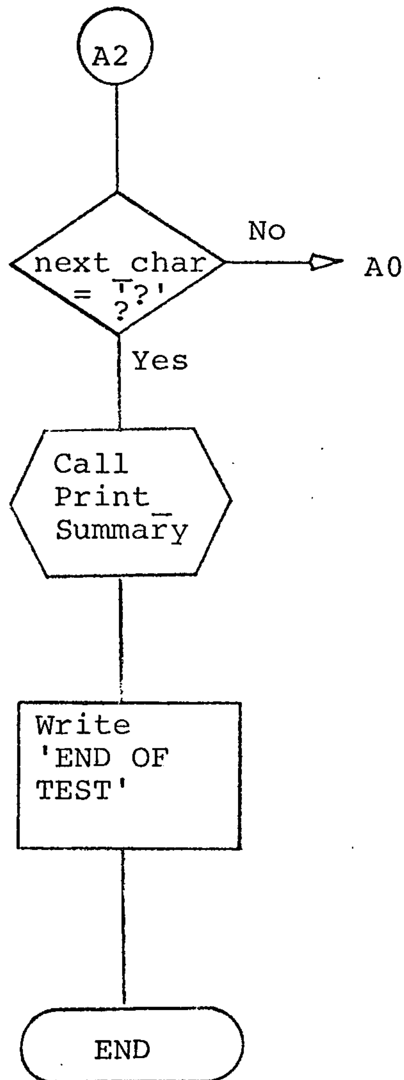


Figure 5.3 (continued)

The checking facilities of the HAL compiler can detect numerous logical errors without having to impose the strict definition rules of XPL. Unitialized variables are easily detected, as are parameter mis-matches. The bulk of the debugging time for HAL-in-HAL was in streamlining the program to make the HAL version more readable, as well as more efficient, since the rule of adding one to all XPL subscripts as a general rule turned out to be both awkward and confusing in many instances. The final version of STREAM is much more efficient than the original translation performed utilizing the rules of Section 5.2. Programs originally coded in HAL will obviously not experience this problem.

PRECEDING PAGE BLANK NOT FILMED

APPENDIX A.

HAL Course Material

PRECEDING PAGE BLANK NOT FILMED

PRECEDING PAGE BLANK NOT FILMED

A.1 OVERVIEW

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

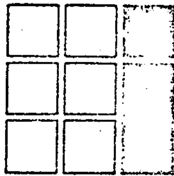
PREFACE

* HAL developed by Intermetrics, Inc.
  * Language design
  * Compiler design and implementation

* Significant Objectives
  * Increased readability
  * Increased reliability
  * Real time control

* Capabilities
  * Primarily designed for on-board computer
  * General enough for:

    ground support and verification
    other real-time applications



INTERMETRICS

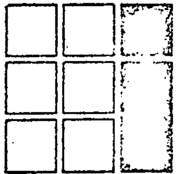
SLIDE 1

SHUTTLE LANGUAGE REQUIREMENTS

* Software Applications
  * Navigation, guidance, targeting
  * Vehicle control
  * Operating systems
  * Data management
  * Communications and displays
  * Support software
  * On-board checkout and monitor

* Computer Environment
  * Wide range of computers (Flight and Ground)
  * Fixed- and floating-point
  * Simplex, multi-computer, multi-processor

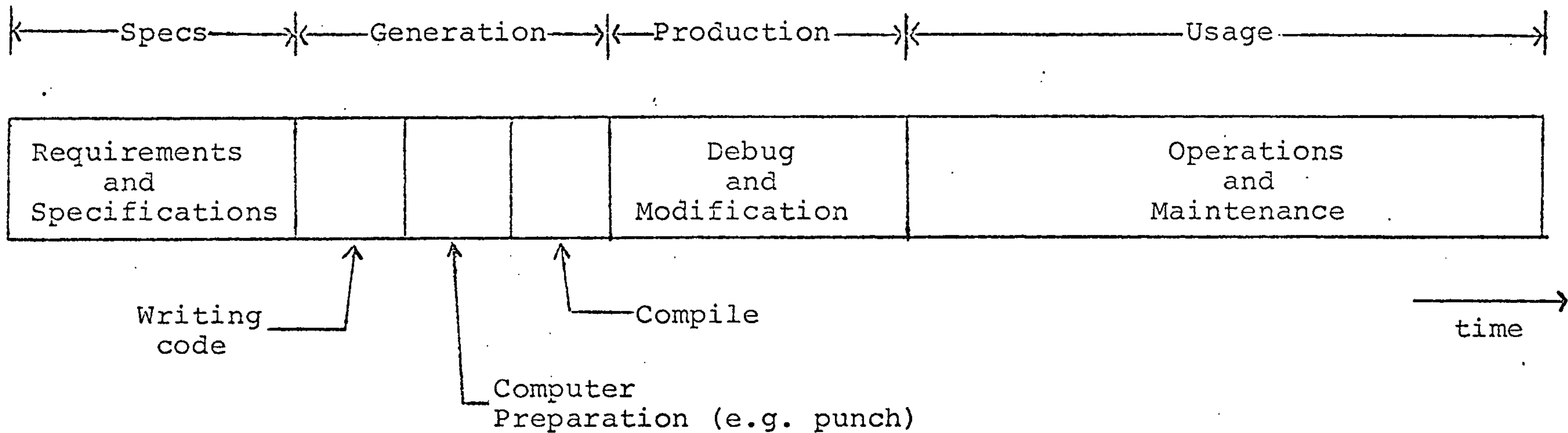
* Language Characteristics
  * Clarity and readability
  * Enforcement of standards and conventions
  * Extensive automatic checking (compile- and run-time)
  * Facilitate software management
  * Promote modularization



INTERMETRICS

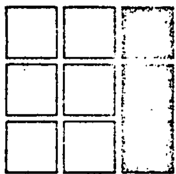
SLIDE 2

CHRONOLOGY OF SOFTWARE DEVELOPMENT



Some Observations

1. The writing of code is closely tied to the specifications.
2. The time required for computer preparation is small compared to the program life.
3. A lengthy period of debug and modification must be provided.
4. Period of program usage extends many times that of program generation.
5. Many more people will use a program than generated it.



Conclusion

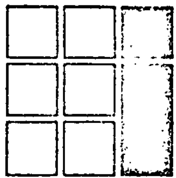
The computer language should promote understanding of the software. The listing should tend toward self-documentation.

INTERMETRICS

SLIDE 3

SALIENT FEATURES OF HAL

Capability	Requirement
1. Two-dimensional Input-Output Annotation of variables	* Increased readability
2. Complete vector-matrix arithmetic	* Targeting, guidance and control
3. Data array and structure handling	* Data management
4. Bit and character manipulations	* Systems, communications and I/O
5. Real-time control statements	* Command and control
6. Data-Pool (COMPOOL), controlled sharing and name scope	* Increased reliability



INTERMETRICS

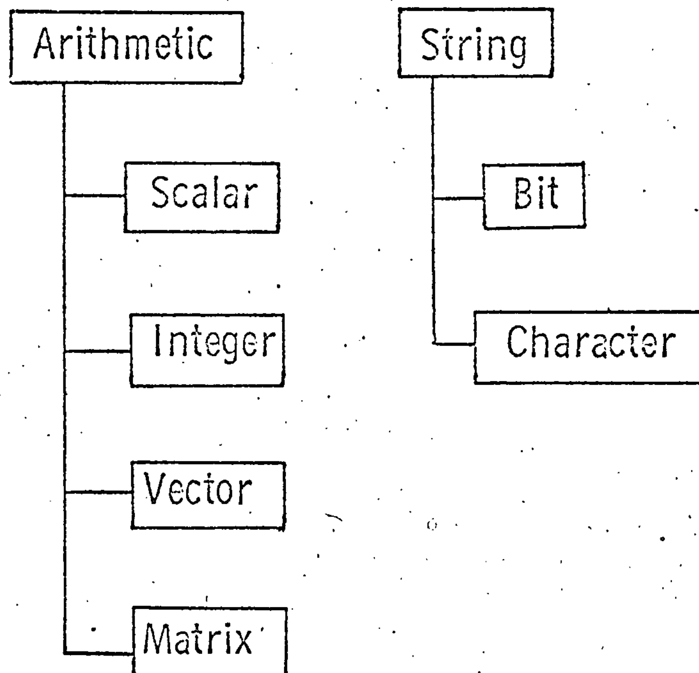
SLIDE 4

ADVANCED FEATURES

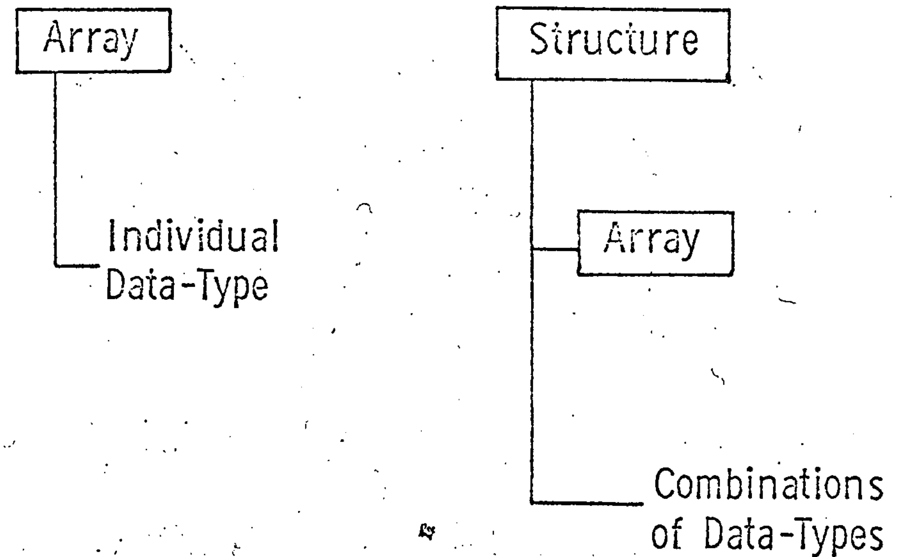
- TWO-DIMENSIONAL (MULTI-LINE) INPUT AND OUTPUT
- VECTOR AND MATRIX DATA TYPES AND OPERATORS
- TASK SCHEDULING AND SYNCHRONIZATION STATEMENTS FOR REAL-TIME CONTROL
- CONTROLLED SHARING OF DATA AMONG MULTIPLE USERS THROUGH A COMPOOL AND DATA LOCKING STATEMENTS
- STATEMENTS TO MANIPULATE DATA-GROUPS (ARRAYS AND STRUCTURES) AND POWERFUL METHODS TO PARTITION AND INDEX THEM
- OUTPUT ORIENTED. LANGUAGE IS SLANTED TOWARDS PRODUCTION OF UNDERSTANDABLE AND UNAMBIGUOUS OUTPUT LISTING RATHER THAN MINIMIZING KEYSTROKES ON INPUT
- A SIMPLE SCIENTIFIC SUBSET IS DEFINED AT THE OUTSET THAT WILL PERMIT EASY USE BY THOSE WITH A SCIENTIFIC BACKGROUND

HAL Data Types and Organizations

Types



Organizations



Unique notation: VECTOR:$\overline{\text{NAME}}$, BIT STRING:$\dot{\text{NAME}}$, CHARACTER STRING:$\overset{,}{\text{NAME}}$, MATRIX:$\overset{*}{\text{NAME}}$, ARRAY:[NAME], STRUCTURE:{NAME}

EXAMPLES OF DATA TYPES

SCALAR: -126.04

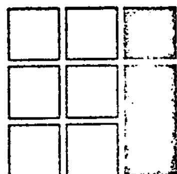
INTEGER: 126

VECTOR: 5, -26.4, 3.061

MATRIX: 5, -26.4, 3.061,
1, -67.2, 106.1,
0, 73.29, 0.06

BIT STRING: 1 0 1 1 0 1 0 1

CHARACTER STRING: VOLTAGE ON BATTERY B 2 VOLTS BELOW SPEC

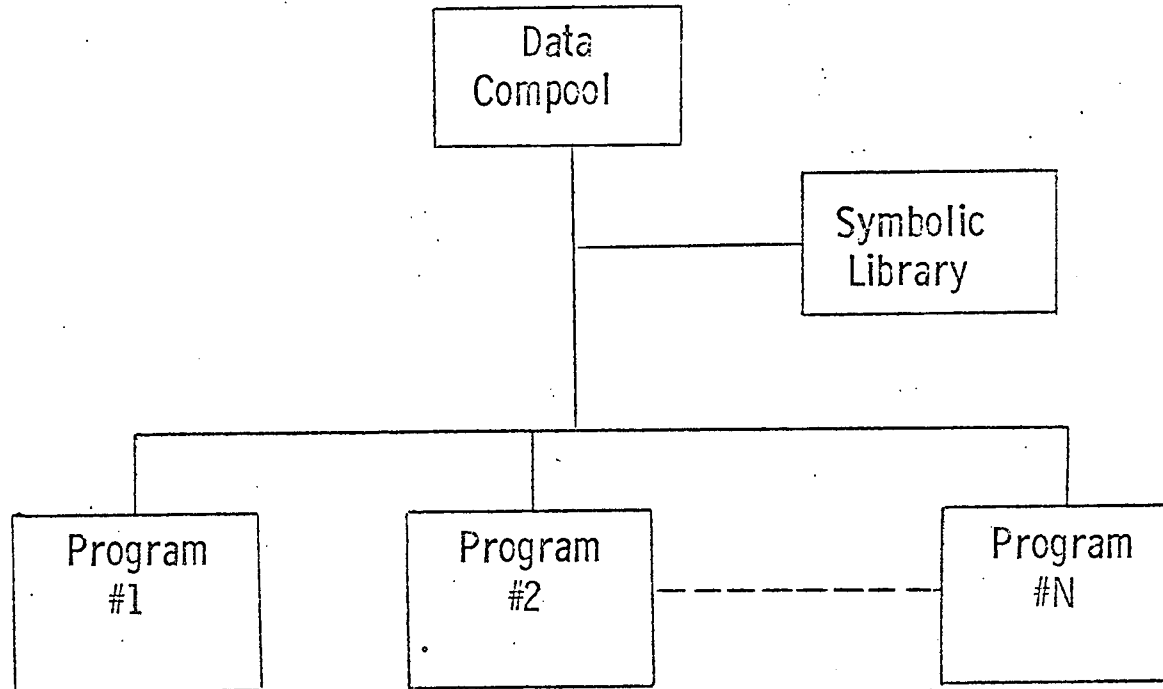


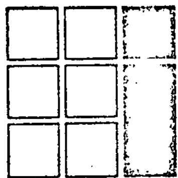
INTERMETRICS

STRUCTURE ORGANIZATION OF DATA

```
DECLARE 1 NAV_STATE (2);
        2 STATE (2),
            3 TIME PRECISION (8),
            3 R VECTOR PRECISION (10),
            3 V VECTOR PRECISION (10),
        2 STATE_FLAGS,
            3 BODY_FLAG BIT INITIAL (TRUE),
            3 PHASE_FLAG BIT,
        2 W MATRIX (9, 9) PRECISION (10);
```

HAL PROGRAM ORGANIZATION



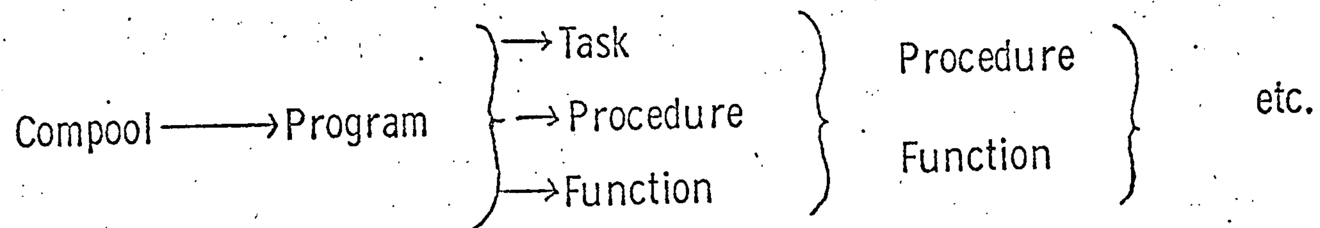


INTERMETRICS

SLIDE 6

SCOPE OF NAMES

- Scope is the region in which a name is recognized.
- Scopes are defined from the outermost block toward the inner; i.e.,

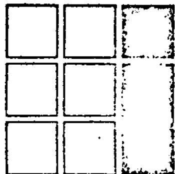


- Names defined in an inner block are <u>never</u> recognized in an outer block. Inner blocks effectively isolate locally defined variables.

BLOCKS OF CODE (NAME SCOPE)

```
ABLE:    PROGRAM;
           DECLARE VECTOR (5) A,B,C;           ---> A,B,C are vectors (5)
           A = B + C;
             .
             .
             .
         BAKER:  TASK;
                 DECLARE A INTEGER;            ---> B,C are vectors (5)
                                                    A is now an integer

                 CHARLIE:  PROCEDURE;
                           DECLARE X;          ---> B,C are vectors (5)
                           DECLARE A BIT (10);      A is now a bit string
                             .                      X is a scalar
                             .
                             .
                           END CHARLIE;
                 END BAKER;
         GRAB:   PROCEDURE;
                 DECLARE X VECTOR (4);         ---> A,B,C are vectors (5)
                   .                                X is a vector (4)
                   .
                   .
                 END GRAB;

         END ABLE;
```

(In $\overline{A} = \overline{B} + \overline{C}$, A, B and C are overbarred as vectors.)



INTERMETRICS

SLIDE 7

HAL Statements

1. Assignment

   LABEL:VARIABLE = EXPRESSION;

2. Declare

   DECLARE --------

3. Control

   GO TO ----, IF-statements, DO-statements,

4. Block

   Procedures, Functions, Tasks, Updates, Programs

5. Real-time Control

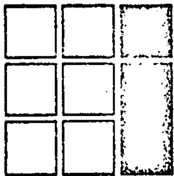
   Schedules, Waits, Signals, Locks

EXAMPLES OF ARITHMETIC OPERATIONS

(From Apollo Navigation Equations)

HAL	GSOP Specification
$\overline{Z} = \overset{*}{W}^{T}\ \overline{B};$	$\underline{Z} = W'^{T}\ \underline{b}$
$OM\overline{E}GA = \overline{Z}\ \overset{*}{W}^{T}/(ZMAG^{2} + ALPHA^{2});$	$\underline{\omega}^{T} = \dfrac{1}{Z^{2} + \bar{\alpha}^{2}}\ \underline{Z}^{T}\ W'^{T}$
$D\overline{E}LX = O\overline{M}EGA\ DELQ;$	$\delta\underline{X} = \underline{\omega}\ \delta Q$
$\overline{X} = \overline{X} + D\overline{E}LX;$	$\underline{X} = \underline{X}' + \delta\underline{X}$
$F = 1 + (ALPHA^{2}/(ZMAG^{2} + APLHA^{2}))^{1/2};$	$W = W' - \dfrac{\underline{\omega}\ \underline{Z}^{T}}{1 + \sqrt{\dfrac{\bar{\alpha}^{2}}{Z^{2} + \bar{\alpha}^{2}}}}$
$\overset{*}{W} = \overset{*}{W} - OM\overline{E}GA\ \overline{Z}/F;$	

where $\underline{b}$ ≡ geometry vector
W ≡ square root of covariance
$\bar{\alpha}^{2}$ ≡ measurement variance
$\underline{X}$ ≡ state vector



INTERMETRICS

SLIDE 9

CONTROL, LOGIC AND COMPUTATION

(Cross product steering of Apollo vehicle)

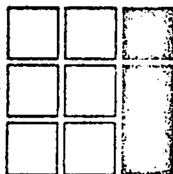
Involves scalars, 3-d vectors, 3x3 matrices, "Booleans"

```
XSTEER:  IF TGO < 4 THEN DO;
                   OMEGA_CNB = 0;
                   SW = OFF;
                   SCHEDULE ENGINE_OFF AT (TIME+TGO)
                            PRIORITY (20)  E_OFF_ID;
                   GO TO START;
                      END;
DELM = C B DELT - DELV;
OMEGA_C = K(VG*DELM)/(ABVAL (VG) ABVAL (DELM));
OMEGA_CNB = SMNB REFSMMAT OMEGA_C;
GO TO START;
```

where TGO ≡ "time-to-go"

VG ≡ "velocity-to-be-gained"

