Final Report

CONTINUED ADVANCEMENT OF THE
PROGRAMMING LANGUAGE HAL TO AN
OPERATIONAL STATUS

NAS 9-11944

December 30, 1971
<table>
<thead>
<tr>
<th>TABLE OF CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page</td>
</tr>
<tr>
<td>FOREWORD</td>
</tr>
<tr>
<td>1.0 TASK SUMMARY</td>
</tr>
<tr>
<td>1.1 Task I: Maintenance and Training</td>
</tr>
<tr>
<td>1.2 Task II: Advanced HAL Development</td>
</tr>
<tr>
<td>2.0 HAL INSTALLATION AND ON-SITE SUPPORT</td>
</tr>
<tr>
<td>2.1 Installation</td>
</tr>
<tr>
<td>2.2 Program Changes and Maintenance Procedures</td>
</tr>
<tr>
<td>3.0 HAL COURSES</td>
</tr>
<tr>
<td>3.1 General Description</td>
</tr>
<tr>
<td>3.2 Course Preparation</td>
</tr>
<tr>
<td>3.3 Course Outline</td>
</tr>
<tr>
<td>4.0 NECESSARY MODIFICATIONS AND ADDITIONS</td>
</tr>
<tr>
<td>4.1 Storage Allocation Problem</td>
</tr>
<tr>
<td>4.2 Miscellaneous Improvements</td>
</tr>
<tr>
<td>5.0 HAL TRANSFERABILITY</td>
</tr>
<tr>
<td>5.1 Technical Approach</td>
</tr>
<tr>
<td>5.2 Translation of XPL Programs Into HAL</td>
</tr>
<tr>
<td>5.3 Feasibility Demonstration (&quot;HAL-in-HAL&quot;)</td>
</tr>
<tr>
<td>APPENDIX A. HAL Course Material</td>
</tr>
<tr>
<td>A.1 Overview</td>
</tr>
<tr>
<td>A.2 Longer HAL Course</td>
</tr>
<tr>
<td>APPENDIX B. HAL-in-HAL Detailed Description and Listing</td>
</tr>
</tbody>
</table>
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.
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 only 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 OS/360 Core Allocation. 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 RTOS Core Allocation. 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 existence 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 "background" utilities running. Although they are termed "background", 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.
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 groundwork 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 printed source code changed to increase readability.
3.0 HAL COURSES

3.1 General Description

Intermetrics personnel prepared and conducted three HAL language 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 homework 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. $\text{HAL}_M$ (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
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
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 allocating 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
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.
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 compatibility many modern computers have included hardware or microprogram 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 adhered 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 mid-level 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 high level 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:
Fig. 5.1 Construction of the HAL Compiler System
Fig. 5.2 Proposed Construction of HAL Compiler for Flight Computers
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
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 provides 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.
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:

\begin{verbatim}
DECLARE ALPHA CHARACTER;  \hfill (XPL)
\end{verbatim}


to

\begin{verbatim}
DECLARE ALPHA CHARACTER (255) VARYING; \hfill (HAL)
\end{verbatim}

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:

\begin{verbatim}
DECLARE (I,J,K) FIXED, L BIT(8); \hfill (XPL)
\end{verbatim}

becomes:

\begin{verbatim}
DECLARE INTEGER, I, J, K; \hfill (HAL)
DECLARE L BIT(8);
\end{verbatim}

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 expressions 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:

\[
\text{IX} = \text{IY} + \text{IZ};
\]
\[
\text{VALUE} = \text{ABLE (IX)};
\]

should become in HAL:

\[
\text{IX} = \text{IY} + \text{IZ};
\]
\[
\text{VALUE} = \text{ABLE}_{\text{IX}} + 1;
\]

as opposed to the HAL sequence:

\[
\text{IX} = \text{IY} + \text{IZ} + 1;
\]
\[
\text{VALUE} = \text{ABLE}_{\text{IX}};
\]

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:

\[
\text{DECLARE INDEX FIXED, INDEXTAB (199) BIT (8)};
\]
\[
\text{DO I = 1 TO 50;}
\]
\[
\text{INDEX (I) = 0;}
\]
\[
\text{END;}
\]

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:

\[
\text{INDEXTAB = 0;}
\]

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

    B1 TO 5 = A;

    B1 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_STRING1 (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 (HAL)
   N AT START+1

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:
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:

```hal
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 (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. Uninitialized 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.
APPENDIX A.

HAL Course Material
A.1 OVERVIEW
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
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
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.
### SALIENT FEATURES OF HAL

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

*INTERMETRICS*
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
- Arithmetic
  - Scalar
  - Integer
  - Vector
  - Matrix

String
- Bit
- Character

Organizations
- Array
  - Individual Data-Type
  - Structure
    - Array

Combinations of Data-Types

Unique notation: VECTOR:NAME, BIT STRING:NAME, CHARACTER STRING:NAME, MATRIX: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: 10110101

CHARACTER STRING: VOLTAGE ON BATTERY B 2 VOLTS BELOW SPEC
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

Data Compool

Symbolic Library

Program #1

Program #2

Program #N
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.,

  \[ \text{Compool} \rightarrow \text{Program} \begin{cases} \rightarrow \text{Task} \\ \rightarrow \text{Procedure} \\ \rightarrow \text{Function} \end{cases} \begin{cases} \text{Procedure} \\ \text{Function} \end{cases} \rightarrow \text{etc.} \]

- Names defined in an inner block are never recognized in an outer block. Inner blocks effectively isolate locally defined variables.
ABLE:

PROGRAM;
DECLARE VECTOR (5) A, B, C;
\[ \bar{A} = \bar{B} + \bar{C}; \]

BAKER:

TASK;
DECLARE A INTEGER;

CHARLIE:

PROCEDURE;
DECLARE X;
DECLARE A BIT (10);

END CHARLIE;

END BAKER;

GRAB:

PROCEDURE;
DECLARE X VECTOR (4);

END GRAB;

END ABLE;
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

\[ \bar{Z} = \bar{w}^T \bar{b}; \]

\[ \Omega = \bar{Z} \bar{w}^{*T}/(ZMAG^2 + \alpha^2); \]

\[ \Delta X = \Omega \Delta Q; \]

\[ X = X + \Delta X; \]

\