OMEGA_ ≡ rate command



INTERMETRICS

SLIDE 10

EXAMPLES OF MATRIX PARTITIONING

Given: 9x9 covariance matrix E of errors in position, velocity and landmark location. That is,

$$\overset{*}{E} = \begin{bmatrix} \overset{*}{E}_{p-p} & \overset{*}{E}_{p-v} & \overset{*}{E}_{p-\ell} \\ \overset{*}{E}_{v-p} & \overset{*}{E}_{v-v} & \overset{*}{E}_{v-\ell} \\ \overset{*}{E}_{\ell-p} & \overset{*}{E}_{\ell-v} & \overset{*}{E}_{\ell-\ell} \end{bmatrix}$$

1. RMS Errors

$$\text{RMS_POS} = \text{SQRT}(\text{TRACE}(\overset{*}{E}_{1 \text{ TO } 3,\ 1 \text{ TO } 3}));$$

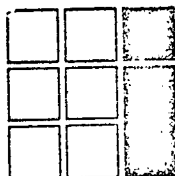
$$\text{RMS_VEL} = \text{SQRT}(\text{TRACE}(\overset{*}{E}_{4 \text{ TO } 6,\ 4 \text{ TO } 6}));$$

2. Initialize $\overset{*}{E}$ for new landmark

$$\overset{*}{E}_{1 \text{ TO } 6,\ 7 \text{ TO } 9} = 0;$$

$$\overset{*}{E}_{7 \text{ TO } 9,\ 1 \text{ TO } 6} = 0;$$

$$\overset{*}{E}_{7 \text{ TO } 9,\ 7 \text{ TO } 9} = \text{MATRIX}_{3,3}\ (A^2,\ 0,\ 0 \\ 0,\ B^2,\ 0 \\ 0,\ 0,\ C^2);$$



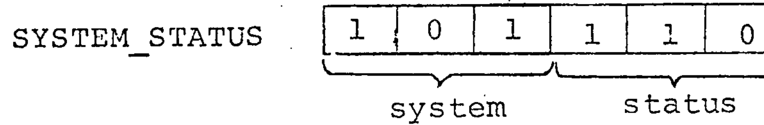
INTERMETRICS

SLIDE 11

BIT AND CHARACTER MANIPULATIONS

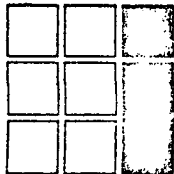
Suppose the system-status word is made up as follows:

SYSTEM_STATUS



Example:

```
              A = 'SYSTEM STATUS:';
DECODE:       DO CASE SYSTEM_STATUS 1 TO 3;

              MESSAGE = 'ENGINE'||A;              CASE 1
              MESSAGE = 'POWER'||A;               CASE 2
              MESSAGE = 'IMU'||A;                 CASE 3
              MESSAGE = 'LIFE_SUPPORT'||A;        CASE 4
                 *                                  *
                 *                                  *
                 *                                  *
                 END;
              DO CASE SYSTEM_STATUS 4 TO 6;

              MESSAGE = MESSAGE||'O.K.';                CASE 1
              MESSAGE = MESSAGE||'RECONFIGURED';        CASE 2
              MESSAGE = MESSAGE||'IN SELF-CHECK';       CASE 3
                 *                                        *
                 *                                        *
                 *                                        *
                 END;

END_DECODE:   WRITE(DISPLAY)MESSAGE;
```



INTERMETRICS

SLIDE 12

EXAMPLE OF IF-STATEMENTS

(Flag-checking in Apollo Rendezvous Data Processing)

```
A:  WAIT FOR SYNCH_SIGNAL;

    IF REFSMMAT_FLAG THEN

       IF  R_60_OP THEN GO TO A;

       ELSE IF UPDATE_FLAG  THEN DO;

                                    IF VHF_RANGE THEN

                                      IF TIME>60-TIME_VHF THEN GO TO VHFREAD;

                                    GO TO D;

                                      END;

       ELSE IF TRACKFLAG THEN GO TO D;

    GO TO EXIT;
```

Note: ELSE always refers to immediately preceding IF (except when IF is within a DO group)

EXAMPLE OF A PROCEDURE

(The Apollo Time-Radius Routine
from GSOP)

CALLER:

CALL TIME_RADIUS ($\overline{RT2}$, $\overline{VT2}$, (ABVAL ($\overline{RT2}$) - 30480), MU_EARTH,

$\dot{T}$_R_FLAG) ASSIGN (TIME_32, $\overline{RT3}$, $\overline{VT3}$);

SUBROUTINE:

TIME_RADIUS: PROCEDURE ($\overline{A}$,$\overline{B}$,C,D,$\dot{E}$) ASSIGN (F,$\overline{G}$,$\overline{H}$);

[Statements
: :
: :
: :]

RETURN;

END TIME_RADIUS;

NOTE: "Call-By-Name", "Call-By-Value"

<u>EXAMPLE OF A FUNCTION</u>

ABLE: N = TRACER($\overset{*}{A}+\overset{*}{B}$);

GO TO BAKER;

TRACER: FUNCTION($\overset{*}{Q}$);

DECLARE Q MATRIX(A,*);

IF TRACE($\overset{*}{Q}$) > 100 THEN

RETURN ($\overset{*}{Q}\ \overset{*}{Q}^{-1} + \overset{*}{Q} + \overset{**}{QQ} + \overset{***}{QQQ}$)

ELSE RETURN(0);

END TRACER;

NOTE: "CALL BY VALUE", "run-time" dimensions

PROGRAMMING REQUIREMENTS FOR REAL TIME SPACE APPLICATIONS

Scheduling and Tasking

Software performs time critical functions and responds to interrupts in a complex environment requiring the capability to schedule, control and synchronize tasks.

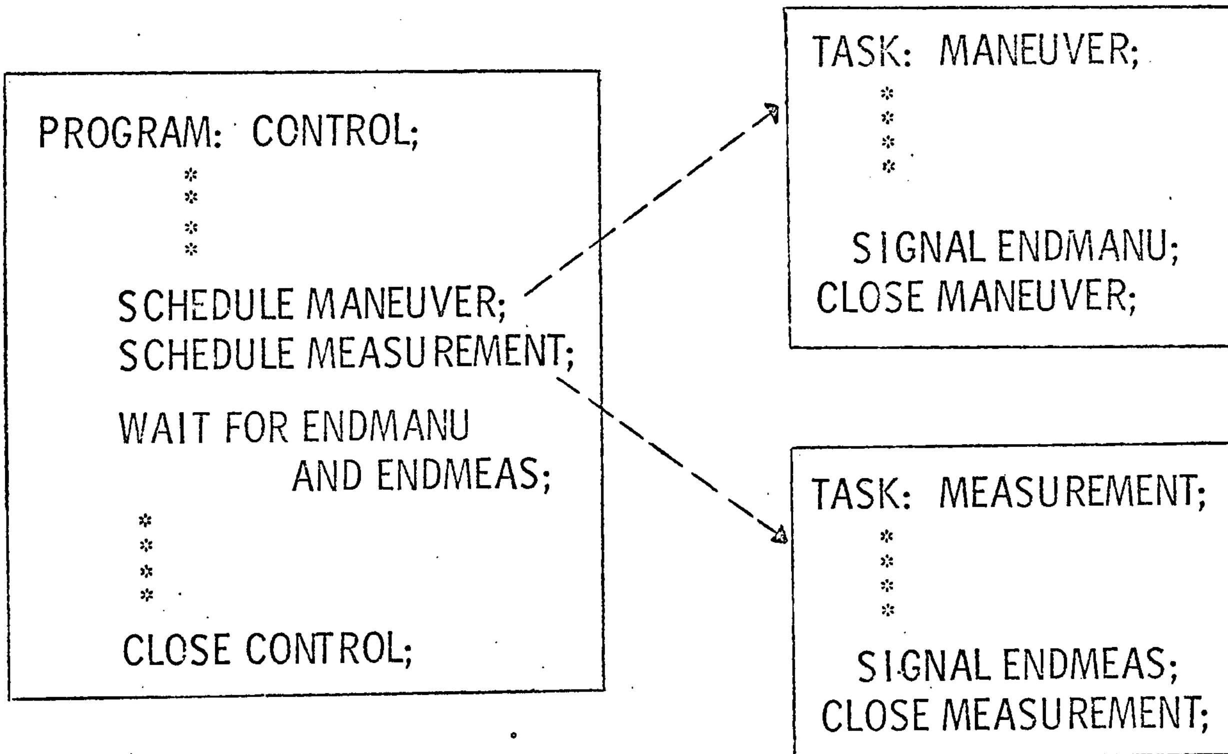
Recovery From Error Conditions

Techniques are required to protect and enable system to "continue" after detection of unexpected error condition.

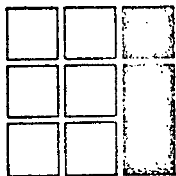
Common Memory Sharing and Control

Techniques are required to dynamically control the use of common data elements among tasks in the environment.

EXAMPLE OF HAL REAL-TIME CONTROL



ENDMANU and ENDMEAS are programmer-defined events



INTERMETRICS

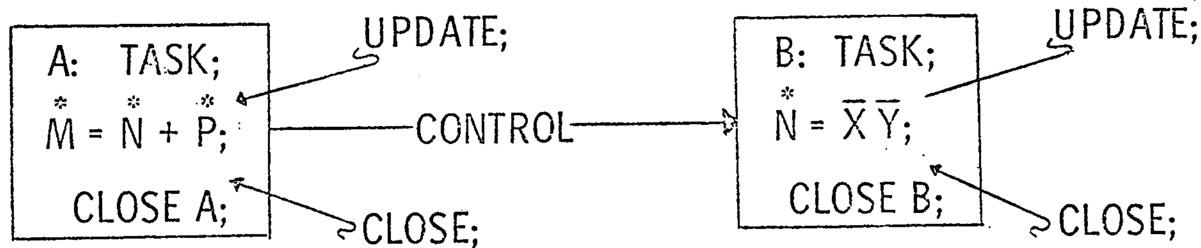
SLIDE 13

REAL TIME STATEMENT EXAMPLES

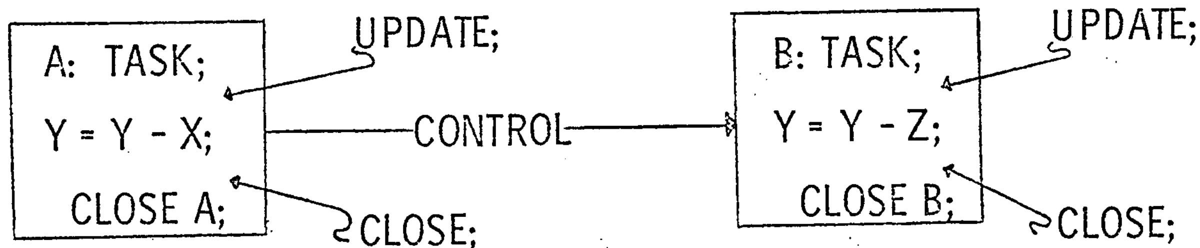
```
SCHEDULE TARGETING PRIORITY(3);

SCHEDULE RADAR ON R_RUPT PRIORITY(PRIO + 2) RADAR_PROG;

IF TRACKFLAG = ON THEN SCHEDULE AUTOMANEUVER IN 5;
        ELSE WAIT UNTIL (TIME + 5);

SCHEDULE STEERING AT(IGNITION + 3)PRIORITY(10) INDEPENDENT;

TERMINATE RADAR_PROG;

WAIT FOR OK;

SIGNAL OK;
```

CONTROL OF SHARED DATA

EXAMPLE 1: READ AND WRITE CONFLICTS

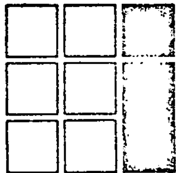


EXAMPLE 2: UPDATE CONFLICTS



NOTES:

1. B "INTERRUPTS" A IN BOTH CASES
2. #1 TASK A RESUMES USING OLD AND NEW VALUES FOR $\overset{*}{N}$
3. #2 TASK A RESUMES "CLOBBERING" THE VALUE FOR Y SET BY TASK B



INTERMETRICS

SLIDE 8

ERROR CONDITION STATEMENTS EXAMPLES:

ON ERROR_{12} GO TO ABLE;

ON $\text{ERROR}_{1 \text{ TO } 5}$ GO TO BAKER;

ON ERROR_{4} SYSTEM;

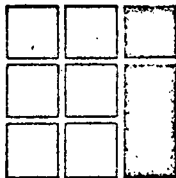
E_RUPT ERROR_{6};

SUMMARY

* HAL emphasizes reliability
    * Readability
    * Data protection

* HAL is a full-capability language
    * Includes all data types
    * Real-time control statements
    * Supports on-board computer software
    * Floating- or fixed-point syntax
    * Supports ground, checkout, simulation software

* Schedule of Events
    * First version delivery to MSC in June, 1971
    * Development to continue compatible with Shuttle schedule
    * Object-code-module required for selected on-board computer



INTERMETRICS

SLIDE 14

A.2 LONGER HAL COURSE

HAL Data Operations - I

Arithmetic

All common operations including:

* Vector dot, cross and outer - products
* Matrix multiplication, inverse transpose
* Integer mathematics
* Combined integer-scalar operation

Bit String

Logical AND, OR, NOT, of:

* Long bit strings, bit-by-bit
* Single bit "booleans"

Concatenation

Character String

Concatenation of characters and data into messages

Arrays

Most valid element-by-element operations apply to arrays

HAL Data Operations - II

Comparisons

A comparison of data always results in a single TRUE or FALSE answer.

* Absolute comparison

VECTOR, MATRIX, ARRAY, STRUCTURE	= NOT=

* Absolute and Relative Comparisons

scalar integer bit character	= NOT= <= >= NOT < NOT > > <

EXAMPLES OF DATA OPERATIONS - I

Arithmetic Operations

$I = (-J)^3 + K(3+J);$ ---- all integers

$A = P/R + R^P - \overline{V}.\overset{*}{M}\,\overline{V};$ ---- scalars, vectors, matrices

$\overline{B} = -\overline{P} * \overline{V} + 5\,\overset{*}{M}\,\overline{V} + A(\overline{V}.\overline{P})^2\,\overline{F} + \overline{F}/(\overline{V}.\overline{V}\,\overset{*}{M});$ --- scalars, vectors, matrices

$\overset{*}{C} = -\overset{*}{M}\,\overset{*}{N} + \overset{*}{M}^{-1} + \overline{V}\,\overline{V}/A - (\overset{*}{M}+\overset{*}{N})^T;$ ---- scalars, vectors, matrices

$A = I/J + B\,J + J^{-6.4};$ --------- integers, scalars

EXAMPLES OF STRING OPERATIONS

Bit String

$\dot{D} = \dot{B}$ AND $\dot{C}$;

$\dot{D} = \dot{A}$ OR ($\dot{B}$ AND $\dot{C}$);

$\dot{A} = \dot{D}$||NOT $\dot{B}$||($\dot{B}$ AND $\dot{C}$);

Character String

C = 'PLEASE'||'HELP';

D = 'THE ANS. IS'||X||'N.M.';

EXAMPLES OF ARRAY OPERATIONS

"Two-array" operations:

$$-[P]/[A],\ [\bar{P}] * [\bar{V}],\ [R]^{[P]},\ [\dot{A}] \text{ OR } [\dot{B}]$$

One-array operations:

$$-P/[A],\ \bar{P} * [\bar{V}],\ R^{[P]},\ [\dot{A}] \text{ OR } \dot{B}$$

EXAMPLES OF COMPARISON OPERATIONS

IF I > J THEN ------

IF A NOT < $(X^2 - 5\ \bar{V}.\bar{V})$ THEN ------

DO WHILE $\dot{S} = (\dot{A}$ AND $\dot{B})$ ------

IF $\overset{*}{M} = \overset{*}{N}$ THEN------

DO WHILE [A] NOT = [B] ------

HAL Explicit Declarations

* In general, all data must be declared by declare statements (with the exception of those permitted by implicit declarations).

* The declare statement specifies the name, organization, type and attributes of data quantities.

Keywords used in declare statements:

Organization	Type	Attributes
ARRAY	INTEGER	PRECISION
	SCALAR (optional)	INITIAL
	VECTOR	CONSTANT
	MATRIX	STATIC
	BIT	AUTOMATIC
	CHARACTER	LOCKTYPE
		DENSE
		ALIGNED
		VARYING

EXAMPLES OF EXPLICIT DECLARATIONS

1. DECLARE I INTEGER INITIAL (65);

   I is an integer with an initial value = 65.

2. DECLARE X PRECISION (8) AUTOMATIC INITIAL (6.061);

   X is a floating point scalar with at least 8 significant decimal digits.

3. DECLARE A ARRAY (5, 3, 4) VECTOR (6) PRECISION (10);

   A is a 5x3x4 array. Each element is a 6 dimensional vector with components represented to 10 significant decimal digits.

4. DECLARE MATRIX (3, 4) INITIAL (0) AUTOMATIC

   A, B, C PRECISION (10);

   A, B, and C are all (3, 4) matrices with automatic storage. All components are set to zero.

5. DECLARE A PRECISION (10, 15)

   A is a fixed point scalar with 10 integer bits and at least 15 fractional bits (i.e. maximum value $< 2^{10}$, granularity $\leq 2^{-15}$).

INDEX OPERATORS

1. The "TO" operator

   Selects a subset of elements from element-i "TO" element-j.

   For example:

   A_1 TO 10

2. The "AT" operator

   Selects a subset of N-elements starting at element-i.

   For example:

   A_{10} AT 1

3. The number of elements in any "partition" must be known at compile-time.

EXAMPLES OF INDEXING - I

1. Vectors and Matrices (given $\bar{V}$, $\overset{*}{M}$)

   V_2 -------- scalar element,

   $\bar{V}_{1 \text{ TO } 5}$ ----- sub-vector,

   $M_{2,3}$ ------ scalar element,

   $\bar{M}_{*,1}$ ------ vector element, $\overset{*}{M}_{3 \text{ AT } P,\ 3 \text{ AT } Q}$ ----- sub-matrix

2. Bit and Character Strings (given $\dot{S}$, C)

   $\dot{S}_3$ --------- single bit,

   $\dot{S}_{2 \text{ TO } 10}$ ----- sub-string, $\dot{S}_{6 \text{ AT } P}$ --------- sub-string

   $\acute{C}_{3 \text{ TO } \#}$ ------ sub-string,

EXAMPLES OF INDEXING - II

3. Arrays (given $[\overset{*}{A}]$, a two-dimensional array of matrices)

   $[\overset{*}{A}]_{1 \text{ TO } 4,\ 3 \text{ TO } 6:}$ --------------- sub-array

   $[A]_{P,Q}$ --------------------------- an array of scalar elements

   $[\bar{A}]_{*,*:*,2} \equiv [\bar{A}]_{*,2}$ --------------- sub-array of vector elements

4. Array of Bit Strings

   $[\ddot{A}]_{3 \text{ TO } 5,\ *:1 \text{ TO } 6}$ ----------------- sub-array of sub-strings

   $\dot{A}_{6,4:3}$ --------------------------- one particular bit

DO - STATEMENTS

- DO - statements block out a set of statements which are to be treated as a single unit.

- There are four types of DO - statements

  1. Simple DO-END

  2. Iterative DO-FOR

  3. Iterative DO-WHILE

  4. Selective DO-CASE

EXAMPLES OF DO-STATEMENTS - I

Simple DO-END

```
IF X>5  THEN BAKER: DO;
                       A = B;
                       C = D;
                    GO TO ABLE:
          END BAKER;

     ELSE CHARLIE:  DO;
```

$$\overline{Z} = \overset{*}{M}\ \overline{V};$$

$$\overline{F} = \overline{V} * \overline{Z};$$

```
               IF Y = 0 THEN  GO TO OUT;
          END CHARLIE;
```

EXAMPLES OF DO-STATEMENTS - II

1. Iterative DO-FOR

ABLE: DO FOR I = P TO (N/S) BY L WHILE N>0; — limits and increment are computed once.

$X = Y^2 + A_I;$

N = N - .006 X;

P = 1; S = 2; L = 3;

END ABLE;

2. Iterative DO-WHILE

ABLE: DO WHILE (X > Y AND GO_FLAG = ON); — X,Y and GO_FLAG are recomputed.

$X = Y^2 + P \, LOG(Z);$

GO_FLAG = TRAKFLAG OR NAV_FLAG;

[Statements
⋮ ⋮]

END ABLE;

EXAMPLE: SEARCHING AN ARRAY OF DATA

The final phase reference for Apollo reentry:

```
DECLARE ARRAY (13) VREF CONSTANT (994, 2103, 3922,...);
DECLARE ARRAY (13) RDOTREF CONSTANT (-690.0, -719,...);
DECLARE ARRAY (13) DREFR CONSTANT (41.15, 60, 81.5,...);
     .
     .
     .
     .
    etc.
```

INTERPOLATE: I = 0;

DO WHILE ($VREF_I$ NOT< V) AND (V NOT< $VREF_{I+1}$);

I = I + 1;

END;

GRAD = (V - $VREF_I$)/($VREF_{I+1}$ - $VREF_I$);

RDOTREF_V = $RDOTREF_I$ + GRAD ($RDOTREF_{I+1}$ - $RDOTREF_I$);

GO TO CONTINUE;

EXAMPLES OF DO-STATEMENTS - III

Selective DO-CASE (Computed DO-Statement)

ABLE: DO CASE N;		
	$X = Y^2$;	CASE 1
	BAKER: DO CASE P;	CASE 2
	$F = A + B$;	CASE 1
	$\overline{G} = \overset{*}{M}\ \overline{V}$;	CASE 2
	END;	
	GO TO CHARLIE;	CASE 3
	$\overline{Z} = \overline{W} + \overline{B}$;	CASE 4
	END ABLE;	

EXAMPLES OF IF-STATEMENTS

1. Simple:

   IF X = 5 AND Y > 6 THEN ABLE: GO TO PLACE;

   ELSE GO TO TRY_AGAIN;

2. Complex:

   IF X = 5 THEN IF Y > 6 THEN IF $\dot{B}$ OR $\dot{C}$ THEN $\overline{Z} = \overset{*}{M}\ \overline{V}$;

   ELSE CHOICE: $\overline{Z} = \overset{*}{M}^{-1}\ \overline{V}$;

3. More Complex:

   IF $\dot{S}$ = ($\dot{A}$ OR $\dot{B}$) THEN IF X > 5 AND Y > 6 THEN GO TO OUT;

   ELSE IF [$\dot{A}$] NOT= [$\dot{B}$] THEN [$\dot{A}$] = [$\dot{C}$];

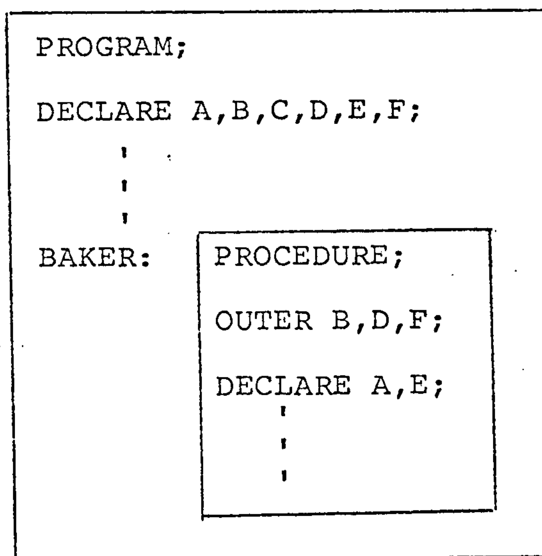
   ELSE IF -----THEN -----; ELSE IF ----- THEN -----;

   ELSE GO TO TRY_AGAIN;

note: ELSE always refers to immediately preceding IF.

SELECTIVE INCLUSION OF OUTER-NAMES

ABLE;



- Only B,D,F are recognized outer names. (A,C,E are "rejected").
- A,E are defined locally.

Note: COMPOOL variable-names may be accepted, rejected and/or locally defined by combinations of DECLARE and OUTER statements. In order to use implicit declarations within a block (except for PROGRAM-level) an OUTER-statement must be present.

APPENDIX B.

HAL-in-HAL Detailed Description and Listing

B.1 INTRODUCTION

The output from the HAL-in-HAL experiment consists of the following:

1. The HAL program listing, symbol table, and cross reference

2. The output of the HAL program execution

3. A listing of the source data read by HAL-in-HAL (a sample HAL program)

B.2 HAL-in-HAL PROGRAM DESCRIPTION

This section will describe the general function of each of the routines in HAL-in-HAL. Refer to the program listing for specific details.

B.2.1 STREAM (Statements 100-542)

The overall functions of the STREAM procedure are as follows:

1. Convert the multi-line format of the input cards to a one-line format, which is required by the scanning and syntactic analysis routines.

2. Process Comment and Heading cards to aid readability of the source program, and also to enable certain compiler toggles for assisting the person who is debugging the compiler.

3. Eliminate HAL in-line comments (strings contained between /* and */).

4. Perform substitutions for replace type strings (not demonstrated).

To convert input to one-line format requires the following:

1. Enclosure of each level of subscripting in parentheses, preceded by the dollar-sign character ($).

   Ex.

```
              M S1  = S2   ;
              S   I     IX
              S           I
        becomes

            S1$(I) = S2$(IX$(I));
```

2. Enclosure of each level of exponentiation in parentheses, preceded by two asterisks (**).

   Ex.

```
            E              2
            E      2     I   ;
            M   S2    = X

        becomes

            S2**(2) = X**(I**(2));
```

STREAM contains ten local subroutines which assist it in performing its function. They are local because they are of no value outside of STREAM. The non-local procedures are general interest routines, which are useful at levels other than within the STREAM procedure.

B2.2 CARD_TYPE (Statements 26-33)

CARD_TYPE is a function which receives as input the first character from an input card and returns an integer typifying the card in one of five classes.

B2.3 CHAR_INDEX (Statements 34-44)

CHAR_INDEX locates one string within another, returning the relative position of the desired substring if found, and 0 otherwise.

B2.4 PAD (Statements 45-53)

The PAD function forces a varying character string to a minimum specified fixed length by appending trailing blanks.

Its primary function is for WRITE list items.

B2.5 I_FORMAT (Statements 54-61)

The I_FORMAT function first converts a number to a character string, and then adds high order blanks to force a right justified integer string of a specified fixed length. This routine is also primarily for WRITE list items.

B2.6 ERRORS (Statements 62-82)

ERRORS both prints error messages when reported, and saves a record of their occurrence for later reporting. It also will terminate the compilation if either too many or too severe errors occur during compilation.

B2.7 PROCESS_COMMENT (Statements 116-133)

This routine processes heading cards, as well as looking for special debugging directives on comment cards.

B2.8 STACK_RETURN_CHAR (Statements 134-144)

This routine locates an available position in the return stack and records both a count and the character to be added to the output stream to formulate one-line output out of multi-line input.

B2.9 READ_CARD (Statements 145-157)

This routine reads the next input card and prints the card previously read. (This is because a group is only defined by the next non-group card.) It also counts the input cards and checks for an end of input condition.

B2.10 ORDER_OK (Statements 158-189)

This routine verifies that cards are in the proper sequence to formulate a proper HAL group. It also signals when a group is completed.

B2.11 COMMENT (Statements 190-201)

This routine removes '/* */' type comments from E and S lines, when they exist.

B2.12 SCAN_CARD (Statements 202-220)

This routine scans E and S cards for non-blank characters, compressing multiple lines into one line, with an indicator recording which level the character appeared on. If an overlap occurs, the one closest to the M line is retained, and a diagnostic is issued.

B2.13 COMP (Statements 221-244)

This routine is called when either an E line or S line is first encountered in a group. It keeps reading cards and calling SCAN until an entire E or S group is compressed into a single line as described in B2.12

B2.14 GET_GROUP (Statements 245-297)

This routine is called to assemble a complete group, which consists of an M line and one or more E and/or S lines, formulating as output a single E line, M line, and S line, with corresponding indicators.

B2.15 CHOP (Statements 298-306)

This routine advances the M line character index by 1, forcing a new group to be read when the M line termination is reached. The information concerning the last character on the previous card is retained.

B2.16 STACK (Statements 307-325)

This routine builds on Exponent or Subscript stack corresponding to a single blank field on an M line; i.e., those subscripts and/or exponents related to a specified variable or function. This is in preparation for outputting from STREAM within stacked Return characters (see B2.8).

B2.17 BUILD_XSCRIPTS (Statements 326-362)

This procedure invokes STACK for building both the E and S stacks for a single blank field, including M line comments as blank fields. Residue blanks for which no E or S line characters exist are treated as blank fields by the scanner (blank fields by the scanner (blank is a legal delimiter in HAL).

B.3 DESCRIPTION OF HAL-IN-HAL OUTPUT

The output of the HAL program is interpreted thusly. The program reads in a group of data cards. A group can consist of:

1. a single M line
2. one or more E lines followed by a single M line
3. a single M line followed by one or more S lines
4. one or more E lines followed by a single M line followed by one or more S lines

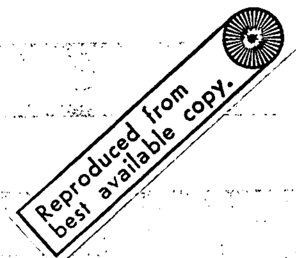
Each group is then converted into one line of output to the scanning routine. A group of type 1 is transmitted directly except for elimination of redundant blanks. A group of the other three types involves processing of S line and/or E line stacks and the addition of the appropriate subscript and/or exponent enclosures (subscripting is performed first if the M line identifier has both a subscript and an exponent attached). As the scanning routine continuously calls STREAM, identifier tokens are formed. All identifier and special character tokens are printed as they are encountered, including identifier over punches. When the token is a blank, a count of the blanks scanned is printed.

The full output consists of the following:

1. a printout of the entire card group just read, complete with card numbering
2. a printout of the individual tokens encountered within the group just read (signalled by TOKEN= or BLANKS=)
3. a printout of the combined one-line format and any possible over-punch characters whenever a semi-colon (;) is returned by STREAM. This represents a complete HAL statement as seen by the scanning routine (signalled by OVER and MAIN in succession).

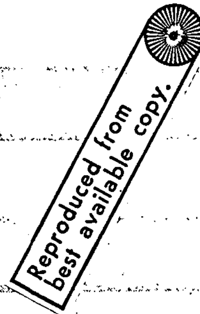
This sequence is repeated until the input is exhausted (signalled by a "?" for this test). Note in the TOGGLE sequences where the subscript and exponent enclosures are added to the characters on the actual input cards.

```
 STMT                                  SOURCE                                              LINE CURRENT SCOPE

    1 M| HALINHAL: PROGRAM;                                                               | 1
      C|    A   B   C   D   E   F   G                                                     | 2
      C| THE FOLLOWING GROUP OF DECLARES ARE ONLY TEMPORARY                               | 3
    2 M| REPLACE R BY 'REPLACE';                                                          | 4   HALINHAL
    3 M| R FALSE BY '0';  R TRUE BY '1';  R BIT_1 BY 'INTEGER';                           | 5   HALINHAL
    6 M| R IF_NOT BY 'IF 0 =';  R IFF BY 'IF 1 =';                                        | 6   HALINHAL
    8 M| R OUT BY 'WRITE(6)';                                                             | 7   HALINHAL
    9 M| R FOREVER BY 'WHILE 1 = 1';                                                      | 8   HALINHAL
   10 M| DEFAULT INTEGER;                                                                 | 9   HALINHAL
   11 M| DECLARE X70 CHARACTER(70) INITIAL (' ');                                         | 10  HALINHAL
   12 M| DECLARE CHARACTER(255) VARYING, BUILT, BUILT_UP, BUILT_TOKEN, BUILT_O_P;         | 11  HALINHAL
   13 M| DECLARE BLANK_FLAG BIT_1 INITIAL(FALSE);                                         | 12  HALINHAL
   14 M| DECLARE INTEGER INITIAL(0),                                                      | 13  HALINHAL
   14 M|  ERROR_COUNT, MACRO_POINT, MACRO_LIMIT,                                          | 14  HALINHAL
   14 M|  OLD_LEVEL, NEW_LEVEL,                                                           | 15  HALINHAL
   14 M|  MAX_SEVERITY, STATEMENT_SEVERITY,                                               | 16  HALINHAL
   14 M|  SAVE_SEVERITY  ARRAY(100),                                                      | 17  HALINHAL
   14 M|  SAVE_LINE ARRAY(100), PREVIOUS_ERROR, I;                                        | 18  HALINHAL
   15 M| DECLARE COMPILING BIT_1 INITIAL(TRUE);                                           | 19  HALINHAL
   16 M| DECLARE INTEGER, TIME1, TIME2;                                                   | 20  HALINHAL
   17 M| DECLARE MACRO_STREAM CHARACTER(255) VARYING;                                     | 21  HALINHAL
   18 M| DECLARE MACRO_FOUND BIT_1 INITIAL(FALSE);                                        | 22  HALINHAL
   19 M| DECLARE CARD_COUNT INTEGER INITIAL(-1);                                          | 23  HALINHAL
   20 M| DECLARE DISASTER LABEL;                                                          | 24  HALINHAL
   21 M| DECLARE NEXT_CHAR CHARACTER(1);                                                  | 25  HALINHAL
```


Reproduced from best available copy.

```
STMT                              SOURCE                                        LINE CURRENT SCOPE

  22 M|  DECLARE OVER_PUNCH CHARACTER(1);                                       | 26   HALINHAL
  23 M|  DECLARE BLANK_COUNT INTEGER INITIAL(0);                                | 27   HALINHAL
  24 M|  DECLARE CONTROL ARRAY(10) BIT(32);                                     | 28   HALINHAL
  25 M|  DECLARE TOGGLES CHARACTER(10) INITIAL('1234567890');                   | 29   HALINHAL
     C|                                                                         | 30   HALINHAL
     C|                                                                         | 31   HALINHAL
     C|  THE ARRAY CARD_TYPE IN XPL IS REPLACED BY THIS FUNCTION                | 32   HALINHAL
     C|                                                                         | 33   HALINHAL
  26 M|  CARD_TYPE:                                                             | 34   HALINHAL
  26 M|     FUNCTION(SELECT) INTEGER;                                           | 35   CARD_TYPE
  27 M|     DECLARE SELECT CHARACTER(1);                                        | 36   CARD_TYPE
  28 M|     IF SELECT = 'M' OR SELECT = ' ' THEN RETURN 2;                      | 37   CARD_TYPE
  29 M|     IF SELECT = 'S' THEN RETURN 3;                                      | 38   CARD_TYPE
  30 M|     IF SELECT = 'E' THEN RETURN 1;                                      | 39   CARD_TYPE
  31 M|     IF SELECT = 'C' OR SELECT = 'D' OR SELECT = 'H'                     | 40   CARD_TYPE
  31 M|        THEN RETURN 4;                                                   | 41   CARD_TYPE
  32 M|     RETURN 0;                                                           | 42   CARD_TYPE
  33 M|     CLOSE CARD_TYPE;                                                    | 43   HALINHAL
     C|                                                                         | 44   HALINHAL
     C|                                                                         | 45   HALINHAL
     C|  THE CHAR_INDEX FUNCTION IS THE SAME AS THE INDEX BUILT IN FUNCTION     | 46   HALINHAL
     C|                                                                         | 47   HALINHAL
  34 M|  CHAR_INDEX: FUNCTION(STRING,PATTERN) INTEGER;                          | 48   CHAR_INDEX
  35 M|   DECLARE CHARACTER(*), STRING, PATTERN;                                | 49   CHAR_INDEX
  36 M|   DECLARE INTEGER, I, J, K;                                             | 50   CHAR_INDEX
  37 M|   J = LENGTH(STRING);                                                   | 51   CHAR_INDEX
  38 M|   K = LENGTH(PATTERN);                                                  | 52   CHAR_INDEX
  39 M|   IF K > J THEN RETURN 0;                                               | 53   CHAR_INDEX
  40 M|   ELSE DO FOR I = 1 TO J-K+1;                                           | 54   CHAR_INDEX
  41 M|     IF STRING$(K AT I) = PATTERN THEN RETURN I;                         | 55   CHAR_INDEX
  42 M|   END;                                                                  | 56   CHAR_INDEX
```

```
STMT                                SOURCE                                          LINE CURRENT SCOPE
  43 M|   RETURN 0;                                                               | 57   CHAR_INDEX
  44 M|  CLOSE CHAR_INDEX;                                                        | 58   HALINHAL
     C|                                                                           | 59   HALINHAL
     C|                                                                           | 60   HALINHAL
     C|  HERE IS PAD A FUNCTION FROM HALPASS1                                     | 61   HALINHAL
     C|                                                                           | 62   HALINHAL
  45 M|  PAD: FUNCTION (STRING.WIDTH) CHARACTER(255) VARYING;                     | 63   PAD
  46 M|  DECLARE CHARACTER(*), STRING;                                            | 64   PAD
  47 M|  DECLARE CHARACTER(255) VARYING, TEMP_STRING;                             | 65   PAD
  48 M|  DECLARE INTEGER, L, WIDTH;                                               | 66   PAD
  49 M|  L = LENGTH(STRING);                                                      | 67   PAD
  50 M|  IF L < WIDTH THEN                                                        | 68   PAD
  50 M|  TEMP_STRING = STRING || X70$(1 TO WIDTH-L);                              | 69   PAD
  51 M|   ELSE TEMP_STRING = STRING;                                              | 70   PAD
  52 M|  RETURN TEMP_STRING;                                                      | 71   PAD
  53 M|  CLOSE PAD;                                                               | 72   HALINHAL
     C|                                                                           | 73   HALINHAL
     C|                                                                           | 74   HALINHAL
     C|   HERE IS THE FUNCTION I_FORMAT FROM HALPASS1                             | 75   HALINHAL
     C|                                                                           | 76   HALINHAL
  54 M|  I_FORMAT: FUNCTION(NUMBER,WIDTH) CHARACTER(*);                           | 77   I_FORMAT
  55 M|  DECLARE CHARACTER(255) VARYING, STRING;                                  | 78   I_FORMAT
  56 M|  DECLARE INTEGER, L, NUMBER, WIDTH;                                       | 79   I_FORMAT
  57 M|  STRING = NUMBER;                                                         | 80   I_FORMAT
  58 M|  L = LENGTH(STRING);                                                      | 81   I_FORMAT
  59 M|  IF L < WIDTH THEN                                                        | 82   I_FORMAT
  59 M|  STRING = X70$(1 TO WIDTH-L) || STRING;                                    | 83   I_FORMAT
  60 M|  RETURN STRING;                                                           | 84   I_FORMAT
  61 M|  CLOSE I_FORMAT;                                                          | 85   HALINHAL
     C|                                                                           | 86   HALINHAL
     C|                                                                           | 87   HALINHAL
     C|  THE FOLLOWING IS THE ERROR PROCEDURE FROM  PASS1.                        | 88   HALINHAL
```



```
STMT                              SOURCE                                    LINE CURRENT SCOPE
     C|                                                                    | 89   HALINHAL
  62 M|  ERRORS: PROCEDURE (MESSAGE,SEVERITY);                             | 90   ERRORS
     C|  PRINTS AND ACCOUNTS FOR ALL ERRORS                                | 91   ERRORS
  63 M|  DECLARE MSG CHARACTER(255) VARYING;                               | 92   ERRORS
  64 M|  DECLARE MESSAGE CHARACTER(*);                                     | 93   ERRORS
  65 M|  DECLARE SEVERITY INTEGER;                                         | 94   ERRORS
  66 M|  ERROR_COUNT = ERROR_COUNT  + 1;                                   | 95   ERRORS
  67 M|  MSG = '***** ERROR # ' || ERROR_COUNT || ' OF SEVERITY ' ||       | 96   ERRORS
  67 M|     SEVERITY || ': ' || MESSAGE;                                   | 97   ERRORS
  68 M|  IF ERROR_COUNT > 1 THEN                                           | 98   ERRORS
  68 M|     MSG = MSG || '. LAST ERROR ON LINE ' || PREVIOUS_ERROR;        | 99   ERRORS
  69 M|  WRITE(6)MSG || '. *****';                                         | 100  ERRORS
  70 M|  SAVE_SEVERITY$(ERROR_COUNT) = SEVERITY;                           | 101  ERRORS
  71 M|  SAVE_LINE$(ERROR_COUNT) = CARD_COUNT;                             | 102  ERRORS
  72 M|  PREVIOUS_ERROR = CARD_COUNT;                                      | 103  ERRORS
  73 M|  IF ERROR_COUNT >= 99 THEN DO;                                     | 104  ERRORS
  74 M|  IF ERROR_COUNT = 100 THEN                                         | 105  ERRORS
  74 M|    WRITE(6) 'TOO MANY ERRORS: COMPILATION ABORTED.';               | 106  ERRORS
  75 M|  IFF COMPILING THEN COMPILING = FALSE;                             | 107  ERRORS
  76 M|    ELSE GO TO DISASTER;  /* THEN END IT THERE */                   | 108  ERRORS
  77 M|    END;                                                            | 109  ERRORS
  78 M|  IF SEVERITY > MAX_SEVERITY THEN MAX_SEVERITY = SEVERITY;          | 110  ERRORS
  79 M|  IF SEVERITY > STATEMENT_SEVERITY THEN                             | 111  ERRORS
  79 M|     STATEMENT_SEVERITY = SEVERITY;                                 | 112  ERRORS
  80 M|  IF SEVERITY > 2 THEN IFF COMPILING THEN                           | 113  ERRORS
  80 M|     COMPILING = FALSE;                                             | 114  ERRORS
  81 M|  ELSE GO TO DISASTER;                                              | 115  ERRORS
     C|  END OF PASS1 ERROR                                                | 116  ERRORS
```

```
STMT                              SOURCE                                        LINE CURRENT SCOPE
  82 M|  CLOSE ERRORS; /* CLOSING ERRORS AND RETURNING. */                      | 117  HALINHAL
     C|                                                                         | 118  HALINHAL
     C|                                                                         | 119  HALINHAL
     C|   THIS NEXT PROCEDURE IS PRINT_SUMMARY                                  | 120  HALINHAL
     C|                                                                         | 121  HALINHAL
  83 M|  PRINT_SUMMARY: PROCEDURE;                                              | 122  PRINT_SUMMARY
  84 M|  WRITE(6)CARD_COUNT || ' CARDS WERE PROCESSED';                         | 123  PRINT_SUMMARY
  85 M|  IF ERROR_COUNT = 0 THEN WRITE(6) ' NO ERRORS WERE DETECTED.';          | 124  PRINT_SUMMARY
  86 M|  ELSE IF ERROR_COUNT > 1 THEN DO;                                       | 125  PRINT_SUMMARY
  87 M|  WRITE(6) ERROR_COUNT || ' ERRORS WERE DETECTED,'                       | 126  PRINT_SUMMARY
  87 M|      || ' THE LAST ERROR WAS ON LINE ' || PREVIOUS_ERROR;                | 127  PRINT_SUMMARY
  88 M|  WRITE(6) '*****SUMMARY OF DETECTED ERRORS.*****';                      | 128  PRINT_SUMMARY
  89 M|  DO FOR I = 1 TO ERROR_COUNT;                                           | 129  PRINT_SUMMARY
  90 M|  WRITE(6) 'ERROR # ' || I || ' ON LINE ' ||                             | 130  PRINT_SUMMARY
  90 M|      SAVE_LINE$I || ' OF SEVERITY ' ||                                  | 131  PRINT_SUMMARY
  90 M|      SAVE_SEVERITY$I || '.';                                            | 132  PRINT_SUMMARY
  91 M|  END;                                                                   | 133  PRINT_SUMMARY
  92 M|  END;                                                                   | 134  PRINT_SUMMARY
  93 M|   ELSE WRITE(6) 'ONE ERROR WAS DETECTED WHICH OCCURRED ON LINE ' ||     | 135  PRINT_SUMMARY
  93 M|      PREVIOUS_ERROR || ' OF SEVERITY ' || SAVE_SEVERITY$1 || '.';       | 136  PRINT_SUMMARY
  94 M|  TIME2 = TIME;                                                          | 137  PRINT_SUMMARY
  95 M|  I =TIME2-TIME1;                                                        | 138  PRINT_SUMMARY
  96 M|  WRITE(6) 'CARD-PROCESSING RATE: ' || 6000 CARD_COUNT/I ||              | 139  PRINT_SUMMARY
  96 M|      ' CARDS PER MINUTE.';                                              | 140  PRINT_SUMMARY
  97 M|  WRITE(6) ' CLOCK TIME IS ' || TIME2;                                   | 141  PRINT_SUMMARY
  98 M|  RETURN;                                                                | 142  PRINT_SUMMARY
  99 M|  CLOSE PRINT_SUMMARY;                                                   | 143  HALINHAL
 100 M|  STREAM: PROCEDURE;                                                     | 144  STREAM
     C|   THIS PROCEDURE FILLS THE VARIABLES NEXT_CHAR,                         | 145  STREAM
     C|      ARROW, AND OVER_PUNCH.                                             | 146  STREAM
```

```
STMT                         SOURCE                                          LINE CURRENT SCOPE

     C|  NEXT_CHAR IS A ONE BYTE VARIABLE THAT CONTAINS THE                 | 147  STREAM
     C|      NEXT CHARACTER IN THE INPUT STREAM.                            | 148  STREAM
     C|  ARROW IS A HALF WORD VARIABLE THAT CONTAINS THE                    | 149  STREAM
     C|      INTEGER WHICH REPRESENTS THE RELATIVE                          | 150  STREAM
     C|      DISPLACEMENT OF THE CHARACTER IN NEXT_CHAR                     | 151  STREAM
     C|      WITH RESPECT TO THE LAST CHARACTER.                            | 152  STREAM
     C|      A POSSITIVE VALUE INDICATES A MOVE UP.                         | 153  STREAM
     C|  OVER_PUNCH IS A ONE BYTE VARIABLE THAT IS FILLED WITH              | 154  STREAM
     C|      A NON ZERO VALUE WHEN A CHARACTER OTHER                        | 155  STREAM
     C|      THAN A BLANK APPEARS DIRECTLY OVER AN                          | 156  STREAM
     C|      M-LIN CHARACTER--THE VALUE IS THE BYTE VALUE                   | 157  STREAM
     C|      OF THE OVER_PUNCH.                                             | 158  STREAM
     C|                                                                     | 159  STREAM

 101 M|  DECLARE CHARACTER(255) VARYING, E_LINE, S_LINE, M_LINE;            | 160  STREAM
 102 M|  DECLARE CHARACTER(80), SAVE_CARD, CURRENT_CARD;                    | 161  STREAM
 103 M|   DECLARE BLANKS CHARACTER(128) INITIAL(' ');                       | 162  STREAM
 104 M|  DECLARE E_LINE_ERR CHARACTER(50) VARYING                           | 163  STREAM
 104 M|  INITIAL('E_LINE CHARACTER MORE THAN 1 LINE ABOVE PRECEEDING CHARACTER'); | 164  STREAM
 105 M|  DECLARE ORDER_ERR CHARACTER(27) INITIAL('SOURCE PROGRAM OUT OF ORDER'); | 165  STREAM
 106 M|  DECLARE CHARACTER(1), PREV_CARD, SAVE_NEXT_CHAR, SAVE_OVER_PUNCH;  | 166  STREAM
 107 M|  DECLARE INTEGER INITIAL(0), LAST_E_IND, LAST_S_IND, E_BLANKS, M_BLANKS, | 167  STREAM
 107 M|      S_BLANKS, EP, SP, TEXT_LIMIT, E_COUNT, LAST_E_COUNT, S_COUNT,  | 168  STREAM
 107 M|      LAST_S_COUNT, INDX, CP, POINTER, ARROW, II;                    | 169  STREAM
 108 M|  DECLARE ARRAY(256) INTEGER INITIAL(0), E_IND, S_IND, E_INDICATOR,   | 170  STREAM
 108 M|      S_INDICATOR;                                                   | 171  STREAM
 109 M|  DECLARE BIT_1 INITIAL(FALSE), RETURNING_E, RETURNING_M, RETURNING_S, | 172  STREAM
 109 M|      END_GROUP, M_COMMENT, ARROW_FLAG;                              | 173  STREAM
 110 M|  DECLARE BIT_1 INITIAL(TRUE), FIRST_CALL_TO_STREAM;                 | 174  STREAM
 111 M|  DECLARE ARRAY(3) CHARACTER(1), TYPE_CHAR;                          | 175  STREAM
 112 M|  DECLARE ARRAY(3) INTEGER INITIAL(0), RETURN_CHAR;                  | 176  STREAM
 113 M|  DECLARE CHARACTER(255) VARYING, E_STACK, S_STACK;                  | 177  STREAM
 114 M|  DECLARE INPUT_PAD CHARACTER(23) INITIAL                            | 178  STREAM
 114 M|      ('M /* */ EOF''EOF EOF');                                      | 179  STREAM

     C|                                                                     | 180  STREAM
     C|                                                                     | 181  STREAM
```

```
STMT                         SOURCE                                          LINE CURRENT SCOPE

     C|  TWO DIMENSIONAL PROCEDURES.                                         | 182  STREAM
     C|                                                                      | 183  STREAM
 115 M|  GO TO STREAM_START;                                                 | 184  STREAM
 116 M|  PROCESS_COMMENT: PROCEDURE;                                         | 185  PROCESS_COMMENT
 117 M|  DECLARE K CHARACTER(1);                                             | 186  PROCESS_COMMENT
 118 M|  DECLARE J INTEGER;                                                  | 187  PROCESS_COMMENT
 119 M|  IF CURRENT_CARD$1 = 'H' THEN                                        | 188  PROCESS_COMMENT
 119 M|     WRITE(6) LINE(1), CURRENT_CARD$(2 TO #) /* ISSUE A NEW HEADER*/  | 189  PROCESS_COMMENT
 119 M|     ;                                                                | 190  PROCESS_COMMENT
 120 M|  ELSE                                                                | 191  PROCESS_COMMENT
 120 M|  IF CURRENT_CARD$1 = 'C' THEN                                        | 192  PROCESS_COMMENT
 120 M|   DO FOR I=1 TO TEXT_LIMIT-1;                                        | 193  PROCESS_COMMENT
 121 M|    IF CURRENT_CARD$I = ' ' THEN DO;                                  | 194  PROCESS_COMMENT
 122 M|     K = CURRENT_CARD$(I+1);                                          | 195  PROCESS_COMMENT
 123 M|     J = CHAR_INDEX(TOGGLES, K);                                      | 196  PROCESS_COMMENT
 124 M|     IF J ¬= 0 THEN DO;                                               | 197  PROCESS_COMMENT
 125 M|      IF I < TEXT_LIMIT-1 THEN                                        | 198  PROCESS_COMMENT
 125 M|        K = CURRENT_CARD$(I+2);                                       | 199  PROCESS_COMMENT
 126 M|        ELSE GO TO COMPLEMENT;                                        | 200  PROCESS_COMMENT
 127 M|      IF K = '+' THEN CONTROL  = TRUE;                                | 201  PROCESS_COMMENT
 127 S|                            J:                                       | 202  PROCESS_COMMENT
 128 M|      ELSE IF K = '-' THEN CONTROL  = FALSE;                          | 203  PROCESS_COMMENT
 128 S|                                  J:                                  | 204  PROCESS_COMMENT
 129 M|      ELSE                                                            | 205  PROCESS_COMMENT
 129 M|      COMPLEMENT: CONTROL  = NOT CONTROL  ;                           | 206  PROCESS_COMMENT
 129 S|                         J:             J:                           | 207  PROCESS_COMMENT
 130 M|     END;                                                             | 208  PROCESS_COMMENT
 131 M|    END;                                                              | 209  PROCESS_COMMENT
 132 M|   END;                                                               | 210  PROCESS_COMMENT
 133 M|  CLOSE PROCESS_COMMENT;                                              | 211  STREAM
```

Reproduced from best available copy.

```
STMT                              SOURCE                                        LINE CURRENT SCOPE
     C|                                                                       | 212  STREAM
     C|                                                                       | 213  STREAM

 134 M|  STACK_RETURN_CHAR:                                                   | 214  STREAM

 134 M|  PROCEDURE(NUMBER,CHARR);                                             | 215  STACK_RETURN_CHAR

 135 M|  DECLARE INTEGER, NUMBER, I;                                          | 216  STACK_RETURN_CHAR

 136 M|  DECLARE CHARR CHARACTER(1);                                          | 217  STACK_RETURN_CHAR

 137 M|  DO FOR I = 1 TO 3;                                                   | 218  STACK_RETURN_CHAR

 138 M|  IF RETURN_CHAR   = 0 THEN DO;                                        | 219  STACK_RETURN_CHAR
 138 S|                 I:                                                   | 220  STACK_RETURN_CHAR

 139 M|  RETURN_CHAR   = NUMBER;                                              | 221  STACK_RETURN_CHAR
 139 S|              I:                                                      | 222  STACK_RETURN_CHAR

 140 M|  TYPE_CHAR   = CHARR;                                                 | 223  STACK_RETURN_CHAR
 140 S|            I:                                                        | 224  STACK_RETURN_CHAR

     C|  THE FOLLOWING STATEMENT IS FOR DEBUGGING PURPOSES ONLY               | 225  STACK_RETURN_CHAR
     C|  OUT 'STACKED "' || CHARR || '"(' || NUMBER || ') - INDEX(' || I || ')';  | 226  STACK_RETURN_CHAR

 141 M|  RETURN;                                                              | 227  STACK_RETURN_CHAR

 142 M|  END;                                                                 | 228  STACK_RETURN_CHAR

 143 M|  END;                                                                 | 229  STACK_RETURN_CHAR

 144 M|  CLOSE STACK_RETURN_CHAR;                                             | 230  STREAM

     C|                                                                       | 231  STREAM
     C|                                                                       | 232  STREAM

 145 M|  READ_CARD:                                                           | 233  STREAM

 145 M|  PROCEDURE ;                                                          | 234  READ_CARD

 146 M|  DECLARE END_OF_INPUT BIT_1 INITIAL(FALSE);                           | 235  READ_CARD

 147 M|  IFF END_OF_INPUT THEN DO;                                            | 236  READ_CARD

 148 M|  CURRENT_CARD = INPUT_PAC;                                            | 237  READ_CARD

 149 M|  RETURN;                                                              | 238  READ_CARD

 150 M|  END;                                                                 | 239  READ_CARD

 151 M|  READ(5) CURRENT_CARD;                                                | 240  READ_CARD

     C|      END_OF_INPUT = TRUE;                                             | 241  READ_CARD

 152 M|  CARD_COUNT = CARD_COUNT + 1;                                         | 242  READ_CARD
```

```
 STMT                              SOURCE                                              LINE CURRENT SCOPE
  153 M|  IF CARD_COUNT NOT = 0 THEN DO;                                              | 243  READ_CARD
  154 M|  WRITE(6) I_FORMAT(CARD_COUNT, 4)                                             | 244  READ_CARD
  154 M|     || ' ' || SAVE_CARD$1 || '|' || SAVE_CARD$(2 TO #) || '| ' || CARD_COUNT; | 245  READ_CARD
  155 M|  END;                                                                         | 246  READ_CARD
  156 M|  SAVE_CARD = CURRENT_CARD;                                                    | 247  READ_CARD
  157 M|  CLOSE READ_CARD;                                                             | 248  STREAM
      C|                                                                               | 249  STREAM
      C|                                                                               | 250  STREAM
  158 M|  ORDER_CK:                                                                    | 251  STREAM
  159 M|       FUNCTION(TYPE) BIT_1;                                                   | 252  ORDER_OK
  159 M|  DECLARE TYPE CHARACTER(1);                                                   | 253  ORDER_OK
  160 M|  DO CASE CARD_TYPE(CURRENT_CARD$1) + 1;                                       | 254  ORDER_OK
  161 M|  DO; /*CASE 1--ILLEGAL CARD TYPE*/                                            | 255  ORDER_OK CASE 1
  162 M|  END_GROUP = FALSE;                                                           | 256  ORDER_OK
  163 M|  RETURN FALSE;                                                                | 257  ORDER_OK
  164 M|  END; /*OF CASE 1*/                                                           | 258  ORDER_OK
  165 M|  /* CASE 2--E CARD*/                                                          | 259  ORDER_OK
  165 M|  E_CARD:                                                                      | 260  ORDER_OK CASE 2
  165 M|  DO;                                                                          | 261  ORDER_OK
  166 M|  IF CARD_TYPE(TYPE) = 2 OR                                                    | 262  ORDER_OK
  166 M|        CARD_TYPE(TYPE) = 3 THEN END_GROUP = TRUE;                             | 263  ORDER_OK
  167 M|  ELSE END_GROUP = FALSE;                                                      | 264  ORDER_OK
  168 M|  RETURN TRUE;                                                                 | 265  ORDER_OK
  169 M|  END; /*CASE 2*/                                                              | 266  ORDER_OK
  170 M|  /*CASE 3--M CARD*/ GO TO E_CARD;                                             | 267  ORDER_OK CASE 3
  171 M|  DO; /* CASE 4--S CARD*/                                                      | 268  ORDER_OK CASE 4
  172 M|  END_GROUP = FALSE;                                                           | 269  ORDER_OK
  173 M|  IF CARD_TYPE(TYPE) = 2 OR                                                    | 270  ORDER_OK
```

```
STMT                               SOURCE                                  LINE CURRENT SCOPE
 173 M|      CARD_TYPE(TYPE) = 3 THEN RETURN TRUE;                        | 271  ORDER_OK
 174 M|   ELSE RETURN FALSE;                                              | 272  ORDER_OK
 175 M|   END; /* CASE 4*/                                                | 273  ORDER_OK
 176 M|   DO; /*CASE 5--A COMMENT*/                                       | 274  ORDER_OK CASE 5
 177 M|   IF CARD_TYPE(TYPE) = 2 OR                                       | 275  ORDER_OK
 177 M|      CARD_TYPE(TYPE) = 3 THEN END_GROUP = TRUE;                   | 276  ORDER_OK
 178 M|   ELSE END_GROUP = FALSE;                                         | 277  ORDER_OK
 179 M|   IF CURRENT_CARD$1 = 'P' THEN DO;                                | 278  ORDER_OK
 180 M|   IF TYPE = 'C' THEN RETURN TRUE;                                 | 279  ORDER_OK
 181 M|   ELSE RETURN FALSE;                                              | 280  ORDER_OK
 182 M|   END;                                                            | 281  ORDER_OK
 183 M|   ELSE DO;                                                        | 282  ORDER_OK
 184 M|   IF CARD_TYPE(TYPE) = 1 THEN RETURN FALSE;                       | 283  ORDER_OK
 185 M|   ELSE RETURN TRUE;                                               | 284  ORDER_OK
 186 M|   END;                                                            | 285  ORDER_OK
 187 M|   END; /* CASE 5 */                                               | 286  ORDER_OK
 188 M|   END; /*OF DO CASE*/                                             | 287  ORDER_OK CASE 6
 189 M|   CLOSE ORDER_OK;                                                 | 288  STREAM
     C|                                                                   | 289  STREAM
     C|                                                                   | 290  STREAM
 190 M|   COMMENT:                                                        | 291  STREAM
 190 M|      FUNCTION BIT_1;                                              | 292  COMMENT
 191 M|   IF CURRENT_CARD$CP = '/' THEN                                   | 293  COMMENT
 191 M|   IF CP < TEXT_LIMIT THEN                                         | 294  COMMENT
 191 M|   IF CURRENT_CARD$(CP + 1) = '*' THEN DO;                         | 295  COMMENT
     C|   LOOK FOR END OF COMMENT.                                        | 296  COMMENT
 192 M|   DO FOR CP = CP+2 TO TEXT_LIMIT-1;                               | 297  COMMENT
 193 M|   IF CURRENT_CARD$CP = '*' THEN                                   | 298  COMMENT
```

```
STMT                              SOURCE                                   LINE CURRENT SCOPE
 192 M|  IF CURRENT_CARD$(CP+1) = '/' THEN                                | 299  COMMENT
 193 M|  RETURN_TRUE:                                                     | 300  COMMENT
 193 M|  DO;                                                              | 301  COMMENT
 194 M|  CP = CP + 1;                                                     | 302  COMMENT
 195 M|  RETURN TRUE;                                                     | 303  COMMENT
 196 M|  END;                                                             | 304  COMMENT
 197 M|  END;                                                             | 305  COMMENT
 198 M|  GO TO RETURN_TRUE;                                               | 306  COMMENT
 199 M|  END;                                                             | 307  COMMENT
 200 M|  RETURN FALSE;                                                    | 308  COMMENT
 201 M|  CLOSE COMMENT;                                                   | 309  STREAM
     C|                                                                   | 310  STREAM
     C|                                                                   | 311  STREAM
 202 M|  SCAN_CARD:                                                       | 312  STREAM
 202 M|      PROCEDURE(TYPE,COUNT) ASSIGN(LIN,INDICATOR);                 | 313  SCAN_CARD
 203 M|  DECLARE INDICATOR ARRAY(256) INTEGER;                             | 314  SCAN_CARD
 204 M|  DECLARE INTEGER, TYPE, COUNT;                                    | 315  SCAN_CARD
 205 M|  DECLARE LIN CHARACTER(*);                                        | 316  SCAN_CARD
 206 M|  DO FOR CP = 2 TO TEXT_LIMIT;                                     | 317  SCAN_CARD
 207 M|  IF CURRENT_CARD$CP NOT = ' ' THEN DO;                            | 318  SCAN_CARD
 208 M|  IFF COMMENT THEN GO TO CONTINUE;                                 | 319  SCAN_CARD
 209 M|  IF LIN$CP NOT = ' ' THEN                                         | 320  SCAN_CARD
 209 M|  DO CASE TYPE + 1; /*HAL DO CASES START WITH 1*/                  | 321  SCAN_CARD
 210 M|  /* CASE 1*/ CALL ERRORS('OVERLAPPING E-LINE CHARACTERS',1);      | 322  SCAN_CARD CASE 1
 211 M|  DO; /* CASE 2*/                                                  | 323  SCAN_CARD CASE 2
 212 M|  CALL ERRORS('OVERLAPPING S-LINE CHARACTERS',1);                  | 324  SCAN_CARD
 213 M|  GO TO CONTINUE;                                                  | 325  SCAN_CARD
 214 M|  END; /*OF CASE 2*/                                               | 326  SCAN_CARD
```

Reproduced from best available copy.

3"
2"
1"

```
STMT                              SOURCE                                     LINE CURRENT SCOPE
 215 M|  END; /*OF DO CASE TYPE + 1*/                                       | 327  SCAN_CARD CASE 3
 216 M|  INDICATOR$CP = COUNT;                                              | 328  SCAN_CARD
 217 M|  LIN$CP = CURRENT_CARD$CP;                                          | 329  SCAN_CARD
 218 M|  END;                                                               | 330  SCAN_CARD
 219 M|  CONTINUE:                                                          | 331  SCAN_CARD
 219 M|  END;                                                               | 332  SCAN_CARD
 220 M|  CLOSE SCAN_CARD;                                                   | 333  STREAM
     C|                                                                     | 334  STREAM
     C|                                                                     | 335  STREAM
 221 M|  COMP:                                                              | 336  STREAM
 221 M|      PROCEDURE(TYPE) ASSIGN(LIN,INDICATOR,COUNT);                   | 337  COMP
 222 M|  DECLARE POINT CHARACTER(1);                                        | 338  COMP
 223 M|  DECLARE INTEGER, COUNT, TYPE;                                      | 339  COMP
 224 M|  DECLARE CHARACTER(*), LIN;                                         | 340  COMP
 225 M|  DECLARE ARRAY(256) INTEGER, INDICATOR;                             | 341  COMP
 226 M|  IF TYPE = 1 THEN POINT = 'S';                                      | 342  COMP
 227 M|  ELSE POINT = 'E';                                                  | 343  COMP
 228 M|  COUNT = 1;                                                         | 344  COMP
 229 M|  DO FOREVER;                                                        | 345  COMP
 230 M|  CALL SCAN_CARD(TYPE,COUNT) ASSIGN                                  | 346  COMP
 230 M|      (LIN,INDICATOR);                                               | 347  COMP
 231 M|  CALL READ_CARD;                                                    | 348  COMP
 232 M|  IF CURRENT_CARD$1 NOT = POINT THEN DO;                             | 349  COMP
     C|  NO MORE OF THIS TYPE CARD.                                         | 350  COMP
 233 M|  IF NOT ORDER_CK(POINT) THEN                                        | 351  COMP
 233 M|      CALL ERRORS(ORDER_ERR,1);                                      | 352  COMP
 234 M|  DO FOR CP = 2 TO TEXT_LIMIT;                                       | 353  COMP
 235 M|  IF LIN$CP = ' ' THEN                                               | 354  COMP
```

```
STMT                              SOURCE                                   LINE CURRENT SCOPE
 235 M|       INDICATOR$CP = 0;                                          | 355  COMP
 236 M|  ELSE IF TYPE = 0 THEN DO;                                       | 356  COMP
 237 M|  INDICATOR$CP = COUNT-INDICATOR$CP +1;                           | 357  COMP
 238 M|  END;                                                            | 358  COMP
 239 M|  END;                                                            | 359  COMP
 240 M|  RETURN;                                                         | 360  COMP
 241 M|  END;                                                            | 361  COMP
 242 M|  COUNT = COUNT + 1;                                              | 362  COMP
 243 M|  END;                                                            | 363  COMP
 244 M|  CLOSE COMP;                                                     | 364  STREAM
     C|                                                                  | 365  STREAM
     C|                                                                  | 366  STREAM
 245 M|  GET_GROUP:                                                      | 367  STREAM
 245 M|       PROCEDURE;                                                 | 368  GET_GROUP
 246 M|  E_LINE = E_LINES$(INDX TO #) || BLANKS;                         | 369  GET_GROUP
 247 M|  S_LINE = S_LINES$(INDX TO #) || BLANKS;                         | 370  GET_GROUP
 248 M|  LAST_E_COUNT = E_COUNT;                                         | 371  GET_GROUP
 249 M|  LAST_S_COUNT = S_COUNT;                                         | 372  GET_GROUP
 250 M|  E_COUNT, S_COUNT = 0;                                           | 373  GET_GROUP
 251 M|  GO TO LOOP;                                                     | 374  GET_GROUP
 252 M|  READ_IT:                                                        | 375  GET_GROUP
 252 M|  CALL READ_CARD;                                                 | 376  GET_GROUP
 253 M|  IF NOT ORDER_CK(PREV_CARD) THEN DO;                             | 377  GET_GROUP
 254 M|  CALL ERRORS(ORDER_ERR,1);                                       | 378  GET_GROUP
 255 M|  GO TO READ_IT;                                                  | 379  GET_GROUP
 256 M|  END;                                                            | 380  GET_GROUP
 257 M|  LOOP:                                                           | 381  GET_GROUP
 257 M|  IFF END_GROUP THEN GO TO FOUND_GROUP:                           | 382  GET_GROUP
```

```
STMT                        SOURCE                                LINE CURRENT SCOPE
 258 M| DO CASE CARD_TYPE(CURRENT_CARD$1) + 1;                   | 383  GET_GROUP
 259 M| DO; /*CASE 1, A DUMMY*/ END;                             | 384  GET_GROUP CASE 1
 261 M| DO; /*CASE 2 E-LIN*/                                     | 385  GET_GROUP CASE 2
 262 M| CALL COMP(0) ASSIGN                                      | 386  GET_GROUP
 262 M|     (E_LINE,E_INDICATOR,E_COUNT);                        | 387  GET_GROUP
 263 M| GO TO LOOP;                                              | 388  GET_GROUP
 264 M| END; /*OF CASE 2*/                                       | 389  GET_GROUP
 265 M| DO; /*CASE 3 M-LIN*/                                     | 390  GET_GROUP CASE 3
 266 M| M_LINE = M_LINE$(INDX TO #) || CURRENT_CARD$(2 TO #);    | 391  GET_GROUP
 267 M| SAVE_CARD$1 = 'M';                                       | 392  GET_GROUP
 268 M| PREV_CARD = CURRENT_CARD$1;                              | 393  GET_GROUP
 269 M| GO TO READ_IT;                                           | 394  GET_GROUP
 270 M| END; /*OF CASE 3*/                                       | 395  GET_GROUP
 271 M| DO; /*CASE 4 S-LIN*/                                     | 396  GET_GROUP CASE 4
 272 M| CALL COMP(1) ASSIGN                                      | 397  GET_GROUP
 272 M|     (S_LINE,S_INDICATOR,S_COUNT);                        | 398  GET_GROUP
 273 M| GO TO LOOP;                                              | 399  GET_GROUP
 274 M| END; /*OF CASE 4*/                                       | 400  GET_GROUP
 275 M| DO; /*CASE 5 COMMENT*/                                   | 401  GET_GROUP CASE 5
 276 M| PREV_CARD = CURRENT_CARD$1;                              | 402  GET_GROUP
 277 M| CALL PROCESS_COMMENT;                                    | 403  GET_GROUP
 278 M| GO TO READ_IT;                                           | 404  GET_GROUP
 279 M| END; /*OF CASE 5*/                                       | 405  GET_GROUP
 280 M| END; /*OF DO CASE*/                                      | 406  GET_GROUP CASE 6
 281 M| FOUND_GROUP:                                             | 407  GET_GROUP
 281 M| WRITE(6) SKIP(1);                                        | 408  GET_GROUP
 282 M| END_GROUP = FALSE;                                       | 409  GET_GROUP
 283 M| E_LINE = E_LINE$(1 TO LENGTH(M_LINE));                   | 410  GET_GROUP
```

```
STMT                              SOURCE                                      LINE CURRENT SCOPE
 284 M|  IF E_COUNT NOT > 0 THEN DO;                                          | 411  GET_GROUP
 285 M|  DO FOR CP = 2 TO TEXT_LIMIT;                                         | 412  GET_GROUP
 286 M|  E_INDICATOR$CP = 0;                                                  | 413  GET_GROUP
 287 M|  END;                                                                 | 414  GET_GROUP
 288 M|  E_COUNT = LAST_E_COUNT;                                              | 415  GET_GROUP
 289 M|  END;                                                                 | 416  GET_GROUP
 290 M|  S_LINE = S_LINE$(1 TO LENGTH(M_LINE));                               | 417  GET_GROUP
 291 M|  IF S_COUNT NOT > 0 THEN DO;                                          | 418  GET_GROUP
 292 M|  DO FOR CP = 2 TO TEXT_LIMIT;                                         | 419  GET_GROUP
 293 M|  S_INDICATOR$CP = 0;                                                  | 420  GET_GROUP
 294 M|  END;                                                                 | 421  GET_GROUP
 295 M|  S_COUNT = LAST_S_COUNT;                                              | 422  GET_GROUP
 296 M|  END;                                                                 | 423  GET_GROUP
     C|  THE FOLLOWING STATEMENTS ARE FOR DEBUGGING PURPOSES ONLY             | 424  GET_GROUP
     C|  OUT 'E_LINE="' || E_LINE || '"';                                     | 425  GET_GROUP
     C|  OUT 'M_LINE="' || M_LINE || '"';                                     | 426  GET_GROUP
     C|  OUT 'S_LINE="' || S_LINE || '"';                                     | 427  GET_GROUP
 297 M|  CLOSE GET_GROUP;                                                     | 428  STREAM
     C|                                                                       | 429  STREAM
     C|                                                                       | 430  STREAM
 298 M|  CHOP:                                                                | 431  STREAM
 298 M|      PROCEDURE;                                                       | 432  CHOP
 299 M|  INDX = INDX + 1;                                                     | 433  CHOP
 300 M|  IF INDX = TEXT_LIMIT THEN DO;                                        | 434  CHOP
     C|   OUT OF DATA, GET MORE.                                              | 435  CHOP
 301 M|  E_INDICATOR$1 = E_INDICATOR$TEXT_LIMIT;                              | 436  CHOP
 302 M|  S_INDICATOR$1 = S_INDICATOR$TEXT_LIMIT;                              | 437  CHOP
 303 M|  CALL GET_GROUP;                                                      | 438  CHOP
 304 M|  INDX = 1;                                                            | 439  CHOP
 305 M|  END;                                                                 | 440  CHOP
```

```
 STMT                              SOURCE                                    LINE CURRENT SCOPE
  306 M|  CLOSE CHOP;                                                       | 441  STREAM
      C|                                                                    | 442  STREAM
      C|                                                                    | 443  STREAM
  307 M|  STACK:                                                            | 444  STREAM
  307 M|       PROCEDURE(TYPE,INDICATOR,LIN) ASSIGN                         | 445  STACK
  307 M|       (IND,STACK,PP);                                              | 446  STACK
  308 M|  DECLARE INTEGER, TYPE, PP;                                        | 447  STACK
  309 M|  DECLARE CHARACTER(*), LIN, STACK;                                 | 448  STACK
  310 M|  DECLARE ARRAY(256) INTEGER, INDICATOR, IND;                       | 449  STACK
  311 M|  IF PP < 1 THEN GO TO NOT_MULTIPLE;                                | 450  STACK
  312 M|  IF LIN$INDX = ' ' THEN DO;                                        | 451  STACK
  313 M|  IF STACK$PP = ' ' THEN                                            | 452  STACK
  313 M|       IND$PP = IND$PP +1;                                          | 453  STACK
  314 M|  ELSE GO TO NOT_MULTIPLE;                                          | 454  STACK
  315 M|  END;                                                              | 455  STACK
  316 M|  ELSE DO;                                                          | 456  STACK
  317 M|  NOT_MULTIPLE:                                                     | 457  STACK
  317 M|  PP = PP + 1;                                                      | 458  STACK
  318 M|  IF PP > 256 THEN DO CASE TYPE+1;                                  | 459  STACK
  319 M|  CALL ERRORS('EXPONENT STRING OVER FLOW',3);                       | 460  STACK CASE 1
  320 M|  CALL ERRORS('SUBSCRIPT STRING OVER FLOW',3);                      | 461  STACK CASE 2
  321 M|  END; /*OF DO CASE*/                                               | 462  STACK CASE 3
  322 M|  STACK = STACK || LIN$INDX;                                        | 463  STACK
  323 M|  IND$PP = INDICATOR$INDX;                                          | 464  STACK
  324 M|  END;                                                              | 465  STACK
  325 M|  CLOSE STACK;                                                      | 466  STREAM
      C|                                                                    | 467  STREAM
      C|                                                                    | 468  STREAM
  326 M|  BUILD_XSCRIPTS:                                                   | 469  STREAM
```

```
STMT                           SOURCE                                    LINE CURRENT SCOPE
 326 M|      PROCEDURE;                                                | 470  BUILD_XSCRIPTS
 327 M|  E_STACK, S_STACK = '';                                        | 471  BUILD_XSCRIPTS
 328 M|  E_BLANKS, S_BLANKS = -1;                                      | 472  BUILD_XSCRIPTS
 329 M|  EP, SP = 0;                                                   | 473  BUILD_XSCRIPTS
 330 M|  CHECK_M:                                                      | 474  BUILD_XSCRIPTS
 330 M|  IF M_LINE$INDX = ' ' THEN                                     | 475  BUILD_XSCRIPTS
 330 M|  PROCESS:                                                      | 476  BUILD_XSCRIPTS
 330 M|  DO;                                                           | 477  BUILD_XSCRIPTS
 331 M|  CALL STACK(0,E_INDICATOR,E_LINE) ASSIGN                       | 478  BUILD_XSCRIPTS
 331 M|      (E_IND,E_STACK,EP);                                       | 479  BUILD_XSCRIPTS
 332 M|  CALL STACK(1,S_INDICATOR,S_LINE) ASSIGN                       | 480  BUILD_XSCRIPTS
 332 M|      (S_IND,S_STACK,SP);                                       | 481  BUILD_XSCRIPTS
 333 M|  CALL CHOP;                                                    | 482  BUILD_XSCRIPTS
 334 M|  GO TO CHECK_M;                                                | 483  BUILD_XSCRIPTS
 335 M|  END;                                                          | 484  BUILD_XSCRIPTS
 336 M|  ELSE /* A NON BLANK ON S-LIN*/                                 | 485  BUILD_XSCRIPTS
 336 M|  DO;                                                           | 486  BUILD_XSCRIPTS
 337 M|  IFF M_COMMENT THEN DO;                                        | 487  BUILD_XSCRIPTS
 338 M|  IF M_LINE$INDX = '*' THEN                                     | 488  BUILD_XSCRIPTS
 338 M|  IF M_LINE$(INDX +1) = '/' THEN DO;                            | 489  BUILD_XSCRIPTS
 339 M|  M_COMMENT = FALSE;                                            | 490  BUILD_XSCRIPTS
 340 M|  M_LINE$(INDX +1) = ' ';                                       | 491  BUILD_XSCRIPTS
 341 M|  END;                                                          | 492  BUILD_XSCRIPTS
 342 M|  GO TO PROCESS;                                                | 493  BUILD_XSCRIPTS
 343 M|  END;                                                          | 494  BUILD_XSCRIPTS
 344 M|  IF M_LINE$INDX = '/' THEN                                     | 495  BUILD_XSCRIPTS
 344 M|  IF M_LINE$(INDX +1) = '*' THEN DO;                            | 496  BUILD_XSCRIPTS
 345 M|  M_COMMENT = TRUE;                                             | 497  BUILD_XSCRIPTS
```

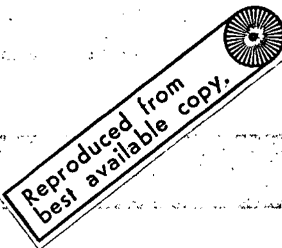
Reproduced from best available copy.

```
 STMT                              SOURCE                                        LINE CURRENT SCOPE

  346 M|  M_LINE$(INDX +1) = ' ';                                                | 498  BUILD_XSCRIPTS
  347 M|  GO TO PROCESS;                                                         | 499  BUILD_XSCRIPTS
  348 M|  END;                                                                   | 500  BUILD_XSCRIPTS
  349 M|  IF S_STACK$SP = ' ' THEN DO;                                           | 501  BUILD_XSCRIPTS
  350 M|  IF SP > 1 THEN S_STACK = S_STACK$(1 TO SP-1);                          | 502  BUILD_XSCRIPTS
  351 M|  ELSE S_STACK = '';                                                     | 503  BUILD_XSCRIPTS
  352 M|  S_BLANKS = S_IND$SP;                                                   | 504  BUILD_XSCRIPTS
  353 M|  END;                                                                   | 505  BUILD_XSCRIPTS
  354 M|  IF E_STACK$EP = ' ' THEN DO;                                           | 506  BUILD_XSCRIPTS
  355 M|  IF EP > 1 THEN E_STACK = E_STACK$(1 TO EP-1);                          | 507  BUILD_XSCRIPTS
  356 M|  ELSE E_STACK = '';                                                     | 508  BUILD_XSCRIPTS
  357 M|  E_BLANKS = E_IND$EP;                                                   | 509  BUILD_XSCRIPTS
  358 M|  END;                                                                   | 510  BUILD_XSCRIPTS
  359 M|  IF E_BLANKS >= S_BLANKS THEN                                           | 511  BUILD_XSCRIPTS
  359 M|       M_BLANKS = S_BLANKS;                                              | 512  BUILD_XSCRIPTS
  360 M|  ELSE M_BLANKS = E_BLANKS;                                              | 513  BUILD_XSCRIPTS
  361 M|  END;                                                                   | 514  BUILD_XSCRIPTS
      C|  THE FOLLOWING STATEMENTS ARE FOR DEBUGGING PURPOSES ONLY               | 515  BUILD_XSCRIPTS
      C|  OUT 'E_STACK="' || E_STACK || '"';                                     | 516  BUILD_XSCRIPTS
      C|  OUT 'S_STACK="' || S_STACK || '"';                                     | 517  BUILD_XSCRIPTS
  362 M|  CLOSE BUILD_XSCRIPTS;                                                  | 518  STREAM
      C|                                                                         | 519  STREAM
      C|                                                                         | 520  STREAM
  363 M|  STREAM_START:                                                          | 521  STREAM
  363 M|  IFF FIRST_CALL_TO_STREAM THEN DO;                                      | 522  STREAM
  364 M|  TIME1 = TIME;                                                          | 523  STREAM
  365 M|  FIRST_CALL_TO_STREAM = FALSE;                                          | 524  STREAM
  366 M|  AGAIN:                                                                 | 525  STREAM
  366 M|  CALL READ_CARD;                                                        | 526  STREAM
```

```
STMT                          SOURCE                                    LINE CURRENT SCOPE
 367 M| IF_NOT CRDER_CK('C') THEN DO;                                  | 527  STREAM
 368 M| CALL ERRORS(ORDER_ERR,1);                                      | 528  STREAM
 369 M| GO TO AGAIN;                                                   | 529  STREAM
 370 M| END;                                                           | 530  STREAM
 371 M| TEXT_LIMIT = LENGTH(CURRENT_CARD) ;                            | 531  STREAM
 372 M| RETURNING_M = TRUE;                                            | 532  STREAM
 373 M| M_LINE, E_LINE, S_LINE = ' ';                                  | 533  STREAM
 374 M|  INDX = 1;                                                     | 534  STREAM
 375 M| CALL GET_GROUP;                                                | 535  STREAM
 376 M| M_BLANKS = -1;                                                 | 536  STREAM
 377 M| INDX = 2;                                                      | 537  STREAM
 378 M|  II = 1;                                                       | 538  STREAM
 379 M| END;                                                           | 539  STREAM
 380 M| IFF MACRO_FOUND THEN DO;                                       | 540  STREAM
 381 M| IF MACRO_POINT < MACRO_LIMIT THEN DO;                          | 541  STREAM
 382 M| NEXT_CHAR =                                                    | 542  STREAM
 382 M|      MACRO_STREAM$MACRO_POINT;                                 | 543  STREAM
 383 M| MACRO_POINT = MACRO_POINT +1;                                  | 544  STREAM
 384 M| RETURN;                                                        | 545  STREAM
 385 M| END;                                                           | 546  STREAM
 386 M| IF SAVE_BLANK_COUNT >= 0 THEN DO;                              | 547  STREAM
 387 M| NEXT_CHAR = ' ';                                               | 548  STREAM
 388 M| BLANK_COUNT = SAVE_BLANK_COUNT;                                | 549  STREAM
 389 M| SAVE_BLANK_COUNT = -1;                                         | 550  STREAM
 390 M| RETURN;                                                        | 551  STREAM
 391 M| END;                                                           | 552  STREAM
 392 M| MACRO_FOUND = FALSE;                                           | 553  STREAM
 393 M| NEXT_CHAR = SAVE_NEXT_CHAR;                                    | 554  STREAM
```

```
 STMT                              SOURCE                                        LINE CURRENT SCOPE
  394 M|  OVER_PUNCH = SAVE_OVER_PUNCH;                                         | 555  STREAM
  395 M|  MACRO_STREAM = '';                                                    | 556  STREAM
  396 M|  RETURN;                                                               | 557  STREAM
  397 M|  END;                                                                  | 558  STREAM
  398 M|  BLANK_COUNT = -1;                                                     | 559  STREAM
  399 M|  STACK_CHECK:                                                          | 560  STREAM
  399 M|  DO FOR II = II TO 3;/*CHECK LIMIT*/                                   | 561  STREAM
  400 M|  IF RETURN_CHAR    NOT = 0 THEN DO;                                    | 562  STREAM
  400 S|                 II:                                                    | 563  STREAM
  401 M|  ARROW_FLAG = TRUE;                                                    | 564  STREAM
  402 M|  RETURN_CHAR     = RETURN_CHAR    -1;                                  | 565  STREAM
  402 S|             II:               II:                                      | 566  STREAM
  403 M|  NEXT_CHAR = TYPE_CHAR   ;                                             | 567  STREAM
  403 S|                       II:                                              | 568  STREAM
  404 M|  OVER_PUNCH =0;                                                        | 569  STREAM
  405 M|  RETURN;                                                               | 570  STREAM
  406 M|  END;                                                                  | 571  STREAM
  407 M|  END;                                                                  | 572  STREAM
  408 M|  II = 1;                                                               | 573  STREAM
  409 M|  IFF ARROW_FLAG THEN DO;                                               | 574  STREAM
  410 M|  ARROW_FLAG = FALSE;                                                   | 575  STREAM
  411 M|  NEXT_CHAR = SAVE_NEXT_CHAR;                                           | 576  STREAM
  412 M|  OVER_PUNCH = SAVE_OVER_PUNCH;                                         | 577  STREAM
  413 M|  BLANK_COUNT = SAVE_BLANK_COUNT;                                       | 578  STREAM
  414 M|  RETURN;                                                               | 579  STREAM
  415 M|  END;                                                                  | 580  STREAM
  416 M|  BEGINING:                                                             | 581  STREAM
  416 M|  IFF RETURNING_M THEN DO;                                              | 582  STREAM
  417 M|  IF M_BLANKS >= 0 THEN DO;                                             | 583  STREAM
```

```
STMT                          SOURCE                                          LINE CURRENT SCOPE
 418 M|  NEXT_CHAR = ' ';                                                      | 584  STREAM
 419 M|  ARROW = - LAST_E_IND;                                                 | 585  STREAM
 420 M|  LAST_E_IND = 0;                                                       | 586  STREAM
 421 M|  BLANK_COUNT = M_BLANKS;                                               | 587  STREAM
 422 M|  M_BLANKS = -1;                                                        | 588  STREAM
 423 M|  GO TO FOUND_CHAR;                                                     | 589  STREAM
 424 M|  END;                                                                  | 590  STREAM
 425 M|  IF M_LINE$INDX NOT = ' ' THEN DO;                                     | 591  STREAM
 426 M|  IF M_LINE$INDX = '/' THEN                                             | 592  STREAM
 426 M|  IF M_LINE$(INDX +1) = '*' THEN DO;                                    | 593  STREAM
     C|  HERE IS A START OF A COMMENT                                          | 594  STREAM
 427 M|  M_COMMENT = TRUE;                                                     | 595  STREAM
 428 M|  M_LINE$INDX = ' ';                                                    | 596  STREAM
 429 M|  M_LINE$(INDX +1) = ' ';                                               | 597  STREAM
 430 M|  GO TO BLANK;                                                          | 598  STREAM
 431 M|  END;                                                                  | 599  STREAM
 432 M|  IF E_COUNT > 0 THEN DO;                                               | 600  STREAM
 433 M|  IF E_LINE$INDX NOT = ' ' THEN DO;                                     | 601  STREAM
 434 M|  IF E_INDICATOR$INDX NOT = 1 THEN                                      | 602  STREAM
 434 M|  CALL ERRORS('E-LINE OVERLAPS M-LINE',1);                              | 603  STREAM
 435 M|  ELSE OVER_PUNCH = E_LINE$INDX;                                        | 604  STREAM
 436 M|  END;                                                                  | 605  STREAM
 437 M|  ELSE OVER_PUNCH = 0;                                                  | 606  STREAM
 438 M|  END;                                                                  | 607  STREAM
 439 M|  ELSE OVER_PUNCH = 0;                                                  | 608  STREAM
 440 M|  IF S_COUNT > 0 THEN                                                   | 609  STREAM
 440 M|  IF S_LINE$INDX NOT = ' ' THEN                                         | 610  STREAM
 440 M|  CALL ERRORS('S-LINE OVERLAPS M-LINE',1);                              | 611  STREAM
```

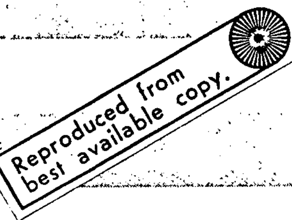


```
STMT                              SOURCE                                      LINE CURRENT SCOPE
 441 M|  ARROW = - LAST_E_IND;                                              | 612  STREAM
 442 M|  LAST_E_IND = 0;                                                    | 613  STREAM
 443 M|  NEXT_CHAR = M_LINE$INDX;                                           | 614  STREAM
 444 M|  CALL CHOP;                                                         | 615  STREAM
 445 M|  GO TO FOUND_CHAR;                                                  | 616  STREAM
 446 M|  END;                                                               | 617  STREAM
 447 M|  ELSE                                                               | 618  STREAM
 447 M|  BLANK:                                                             | 619  STREAM
 447 M|  DO;                                                                | 620  STREAM
 448 M|  CALL BUILD_XSCRIPTS;                                               | 621  STREAM
 449 M|  OVER_PUNCH = 0;                                                    | 622  STREAM
 450 M|  RETURNING_M = FALSE;                                               | 623  STREAM
 451 M|  LAST_S_IND = 0;                                                    | 624  STREAM
 452 M|  RETURNING_S = TRUE;                                                | 625  STREAM
 453 M|  POINTER = 1; /*THIS ONE MAY NOT BE NEEDED*/                        | 626  STREAM
 454 M|  END;                                                               | 627  STREAM
 455 M|  END;                                                               | 628  STREAM
 456 M|  IFF RETURNING_S THEN DO;                                           | 629  STREAM
 457 M|  IF LENGTH(S_STACK) > 0 &                                           | 630  STREAM
 457 M|       POINTER <= LENGTH(S_STACK) THEN DO;                           | 631  STREAM
 458 M|  IF S_STACK$POINTER = ' ' THEN DO;                                  | 632  STREAM
 459 M|  IF S_IND$POINTER >= 0 THEN                                         | 633  STREAM
     C|  MORE LEFT                                                          | 634  STREAM
 459 M|        DO;                                                          | 635  STREAM
 460 M|  NEXT_CHAR = ' ';                                                   | 636  STREAM
 461 M|  BLANK_COUNT = S_IND$POINTER;                                       | 637  STREAM
 462 M|  POINTER = POINTER + 1;                                             | 638  STREAM
 463 M|  ARROW = LAST_S_IND - S_IND$POINTER;                                | 639  STREAM
```

```
STMT                              SOURCE                                LINE CURRENT SCOPE
 464 M| LAST_S_IND = S_IND$POINTER;                                      | 640  STREAM
 465 M| END;                                                             | 641  STREAM
 466 M| END;                                                             | 642  STREAM
 467 M| ELSE DO; /*A NON BLANK*/                                         | 643  STREAM
 468 M| NEXT_CHAR = S_STACK$POINTER;                                     | 644  STREAM
 469 M| ARROW = LAST_S_IND - S_IND$POINTER;                              | 645  STREAM
 470 M| LAST_S_IND = S_IND$POINTER;                                      | 646  STREAM
 471 M| POINTER = POINTER + 1;                                           | 647  STREAM
 472 M| END;                                                             | 648  STREAM
 473 M| GO TO FOUND_CHAR;                                                | 649  STREAM
 474 M| END;                                                             | 650  STREAM
 475 M| ELSE DO; /* CAN NOT RETURN*/                                     | 651  STREAM
 476 M| RETURNING_S  = FALSE;                                            | 652  STREAM
 477 M| RETURNING_E = TRUE;                                              | 653  STREAM
 478 M| LAST_E_IND = -LAST_S_IND;                                        | 654  STREAM
 479 M| POINTER = 1;                                                     | 655  STREAM
 480 M| END;                                                             | 656  STREAM
 481 M| END;                                                             | 657  STREAM
 482 M| IFF RETURNING_E THEN DO;                                         | 658  STREAM
 483 M| IF LENGTH(E_STACK) > 0 &                                         | 659  STREAM
 483 M| POINTER <= LENGTH(E_STACK) THEN DO;                              | 660  STREAM
 484 M| IF E_STACK$POINTER = ' ' THEN DO;                                | 661  STREAM
 485 M| IF E_IND$POINTER >= 0 THEN                                       | 662  STREAM
     C| MORE TO GO                                                       | 663  STREAM
 485 M|      DO;                                                         | 664  STREAM
 486 M| NEXT_CHAR = ' ';                                                 | 665  STREAM
 487 M| BLANK_COUNT = E_IND$POINTER;                                     | 666  STREAM
 488 M| POINTER = POINTER + 1;                                           | 667  STREAM
```

```
 STMT                          SOURCE                                  LINE CURRENT SCOPE
  489 M|  ARROW = E_IND$POINTER - LAST_E_IND;                        | 668  STREAM
  490 M|  LAST_E_IND = E_IND$POINTER;                                | 669  STREAM
  491 M|  END;                                                       | 670  STREAM
  492 M|  END;                                                       | 671  STREAM
  493 M|  ELSE                                                       | 672  STREAM
      C|  A NON BLANK                                                | 673  STREAM
  493 M|  DO;                                                        | 674  STREAM
  494 M|  NEXT_CHAR = E_STACK$POINTER;                               | 675  STREAM
  495 M|  ARROW = E_IND$POINTER - LAST_E_IND;                        | 676  STREAM
  496 M|  LAST_E_IND = E_IND$POINTER;                                | 677  STREAM
  497 M|  POINTER = POINTER + 1;                                     | 678  STREAM
  498 M|  END;                                                       | 679  STREAM
  499 M|  GO TO FOUND_CHAR;                                          | 680  STREAM
  500 M|  END;                                                       | 681  STREAM
  501 M|  ELSE                                                       | 682  STREAM
      C|  CAN NOT RETURN                                             | 683  STREAM
  501 M|      DO;                                                    | 684  STREAM
  502 M|  RETURNING_E = FALSE;                                       | 685  STREAM
  503 M|  RETURNING_M = TRUE;                                        | 686  STREAM
  504 M|  END;                                                       | 687  STREAM
  505 M|  END;                                                       | 688  STREAM
  506 M|  GO TO BEGINING;                                            | 689  STREAM
  507 M|  FOUND_CHAR:                                                | 690  STREAM
  507 M|  IF ARROW NOT = 0 THEN DO;                                  | 691  STREAM
  508 M|  OLD_LEVEL = NEW_LEVEL;                                     | 692  STREAM
  509 M|  NEW_LEVEL = NEW_LEVEL + ARROW;                              | 693  STREAM
  510 M|  SAVE_OVER_PUNCH = OVER_PUNCH;                              | 694  STREAM
  511 M|  SAVE_NEXT_CHAR = NEXT_CHAR;                                | 695  STREAM
```

```
STMT                              SOURCE                                         LINE CURRENT SCOPE
 512 M| SAVE_BLANK_COUNT = BLANK_COUNT;                                          | 696  STREAM
 513 M| IF OLD_LEVEL > 0 THEN DO;                                                | 697  STREAM
 514 M| IF ARROW < 0 THEN                                                        | 698  STREAM
 514 M|      CALL STACK_RETURN_CHAR(-ARROW,')');                                 | 699  STREAM
 515 M| ELSE                                                                     | 700  STREAM
 515 M| EXPONENT:                                                                | 701  STREAM
 515 M| DO;                                                                      | 702  STREAM
 516 M| IF ARROW > 1 THEN                                                        | 703  STREAM
 516 M| CALL ERRORS(E_LINE_ERR,1);                                               | 704  STREAM
 517 M| CALL STACK_RETURN_CHAR(2,'*');                                           | 705  STREAM
 518 M| CALL STACK_RETURN_CHAR(ARROW,'(');                                       | 706  STREAM
 519 M| END;                                                                     | 707  STREAM
 520 M| END;                                                                     | 708  STREAM
 521 M| ELSE IF OLD_LEVEL = 0 THEN DO;                                           | 709  STREAM
 522 M| IF ARROW < 0 THEN                                                        | 710  STREAM
 522 M| SUBS:                                                                    | 711  STREAM
 522 M| DO;                                                                      | 712  STREAM
 523 M| IF ARROW < -1 THEN CALL ERRORS(                                          | 713  STREAM
 523 M|      'S-LINE CHARACTER MORE THAN 1 LINE BELOW PRECEDING CHARACTER',1);   | 714  STREAM
 524 M| CALL STACK_RETURN_CHAR(1,'$');                                           | 715  STREAM
 525 M| CALL STACK_RETURN_CHAR(-ARROW,'(');                                      | 716  STREAM
 526 M| END;                                                                     | 717  STREAM
 527 M| ELSE GO TO EXPONENT;                                                     | 718  STREAM
 528 M| END;                                                                     | 719  STREAM
 529 M| ELSE /*OLD < 0*/ DO;                                                     | 720  STREAM
 530 M| IF ARROW < 0 THEN GO TO SUBS;                                            | 721  STREAM
 531 M| IF NEW_LEVEL <= 0 THEN                                                   | 722  STREAM
 531 M| CALL STACK_RETURN_CHAR(ARROW,')');                                       | 723  STREAM
```



```
STMT                              SOURCE                                  LINE CURRENT SCOPE
 532 M|  ELSE DO;                                                        | 724  STREAM
 533 M|  CALL STACK_RETURN_CHAR(-OLD_LEVEL,')');                         | 725  STREAM
 534 M|  IF NEW_LEVEL > 1 THEN CALL ERRORS(E_LINE_ERR,1);                | 726  STREAM
 535 M|  CALL STACK_RETURN_CHAR(2,'*');                                  | 727  STREAM
 536 M|  CALL STACK_RETURN_CHAR(NEW_LEVEL,'(');                          | 728  STREAM
 537 M|  END;                                                            | 729  STREAM
 538 M|  END;                                                            | 730  STREAM
 539 M|  ARROW = 0;                                                      | 731  STREAM
 540 M|  GO TO STACK_CHECK;                                              | 732  STREAM
 541 M|  END;                                                            | 733  STREAM
 542 M|  RETURN;                                                         | 734  STREAM
 543 M|  CLOSE STREAM;                                                   | 735  HALINHAL
     C|                                                                  | 736  HALINHAL
     C|                                                                  | 737  HALINHAL
     C|   PROGRAM TO TEST THE STREAM PROCEDURE                           | 738  HALINHAL
     C|                                                                  | 739  HALINHAL
 544 M|  MAIN_PROGRAM:                                                   | 740  HALINHAL
 544 M|  WRITE(6) 'BEGIN TEST OF HAL IN HAL';                            | 741  HALINHAL
 545 M|  BUILT, BUILT_UP, BUILT_TOKEN, BUILT_C_P='';                     | 742  HALINHAL
 546 M|  MAIN_LOOP: DO FOREVER;                                          | 743  HALINHAL
 547 M|  CALL STREAM;                                                    | 744  HALINHAL
 548 M|  IF NEXT_CHAR = ' ' THEN DO;                                     | 745  HALINHAL
 549 M|     IFF BLANK_FLAG THEN GO TO MAIN_LOOP;                         | 746  HALINHAL
 550 M|     ELSE BLANK_FLAG = TRUE;                                      | 747  HALINHAL
 551 M|     END;                                                         | 748  HALINHAL
 552 M|  ELSE BLANK_FLAG = FALSE;                                        | 749  HALINHAL
 553 M|  BUILT = BUILT || NEXT_CHAR;                                     | 750  HALINHAL
 554 M|  IF OVER_PUNCH = 'O' THEN DO;                                    | 751  HALINHAL
     C|      OUT 'STREAM RETURNS "' || NEXT_CHAR || '"' || ' - BLANKS='  | 752  HALINHAL
     C|      || BLANK_COUNT;                                             | 753  HALINHAL
```

```
 STMT                                  SOURCE                                              LINE CURRENT SCOPE
  555 M|     BUILT_UP = BUILT_UP || ' ';                                                 | 754  HALINHAL
  556 M|     END;                                                                        | 755  HALINHAL
  557 M|  ELSE DO;                                                                       | 756  HALINHAL
      C|     OUT 'STREAM RETURNS "' || NEXT_CHAR || '" - OVER PUNCH "'                   | 757  HALINHAL
      C|     || OVER_PUNCH || '"' || ' - BLANKS=' || BLANK_COUNT;                        | 758  HALINHAL
  558 M|     BUILT_UP = BUILT_UP || OVER_PUNCH;                                          | 759  HALINHAL
  559 M|  END;                                                                           | 760  HALINHAL
  560 M|  IF NEXT_CHAR NOT < 'A' THEN DO;                                                | 761  HALINHAL
  561 M|     BUILT_TOKEN = BUILT_TOKEN || NEXT_CHAR;                                     | 762  HALINHAL
  562 M|     IF OVER_PUNCH NOT = '0' THEN BUILT_O_P = OVER_PUNCH;                        | 763  HALINHAL
  563 M|  END;                                                                           | 764  HALINHAL
  564 M|  ELSE DO;                                                                       | 765  HALINHAL
  565 M|    IF LENGTH(BUILT_TOKEN) NOT = 0 THEN DO;                                      | 766  HALINHAL
  566 M|       IF BUILT_O_P = '' THEN WRITE(6) '***TOKEN=' || BUILT_TOKEN;               | 767  HALINHAL
  567 M|        ELSE WRITE(6) '***TOKEN=' || BUILT_TOKEN || ',MARKER=' || BUILT_O_P;     | 768  HALINHAL
  568 M|       BUILT_TOKEN, BUILT_O_P = '';                                              | 769  HALINHAL
  569 M|    END;                                                                         | 770  HALINHAL
  570 M|    IF NEXT_CHAR NOT = ' ' THEN WRITE(6) '***TOKEN=' || NEXT_CHAR;               | 771  HALINHAL
  571 M|    ELSE WRITE(6) '**BLANKS=' || BLANK_COUNT + 1;                                | 772  HALINHAL
  572 M|  END;                                                                           | 773  HALINHAL
  573 M|  IF NEXT_CHAR = ';' THEN DO;                                                    | 774  HALINHAL
  574 M|     WRITE(6) SKIP(2), '***OVER***' || BUILT_UP;                                 | 775  HALINHAL
  575 M|     WRITE(6) '***MAIN***' || BUILT, SKIP(2);                                    | 776  HALINHAL
  576 M|     BUILT, BUILT_UP = '';                                                       | 777  HALINHAL
  577 M|  END;                                                                           | 778  HALINHAL
  578 M|   IF NEXT_CHAR = '?' THEN GO TO DISASTER;                                       | 779  HALINHAL
  579 M|   END;  /* DO FOREVER */                                                        | 780  HALINHAL
  580 M|  DISASTER:                                                                      | 781  HALINHAL
```

```
STMT                               SOURCE                                   LINE CURRENT SCOPE
 580 M|  CALL PRINT_SUMMARY;                                                | 782  HALINHAL
 581 M|  WRITE(6) ' THIS TEST IS NOW COMPLETE.';                            | 783  HALINHAL
 582 M|  CLOSE HALINHAL;                                                    | 784
     C|                                                                     | 785
     C|     A    B    C    D    E                                           | 786
```

SYMBOL TABLE LISTING:

LOC	NAME	TYPE	CLASS	LENGTH	PRECISION	NEST	OUTER_LEVEL	FLAGS	SYT_PTR	ARRAY
1	HALINHAL	PROG	LABEL	0	0	0	0	40	2	
2	R	0	REPL	0	0	1	0	8000	1	
3	FALSE	0	REPL	0	0	1	0	8000	2	
4	TRUE	0	REPL	0	0	1	0	8000	3	
5	BIT_1	0	REPL	0	0	1	0	8000	4	
6	IF_NOT	0	REPL	0	0	1	0	8000	5	
7	IFF	0	REPL	0	0	1	0	8000	6	
8	CUT	0	REPL	0	0	1	0	8000	7	
9	FOREVER	0	REPL	0	0	1	0	8000	8	
10	XTO	CHAR	VAR	70	FIXED	1	0	8A0B	0	
11	BUILT	CHAR	VAR	255	VARYING	1	0	820B	0	
12	BUILT_UP	CHAR	VAR	255	VARYING	1	0	820B	0	
13	BUILT_TOKEN	CHAR	VAR	255	VARYING	1	0	820B	0	
14	BUILT_C_P	CHAR	VAR	255	VARYING	1	0	820B	0	
15	BLANK_FLAG	INT	VAR	0	0	1	0	8A0B	0	
16	ERROR_COUNT	INT	VAR	0	0	1	0	8A0B	0	
17	MACRO_POINT	INT	VAR	0	0	1	0	8A0B	0	
18	MACRO_LIMIT	INT	VAR	0	0	1	0	8A0B	0	
19	OLD_LEVEL	INT	VAR	0	0	1	0	8A0B	0	
20	NEW_LEVEL	INT	VAR	0	0	1	0	8A0B	0	
21	MAX_SEVERITY	INT	VAR	0	0	1	0	8A0B	0	
22	STATEMENT_SEVERITY	INT	VAR	0	0	1	0	8A0B	0	
23	SAVE_SEVERITY	INT	ARRAY	0	0	1	0	8A0B	0	100
24	SAVE_LINE	INT	ARRAY	0	0	1	0	8A0B	0	100
25	PREVIOUS_ERROR	INT	VAR	0	0	1	0	8A0B	0	
26	I	INT	VAR	0	0	1	0	8A0B	0	
27	COMPILING	INT	VAR	0	0	1	0	8A0B	0	
28	TIME1	INT	VAR	0	0	1	0	820B	0	
29	TIME2	INT	VAR	0	0	1	0	820B	0	
30	MACRO_STREAM	CHAR	VAR	255	VARYING	1	0	820B	0	
31	MACRO_FOUND	INT	VAR	0	0	1	0	8A0B	0	
32	CARD_COUNT	INT	VAR	0	0	1	0	8A0B	0	
33	DISASTER	STMT	LABEL	0	0	1	0	8040	0	
34	NEXT_CHAR	CHAR	VAR	1	FIXED	1	0	820B	0	
35	OVER_PUNCH	CHAR	VAR	1	FIXED	1	0	820B	0	
36	BLANK_COUNT	INT	VAR	0	0	1	0	8A0B	0	
37	CONTROL	BIT	ARRAY	32	0	1	0	820B	0	10
38	TOGGLES	CHAR	VAR	10	FIXED	1	0	8A0B	0	
39	CARD_TYPE	INT	FUNC	0	0	1	0	8040	40	
40	SELECT	CHAR	VAR	1	FIXED	2	0	C60B	0	
41	CHAR_INDEX	INT	FUNC	0	0	1	0	8040	42	
42	STRING	CHAR	VAR	**	FIXED	2	0	860B	0	
43	PATTERN	CHAR	VAR	**	FIXED	2	0	860B	0	
44	I	INT	VAR	0	0	2	0	820B	0	
45	J	INT	VAR	0	0	2	0	820B	0	
46	K	INT	VAR	0	0	2	0	C20B	0	
47	PAD	CHAR	FUNC	255	VARYING	1	0	8040	48	
48	STRING	CHAR	VAR	**	FIXED	2	0	860B	0	
49	WIDTH	INT	VAR	0	0	2	0	860B	0	
50	TEMP_STRING	CHAR	VAR	255	VARYING	2	0	820B	0	
51	L	INT	VAR	0	0	2	0	C20B	0	
52	I_FORMAT	CHAR	FUNC	**	FIXED	1	0	8040	53	

Reproduced from best available copy.

LOC	NAME	TYPE	CLASS	LENGTH	PRECISION	NEST	OUTER_LEVEL	FLAGS	SYT_PTR	ARRAY
53	NUMBER	INT	VAR	0	0	2	0	860B	0	
54	WIDTH	INT	VAR	0	0	2	0	860B	0	
55	STRING	CHAR	VAR	255	VARYING	2	0	820B	0	
56	L	INT	VAR	0	0	2	0	C20B	0	
57	ERRORS	PROC	LABEL	0	0	1	0	8040	58	
58	MESSAGE	CHAR	VAR	**	FIXED	2	0	860B	0	
59	SEVERITY	INT	VAR	0	0	2	0	860B	0	
60	MSG	CHAR	VAR	255	VARYING	2	0	820B	0	
61	DISASTER	I-ST	LABEL	0	0	2	0	C040	33	
62	PRINT_SUMMARY	PROC	LABEL	0	0	1	0	8040	0	
63	STREAM	PROC	LABEL	0	0	1	0	8040	64	
64	E_LINE	CHAR	VAR	255	VARYING	2	0	820B	0	
65	S_LINE	CHAR	VAR	255	VARYING	2	0	820B	0	
66	M_LINE	CHAR	VAR	255	VARYING	2	0	820B	0	
67	SAVE_CARD	CHAR	VAR	80	FIXED	2	0	820B	0	
68	CURRENT_CARD	CHAR	VAR	80	FIXED	2	0	820B	0	
69	BLANKS	CHAR	VAR	128	FIXED	2	0	8A0B	0	
70	E_LINE_ERR	CHAR	VAR	50	VARYING	2	0	8A0B	0	
71	ORDER_ERR	CHAR	VAR	27	FIXED	2	0	8A0B	0	
72	PREV_CARD	CHAR	VAR	1	FIXED	2	0	820B	0	
73	SAVE_NEXT_CHAR	CHAR	VAR	1	FIXED	2	0	820B	0	
74	SAVE_OVER_PUNCH	CHAR	VAR	1	FIXED	2	0	820B	0	
75	LAST_E_IND	INT	VAR	0	0	2	0	8A0B	0	
76	LAST_S_IND	INT	VAR	0	0	2	0	8A0B	0	
77	E_BLANKS	INT	VAR	0	0	2	0	8A0B	0	
78	M_BLANKS	INT	VAR	0	0	2	0	8A0B	0	
79	S_BLANKS	INT	VAR	0	0	2	0	8A0B	0	
80	EP	INT	VAR	0	0	2	0	8A0B	0	
81	SP	INT	VAR	0	0	2	0	8A0B	0	
82	TEXT_LIMIT	INT	VAR	0	0	2	0	8A0B	0	
83	E_COUNT	INT	VAR	0	0	2	0	8A0B	0	
84	LAST_E_COUNT	INT	VAR	0	0	2	0	8A0B	0	
85	S_COUNT	INT	VAR	0	0	2	0	8A0B	0	
86	LAST_S_COUNT	INT	VAR	0	0	2	0	8A0B	0	
87	INDX	INT	VAR	0	0	2	0	8A0B	0	
88	CP	INT	VAR	0	0	2	0	8A0B	0	
89	POINTER	INT	VAR	0	0	2	0	8A0B	0	
90	ARROW	INT	VAR	0	0	2	0	8A0B	0	
91	II	INT	VAR	0	0	2	0	8A0B	0	
92	E_IND	INT	ARRAY	0	0	2	0	8A0B	0	256
93	S_IND	INT	ARRAY	0	0	2	0	8A0B	0	256
94	E_INDICATOR	INT	ARRAY	0	0	2	0	8A0B	0	256
95	S_INDICATOR	INT	ARRAY	0	0	2	0	8A0B	0	256
96	RETURNING_E	INT	VAR	0	0	2	0	8A0B	0	
97	RETURNING_M	INT	VAR	0	0	2	0	8A0B	0	
98	RETURNING_S	INT	VAR	0	0	2	0	8A0B	0	
99	END_GROUP	INT	VAR	0	0	2	0	8A0B	0	
100	M_COMMENT	INT	VAR	0	0	2	0	8A0B	0	
101	ARROW_FLAG	INT	VAR	0	0	2	0	8A0B	0	
102	FIRST_CALL_TO_STREAM	INT	VAR	0	0	2	0	8A0B	0	
103	TYPE_CHAR	CHAR	ARRAY	1	FIXED	2	0	820B	0	3
104	RETURN_CHAR	INT	ARRAY	0	0	2	0	8A0B	0	3
105	E_STACK	CHAR	VAR	255	VARYING	2	0	820B	0	
106	S_STACK	CHAR	VAR	255	VARYING	2	0	820B	0	
107	INPUT_PAD	CHAR	VAR	23	FIXED	2	0	8A0B	0	
108	STREAM_START	STMT	LABEL	0	0	2	0	8040	0	

LOC	NAME	TYPE	CLASS	LENGTH	PRECISION	NEST	OUTER_LEVEL	FLAGS	SYT_PTR	ARRAY
109	PROCESS_COMMENT	PROC	LABEL	0	0	2	0	8040	110	
110	K	CHAR	VAR	1	FIXED	3	0	820B	0	
111	J	INT	VAR	0	0	3	0	820B	0	
112	COMPLEMENT	STMT	LABEL	0	0	3	0	C040	0	
113	STACK_RETURN_CHAR	PROC	LABEL	0	0	2	0	8040	114	
114	NUMBER	INT	VAR	0	0	3	0	860B	0	
115	CHAR	CHAR	VAR	1	FIXED	3	0	860B	0	
116	I	INT	VAR	0	0	3	0	C20B	0	
117	READ_CARD	PROC	LABEL	0	0	2	0	8040	118	
118	END_OF_INPUT	INT	VAR	0	0	3	0	CA0B	0	
119	ERROR_CK	INT	FUNC	0	0	2	0	8040	120	
120	TYPE	CHAR	VAR	1	FIXED	3	0	860B	0	
121	F_CARD	STMT	LABEL	0	0	3	0	C040	0	
122	COMMENT	INT	FUNC	0	0	2	0	8040	123	
123	RETURN_TRUE	STMT	LABEL	0	0	3	0	C040	0	
124	SCAN_CARD	PROC	LABEL	0	0	2	0	8040	125	
125	TYPE	INT	VAR	0	0	3	0	860B	0	
126	COUNT	INT	VAR	0	0	3	0	860B	0	
127	LIN	CHAR	VAR	**	FIXED	3	0	822B	0	
128	INDICATOR	INT	ARRAY	0	0	3	0	822B	0	256
129	CONTINUE	STMT	LABEL	0	0	3	0	8040	0	
130	ERRORS	I-CL	LABEL	0	0	3	0	C040	57	
131	COMP	PROC	LABEL	0	0	2	0	8040	132	
132	TYPE	INT	VAR	0	0	3	0	860B	0	
133	LIN	CHAR	VAR	**	FIXED	3	0	822B	0	
134	INDICATOR	INT	ARRAY	0	0	3	0	822B	0	256
135	COUNT	INT	VAR	0	0	3	0	822B	0	
136	POINT	CHAR	VAR	1	FIXED	3	0	820B	0	
137	SCAN_CARD	I-CL	LABEL	0	0	3	0	8040	124	
138	READ_CARD	I-CL	LABEL	0	0	3	0	8040	117	
139	ERRORS	I-CL	LABEL	0	0	3	0	C040	57	
140	GET_GROUP	PROC	LABEL	0	0	2	0	8040	141	
141	LOOP	STMT	LABEL	0	0	3	0	8040	0	
142	READ_IT	STMT	LABEL	0	0	3	0	8040	0	
143	READ_CARD	I-CL	LABEL	0	0	3	0	8040	117	
144	ERRORS	I-CL	LABEL	0	0	3	0	8040	57	
145	FOUND_GROUP	STMT	LABEL	0	0	3	0	8040	0	
146	COMP	I-CL	LABEL	0	0	3	0	8040	131	
147	PROCESS_COMMENT	I-CL	LABEL	0	0	3	0	C040	109	
148	CHOP	PROC	LABEL	0	0	2	0	8040	149	
149	GET_GROUP	I-CL	LABEL	0	0	3	0	C040	140	
150	STACK	PROC	LABEL	0	0	2	0	8040	151	
151	TYPE	INT	VAR	0	0	3	0	860B	0	
152	INDICATOR	INT	ARRAY	0	0	3	0	860B	0	256
153	LIN	CHAR	VAR	**	FIXED	3	0	860B	0	
154	IND	INT	ARRAY	0	0	3	0	822B	0	256
155	STACK	CHAR	VAR	**	FIXED	3	0	822B	0	
156	PP	INT	VAR	0	0	3	0	822B	0	
157	NOT_MULTIPLE	STMT	LABEL	0	0	3	0	8040	0	
158	ERRORS	I-CL	LABEL	0	0	3	0	C040	57	
159	BUILD_XSCRIPTS	PROC	LABEL	0	0	2	0	8040	160	
160	CHECK_M	STMT	LABEL	0	0	3	0	8040	0	
161	PROCESS	STMT	LABEL	0	0	3	0	8040	0	
162	STACK	I-CL	LABEL	0	0	3	0	8040	150	
163	CHOP	I-CL	LABEL	0	0	3	0	C040	148	
164	AGAIN	STMT	LABEL	0	0	2	0	8040	0	

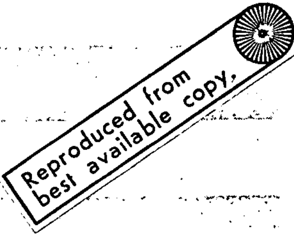
LOC	NAME	TYPE	CLASS	LENGTH	PRECISION	NEST	OUTER_LEVEL	FLAGS	SYT_PTR	ARRAY
165	ERRORS	I-CL	LABEL	0	0	2	0	8040	57	
166	*SAVE_BLANK_COUNT	INT	VAR	0	0	2	0	8218	0	
167	STACK_CHECK	STMT	LABEL	0	0	2	0	8040	0	
168	BEGINING	STMT	LABEL	0	0	2	0	8040	0	
169	FOUND_CHAR	STMT	LABEL	0	0	2	0	8040	0	
170	BLANK	STMT	LABEL	0	0	2	0	8040	0	
171	EXPONENT	STMT	LABEL	0	0	2	0	8040	0	
172	SUBS	STMT	LABEL	0	0	2	0	C040	0	
173	MAIN_PROGRAM	STMT	LABEL	0	0	1	0	8040	0	
174	MAIN_LOOP	STMT	LABEL	0	0	1	0	C040	0	

CROSS REFERENCE LISTING:

```
LOC     FLAGS & STATEMENT NUMBER (FLAG KEY: 4 = ASSIGNMENT, 2 = REFERENCE, 1 = SUBSCRIPT USE)

  1  0 0001
  2  0 0002  2 0003  2 0004  2 0005  2 0006  2 0007  2 0008  2 0009
  3  0 0003  2 0013  2 0018  2 0075  2 0080  2 0109  2 0128  2 0146  2 0162  2 0163  2 0167  2 0172  2 0174  2 0178  2 0181  2 0184
     2 0200  2 0282  2 0339  2 0365  2 0392  2 0410  2 0450  2 0476  2 0502  2 0552
  4  0 0004  2 0015  2 0110  2 0127  2 0166  2 0168  2 0173  2 0177  2 0180  2 0185  2 0195  2 0345  2 0372  2 0401  2 0427  2 0452
     2 0477  2 0503  2 0550
  5  0 0005  2 0013  2 0015  2 0018  2 0109  2 0110  2 0146  2 0158  2 0190
  6  0 0006  2 0233  2 0253  2 0367
  7  0 0007  2 0075  2 0080  2 0147  2 0208  2 0257  2 0337  2 0363  2 0390  2 0409  2 0416  2 0456  2 0482  2 0549
  8  0 0008  NOT REFERENCED
  9  0 0009  2 0229  2 0546
 10  0 0011  2 0050  2 0059
 11  0 0012  4 0545  6 0553  2 0575  4 0576
 12  0 0012  4 0545  6 0555  6 0558  2 0574  4 0576
 13  0 0012  4 0545  6 0561  2 0565  2 0566  2 0567  4 0568
 14  0 0012  4 0545  4 0562  2 0566  2 0567  4 0568
 15  0 0013  2 0549  4 0550  4 0552
 16  0 0014  6 0066  2 0067  2 0069  1 0070  1 0071  2 0073  2 0074  2 0085  2 0086  2 0087  2 0089
 17  0 0014  2 0381  1 0382  6 0383
 18  0 0014  2 0381
 19  0 0014  4 0508  2 0513  2 0521  2 0533
 20  0 0014  2 0503  6 0509  2 0531  2 0534  2 0536
 21  0 0014  6 0078
 22  0 0014  6 0079
 23  0 0014  4 0070  2 0090  2 0093
 24  0 0014  4 0071  2 0090
 25  0 0014  2 0068  4 0072  2 0087  2 0093
 26  0 0014  4 0089  3 0090  4 0095  2 0096  4 0120  1 0121  1 0122  3 0125
 27  0 0015  6 0075  6 0080
 28  0 0016  2 0095  4 0364
 29  0 0016  4 0094  2 0095  2 0097
 30  0 0017  2 0382  4 0395
 31  0 0018  2 0380  4 0392
 32  0 0019  2 0071  2 0072  2 0084  2 0096  6 0152  2 0153  2 0154
 33  0 0020  2 0076  2 0081  2 0578  0 0580
 34  0 0021  4 0392  4 0387  4 0393  4 0403  4 0411  4 0418  4 0443  4 0460  4 0468  4 0486  4 0494  2 0511  2 0548  2 0553  2 0560
     2 0561  2 0570  2 0573  2 0578
 35  0 0022  4 0394  4 0404  4 0412  4 0435  4 0437  4 0439  4 0449  2 0510  2 0554  2 0558  2 0562
 36  0 0023  4 0388  4 0398  4 0413  4 0421  4 0461  4 0467  2 0512  2 0571
 37  0 0024  4 0127  4 0128  6 0129
 38  0 0025  2 0123
 39  0 0026  2 0160  2 0166  2 0173  2 0177  2 0184  2 0258
 40  0 0026  0 0027  2 0028  2 0029  2 0030  2 0031
 41  0 0034  2 0123
 42  0 0034  0 0035  2 0037  2 0041
 43  0 0034  0 0035  2 0038  2 0041
 44  0 0036  4 0040  3 0041
 45  0 0036  4 0037  2 0039  2 0040
 46  0 0036  4 0038  2 0039  2 0040  1 0041
 47  0 0045  NOT REFERENCED
 48  0 0045  0 0046  2 0049  2 0050  2 0051
 49  0 0045  0 0048  3 0050
```

```
 50  0 0047  4 0050  4 0051  2 0052
 51  C 0048  4 0049  3 0050
 52  0 0054  2 0154
 53  C 0054  0 0056  2 0057
 54  C 0054  0 0056  3 0059
 55  C 0055  4 0057  2 0058  6 0059  2 0060
 56  0 0056  4 0058  3 0059
 57  0 0062  2 0210  2 0212  2 0233  2 0254  2 0319  2 0320  2 0368  2 0434  2 0440  2 0516  2 0523  2 0534
 58  0 0062  0 0064  2 0067
 59  0 0062  0 0065  2 0067  2 0070  2 0078  2 0079  2 0080
 60  0 0063  4 0067  6 0068  2 0069
 62  0 0083  2 0580
 63  0 0100  2 0547
 64  0 0101  6 0245  4 0262  6 0283  2 0331  4 0373  2 0433  2 0435
 65  0 0101  6 0247  4 0272  6 0290  2 0332  4 0373  2 0440
 66  0 0101  6 0266  1 0283  1 0290  2 0330  2 0338  4 0340  2 0344  4 0346  4 0373  2 0425  2 0426  4 0428  4 0429  2 0443
 67  0 0102  2 0154  4 0156  4 0267
 68  C 0102  2 0119  2 0120  2 0121  2 0122  2 0125  4 0148  4 0151  2 0156  2 0160  2 0179  2 0191  2 0193  2 0207  2 0217  2 0232
     2 0253  2 0265  2 0268  2 0276  2 0371
 69  C 0103  2 0246  2 0247
 70  0 0104  2 0516  2 0534
 71  0 0105  2 0233  2 0254  2 0368
 72  0 0106  2 0253  4 0268  4 0276
 73  C 0106  2 0393  2 0411  4 0511
 74  C 0106  2 0394  2 0412  4 0510
 75  0 0107  2 0419  4 0420  2 0441  4 0442  4 0478  2 0489  4 0490  2 0495  4 0496
 76  0 0107  4 0451  2 0463  4 0464  2 0469  4 0470  2 0478
 77  C 0107  4 0328  4 0357  2 0359  2 0360
 78  0 0107  4 0359  4 0360  4 0376  2 0417  2 0421  4 0422
 79  C 0107  4 0328  4 0352  2 0359
 80  C 0107  4 0329  4 0331  1 0354  3 0355  1 0357
 81  0 0107  4 0329  4 0332  1 0349  3 0350  1 0352
 82  C 0107  2 0120  2 0125  2 0191  2 0192  2 0206  2 0234  2 0285  2 0292  2 0300  1 0301  1 0302  4 0371
 83  C 0107  2 0248  4 0250  4 0262  2 0284  4 0288  2 0432
 84  0 0107  4 0248  2 0288
 85  C 0107  2 0249  4 0250  4 0272  2 0291  4 0295  2 0440
 86  C 0107  4 0249  2 0295
 87  0 0107  1 0246  1 0247  1 0266  6 0299  2 0300  4 0304  1 0312  1 0322  1 0323  1 0330  1 0338  1 0340  1 0344  1 0346  4 0374
     4 0377  1 0425  1 0426  1 0428  1 0429  1 0433  1 0434  1 0435  1 0440  1 0443
 88  0 0107  3 0191  6 0192  1 0193  6 0194  4 0206  1 0207  1 0209  1 0216  1 0217  4 0234  1 0235  1 0237  4 0285  1 0286  4 0292
     1 0293
 89  C 0107  4 0453  2 0457  1 0458  1 0459  1 0461  6 0462  1 0463  1 0464  1 0468  1 0469  1 0470  6 0471  4 0479  2 0483  1 0484
     1 0485  1 0487  6 0488  1 0489  1 0490  1 0494  1 0495  1 0496  6 0497
 90  C 0107  4 0419  4 0441  4 0463  4 0469  4 0489  4 0495  2 0507  2 0509  2 0514  2 0516  2 0518  2 0522  2 0523  2 0525  2 0530
     2 0531  4 0539
 91  0 0107  4 0378  6 0399  1 0400  1 0402  1 0403  4 0408
 92  0 0108  4 0331  2 0357  2 0485  2 0487  2 0489  2 0490  2 0495  2 0496
 93  C 0108  4 0332  2 0352  2 0459  2 0461  2 0463  2 0464  2 0469  2 0470
 94  0 0108  4 0262  4 0286  6 0301  2 0331  2 0434
 95  C 0108  4 0272  4 0293  6 0302  2 0332
 96  0 0109  4 0477  2 0482  4 0502
 97  0 0109  4 0372  2 0416  4 0450  4 0503
 98  0 0109  4 0452  2 0456  4 0476
 99  C 0109  4 0162  4 0166  4 0167  4 0172  4 0177  4 0178  2 0257  4 0282
100  C 0109  2 0337  4 0339  4 0345  4 0427
101  C 0109  4 0401  2 0409  4 0410
```

```
102  0 0110  2 0363  4 0365
103  0 0111  4 0140  2 0403
104  0 0112  2 0139  4 0139  2 0400  6 0402
105  0 0113  4 0327  4 0331  2 0354  6 0355  4 0356  2 0483  2 0484  2 0494
106  0 0113  4 0327  4 0332  2 0349  6 0350  4 0351  2 0457  2 0458  2 0468
107  0 0114  2 0148
108  2 0115  0 0353
109  0 0116  2 0277
110  0 0117  4 0122  2 0123  4 0125  2 0127  2 0128
111  0 0118  4 0123  2 0124  1 0127  1 0128  1 0129
112  2 0126  0 0129
113  0 0134  2 0514  2 0517  2 0518  2 0524  2 0525  2 0531  2 0533  2 0535  2 0536
114  0 0134  0 0135  2 0139
115  0 0134  0 0136  2 0140
116  0 0135  4 0137  1 0138  1 0139  1 0140
117  0 0145  2 0231  2 0252  2 0366
118  0 0146  2 0147
119  0 0152  2 0233  2 0253  2 0367
120  0 0158  0 0159  2 0166  2 0173  2 0177  2 0180  2 0184
121  0 0165  2 0170
122  0 0190  2 0208
123  0 0193  2 0198
124  0 0202  2 0230
125  0 0202  0 0204  2 0209
126  0 0202  0 0204  2 0216
127  0 0202  0 0205  2 0209  4 0217
128  0 0202  0 0203  4 0216
129  2 0208  2 0213  0 0219
131  0 0221  2 0262  2 0272
132  0 0221  0 0223  2 0226  2 0230  2 0236
133  0 0221  0 0224  4 0230  2 0235
134  0 0221  0 0225  4 0230  4 0235  6 0237
135  0 0221  0 0223  4 0228  2 0230  2 0237  6 0242
136  0 0222  4 0226  4 0227  2 0232  2 0233
140  0 0245  2 0303  2 0375
141  2 0251  0 0257  2 0263  2 0273
142  0 0252  2 0255  2 0269  2 0278
145  2 0257  0 0281
148  0 0298  2 0333  2 0444
150  0 0307  2 0331  2 0332
151  0 0307  0 0308  2 0318
152  0 0307  0 0310  2 0323
153  0 0307  0 0309  2 0312  2 0322
154  0 0307  0 0310  6 0313  4 0323
155  0 0307  0 0309  2 0313  6 0322
156  0 0307  0 0308  2 0311  1 0313  6 0317  2 0318  1 0323
157  2 0311  2 0314  0 0317
159  0 0326  2 0448
160  0 0330  2 0334
161  0 0330  2 0342  2 0347
164  0 0366  2 0369
166  2 0386  2 0398  4 0389  2 0413  4 0512
167  0 0399  2 0540
168  0 0416  2 0506
169  2 0423  2 0445  2 0473  2 0499  0 0507
170  2 0430  0 0447
```


Reproduced from best available copy.

```
171  0 0515  2 0527
172  0 0522  2 0530
173  0 0544  NOT REFERENCED
174  0 0546  2 0549
```

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

```
LOC  TEXT

  1 REPLACE
  2 0
  3 1
  4 INTEGER
  5 IF 0 =
  6 IF 1 =
  7 WRITE(6)
  8 WHILE 1 = 1
```

```
STACKING DECISIONS    = 13662
CALLS TO SCAN         = 3936
CALLS TO IDENTIFY     = 989
NUMBER OF REDUCTIONS = 9727
MAX STACK SIZE        = 25
MAX IND. STACK SIZE   = 32
END IND. STACK SIZE   = 17
MAX EXT_ARRAY INDEX   = 2
XREF LIST ENTRIES     = 870
STATEMENT COUNT       = 593
MAX OUTER_LIST INDEX = 0
MAX NESTING DEPTH     = 3
FREE STRING AREA      = 55355
```

```
786 CARDS WERE PROCESSED.
NO ERRORS WERE DETECTED DURING PHASE 1 .
```

```
TOTAL ELAPSED TIME IN COMPILER    0:0:20.52.
ELAPSED SET UP TIME               0:0:0.04.
ACTUAL ELAPSED COMPILING TIME     0:0:19.13.
ELAPSED CLEAN-UP TIME AT END      0:0:1.35.
PROCESSING RATE: 2465 CARDS PER MINUTE.
```

```
BEGIN TEST OF HAL IN HAL
   1 C|    2  5  6                                                    | 1
   2 M| TPARM:PROGRAM;                                                | 2

**BLANKS=1
***TOKEN=TPARM
***TOKEN=:
***TOKEN=PROGRAM
***TOKEN=;

***OVER***
***MAIN*** TPARM:PROGRAM;

   3 E|            *                                                  | 3
   4 M| DECLARE M1     ,                                              | 4
   5 S|            3,3                                                | 5

**BLANKS=65
***TOKEN=DECLARE
**BLANKS=1
***TOKEN=M1,MARKER=*
***TOKEN=$
***TOKEN=(
***TOKEN=3
***TOKEN=,
***TOKEN=3
***TOKEN=)
***TOKEN=,
   6 E|  *                                                            | 6
   7 M|  M2    ;                                                      | 7
   8 S|    8,8                                                        | 8

**BLANKS=66
***TOKEN=M2,MARKER=*
***TOKEN=$
***TOKEN=(
***TOKEN=8
***TOKEN=,
***TOKEN=8
***TOKEN=)
***TOKEN=;

***OVER***          *         *
***MAIN*** DECLARE M1$(3,3), M2$(8,8);

   9 M| DECLARE I1 ARRAY(5) INTEGER INITIAL(3,2,5,4,1);               | 9

**BLANKS=72
***TOKEN=DECLARE
**BLANKS=1
***TOKEN=I1
**BLANKS=1
***TOKEN=ARRAY
***TOKEN=(
***TOKEN=5
***TOKEN=)
```

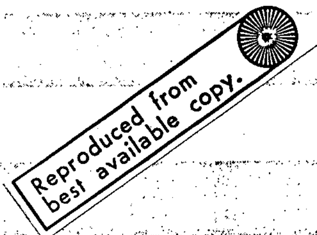
```
**BLANKS=1
***TOKEN=INTEGER
**BLANKS=1
***TOKEN=INITIAL
***TOKEN=(
***TOKEN=3
***TOKEN=,
***TOKEN=2
***TOKEN=,
***TOKEN=5
***TOKEN=,
***TOKEN=4
***TOKEN=,
***TOKEN=1
***TOKEN=)
***TOKEN=;

***OVER***
***MAIN*** DECLARE I1 ARRAY(5) INTEGER INITIAL(3,2,5,4,1);


  10 M| DECLARE ARRAY(5) SCALAR, S1 , S2;                           | 10

**BLANKS=32
***TOKEN=DECLARE
**BLANKS=1
***TOKEN=ARRAY
***TOKEN=(
***TOKEN=5
***TOKEN=)
**BLANKS=1
***TOKEN=SCALAR
***TOKEN=,
**BLANKS=1
***TOKEN=S1
**BLANKS=1
***TOKEN=,
**BLANKS=1
***TOKEN=S2
***TOKEN=;

***OVER***
***MAIN*** DECLARE ARRAY(5) SCALAR, S1 , S2;
```

3"

2"

1"

```
FUNCTION DECLARATIONS
 11 H| FUNCTION DECLARATIONS                                              | 11
 12 E|                    *                                               | 12
 13 M|  PRO1: FUNCTION   (P     ) MATRIX (*,*);                           | 13
 14 S|                    *,*                                             | 14

**BLANKS=47
***TOKEN=PRO1
***TOKEN=:
**BLANKS=1
***TOKEN=FUNCTION
**BLANKS=2
***TOKEN=(
***TOKEN=P,MARKER=*
***TOKEN=$
***TOKEN=(
***TOKEN=*
***TOKEN=,
***TOKEN=*
***TOKEN=)
***TOKEN=)
**BLANKS=1
***TOKEN=MATRIX
**BLANKS=1
***TOKEN=(
***TOKEN=*
***TOKEN=,
***TOKEN=*
***TOKEN=)
***TOKEN=;

***OVER***              *
***MAIN*** PRO1: FUNCTION (P$(*,*)) MATRIX (*,*);

  15 E|          *-1                                                      | 15
  16 M|     RETURN P   ;                                                  | 16

**BLANKS=45
***TOKEN=RETURN
**BLANKS=1
***TOKEN=P,MARKER=*
***TOKEN=*
***TOKEN=*
***TOKEN=(
***TOKEN=-
***TOKEN=1
***TOKEN=)
***TOKEN=;

***OVER***        *
***MAIN*** RETURN P**(-1);

  17 M|    CLOSE PRO1;                                                    | 17

**BLANKS=67
***TOKEN=CLOSE
```



```
**BLANKS=1
***TOKEN=PRO1
***TOKEN=;

***OVER***
***MAIN*** CLOSE PRO1;

  18 M|  PRO: FUNCTION (B) MATRIX (*,*);                          | 18

**BLANKS=67
***TOKEN=PRO
***TOKEN=:
**BLANKS=1
***TOKEN=FUNCTION
**BLANKS=1
***TOKEN=(
***TOKEN=B
***TOKEN=)
**BLANKS=1
***TOKEN=MATRIX
**BLANKS=1
***TOKEN=(
***TOKEN=*
***TOKEN=,
***TOKEN=*
***TOKEN=)
***TOKEN=;

***OVER***
***MAIN*** PRO: FUNCTION (B) MATRIX (*,*);

  19 M|   DECLARE MATRIX (*,*), B;                               | 19

**BLANKS=49
***TOKEN=DECLARE
**BLANKS=1
***TOKEN=MATRIX
**BLANKS=1
***TOKEN=(
***TOKEN=*
***TOKEN=,
***TOKEN=*
***TOKEN=)
***TOKEN=,
**BLANKS=1
***TOKEN=B
***TOKEN=;

***OVER***
***MAIN*** DECLARE MATRIX (*,*), B;

  20 M|     RETURN PRO1(B            );                          | 20
  21 S|                 2 TO 4,1 TO 3                            | 21

**BLANKS=56
```

```
***TOKEN=RETURN
**BLANKS=1
***TOKEN=PROJ
***TOKEN=(
***TOKEN=B
***TOKEN=$
***TOKEN=(
***TOKEN=2
**BLANKS=1
***TOKEN=TO
**BLANKS=1
***TOKEN=4
***TOKEN=,
***TOKEN=1
**BLANKS=1
***TOKEN=TO
**BLANKS=1
***TOKEN=3
***TOKEN=)
***TOKEN=)
***TOKEN=;

***OVER***
***MAIN*** RETURN PROJ(B$(2 TO 4,1 TO 3));

  22 M|   CLOSE PRO;                                   | 22

**BLANKS=50
***TOKEN=CLOSE
**BLANKS=1
***TOKEN=PRO
***TOKEN=;

***OVER***
***MAIN*** CLOSE PRO;
```

```
MAIN PROGRAM
 23 H| MAIN PROGRAM                                                    | 23
 24 M| DO FOR I=1 TO 5;                                                | 24

**BLANKS=67
***TOKEN=DO
**BLANKS=1
***TOKEN=FOR
**BLANKS=1
***TOKEN=I
***TOKEN==
***TOKEN=1
**BLANKS=1
***TOKEN=TO
**BLANKS=1
***TOKEN=5
***TOKEN=;

***OVER***
***MAIN*** DO FOR I=1 TO 5;

 25 M| DO FOR J=1 TO 5;                                                | 25

**BLANKS=63
***TOKEN=DO
**BLANKS=1
***TOKEN=FOR
**BLANKS=1
***TOKEN=J
***TOKEN==
***TOKEN=1
**BLANKS=1
***TOKEN=TO
**BLANKS=1
***TOKEN=5
***TOKEN=;

***OVER***
***MAIN*** DO FOR J=1 TO 5;

 26 E|                  J                                              | 26
 27 E|               I1$I                                              | 27
 28 M|    M2   = S1       ;                                            | 28
 29 S|      I,J     I1                                                 | 29
 30 S|               I                                                 | 30

**BLANKS=65
***TOKEN=M2
***TOKEN=$
***TOKEN=(
***TOKEN=I
***TOKEN=,
***TOKEN=J
***TOKEN=)
***TOKEN==
**BLANKS=1
```

```
***TOKEN=S1
***TOKEN=$
***TOKEN=(
***TOKEN=I1
***TOKEN=$
***TOKEN=(
***TOKEN=I
***TOKEN=)
***TOKEN=)
***TOKEN=*
***TOKEN=*
***TOKEN=(
***TOKEN=I1
***TOKEN=$
***TOKEN=I
***TOKEN=*
***TOKEN=*
***TOKEN=(
***TOKEN=J
***TOKEN=)
***TOKEN=)
***TOKEN=;

***OVER***
***MAIN*** M2$(I,J)= S1$(I1$(I))**(I1$I**(J));


  31 M| END;                                                          | 31

**BLANKS=62
***TOKEN=END
***TOKEN=;

***OVER***
***MAIN*** END;


  32 C|  THE NEXT GROUP HAS AN OVERLAPPING E LINE                     | 32
  33 E|        I                                                      | 33
***** ERROR # 1 OF SEVERITY 1: OVERLAPPING E-LINE CHARACTERS. *****
  34 E|        I1$I                                                   | 34
  35 M| S2 =S1     ;   /* COMMENTS ARE BLANKS */                      | 35
  36 S|    I    I                                                     | 36

**BLANKS=75
***TOKEN=S2
***TOKEN=$
***TOKEN=(
***TOKEN=I
***TOKEN=)
***TOKEN==
***TOKEN=S1
***TOKEN=$
***TOKEN=(
***TOKEN=I
***TOKEN=)
***TOKEN=*
***TOKEN=*
```

```
***TCKEN=(
***TCKEN=I1
***TCKEN=$
***TCKEN=I
***TCKEN=)
***TCKEN=;

***OVER***
***MAIN*** S2$(I)=S1$(I)**(I1$I);


  37 E|        2                                                              | 37
***** ERROR # 2 OF SEVERITY 1: SOURCE PROGRAM OUT OF ORDER. LAST ERROR ON LINE 33. *****
  38 C|    THIS SHOULD DEMONSTRATE CARDS OUT OF ORDER                         | 38
  39 M|S2 = S2 ;                                                              | 39
  40 S| I      I                                                              | 40

**BLANKS=67
***** ERROR # 3 OF SEVERITY 1: S-LINE OVERLAPS M-LINE. LAST ERROR ON LINE 37. *****
***TCKEN=S2
**BLANKS=1
***TCKEN==
**BLANKS=1
***TCKEN=S2
***TCKEN=$
***TCKEN=(
***TCKEN=I
***TCKEN=)
***TOKEN=*
***TCKEN=*
***TCKEN=(
***TCKEN=2
***TCKEN=)
***TCKEN=;

***OVER***
***MAIN*** S2 = S2$(I)**(2);


  41 C| AN S LINE OVERLAP ABOVE                                               | 41
  42 M| END;                                                                  | 42

**BLANKS=71
***TCKEN=END
***TOKEN=;

***OVER***
***MAIN*** END;


  43 E| *       *                                                             | 43
  44 M| M1=PRC(M2 /* IGNORED */ );                                            | 44
  45 S|             4 AT 3,4 AT 5                                             | 45

**BLANKS=75
***TCKEN=M1,MARKER=*
***TCKEN==
***TCKEN=PRC
```

```
***TOKEN=(
***TOKEN=M2.MARKER=*
***TOKEN=$
***TOKEN=(
***TOKEN=4
**BLANKS=1
***TOKEN=AT
**BLANKS=1
***TOKEN=3
***TOKEN=,
***TOKEN=4
**BLANKS=1
***TOKEN=AT
**BLANKS=1
***TOKEN=5
***TOKEN=)
**BLANKS=2
***TOKEN=)
***TOKEN=;

***CVER*** *      *
***MAIN*** M1=PRO(M2$(4 AT 3,4 AT 5) );

   46 M| CLOSE TPARM;                                                 | 46

**BLANKS=53
***TOKEN=CLOSE
**BLANKS=1
***TOKEN=TPARM
***TOKEN=;

***CVER***
***MAIN*** CLOSE TPARM;

   47 M| ?                                                            | 47

**BLANKS=67
***TOKEN=?
47 CARDS WERE PROCESSED
3 ERRORS WERE DETECTED, THE LAST ERROR WAS ON LINE 40
*****SUMMARY OF DETECTED ERRORS.*****
ERROR # 1 ON LINE 33 OF SEVERITY 1.
ERROR # 2 ON LINE 37 OF SEVERITY 1.
ERROR # 3 ON LINE 40 OF SEVERITY 1.
CARD-PROCESSING RATE:  4.4269995E+02 CARDS PER MINUTE.
 CLOCK TIME IS 4361443
 THIS TEST IS NOW COMPLETE.
```

3"

2"

1"

```
C  ¢2 ¢5 ¢6
M TPARM:PROGRAM;
E         *
M DECLARE M1    ,
S           3,3
E  *
M  M2    ;
S    8,8
M DECLARE I1 ARRAY(5) INTEGER INITIAL(3,2,5,4,1);
M DECLARE ARRAY(5) SCALAR, S1 , S2;
H FUNCTION DECLARATIONS
E                    *
M  PRO1: FUNCTION  (P    ) MATRIX (*,*);
S                    *,*
E           *-1
M    RETURN P  ;
M   CLOSE PRO1;
M  PRO: FUNCTION (B) MATRIX (*,*);
M   DECLARE MATRIX (*,*), B;
M    RETURN PRO1(B            );
S                 2 TO 4,1 TO 3
M   CLOSE PRO;
H MAIN PROGRAM
M DO FOR I=1 TO 5;
M DO FOR J=1 TO 5;
E                 J
E             I1$I
M   M2    = S1      ;
S     I,J     I1
S               I
M END;
C  THE NEXT GROUP HAS AN OVERLAPPING E LINE
E         I
E        I1$I
M S2 =S1     ;  /* COMMENTS ARE BLANKS */
S    I    I
E        2
C    THIS SHOULD DEMONSTRATE CARDS OUT OF ORDER
MS2 = S2 ;
S I      I
C AN S LINE OVERLAP ABOVE
M END;
E *      *
M M1=PRO(M2 /* IGNORED */ );
S          4 AT 3,4 AT 5
M CLOSE TPARM;
M ?
```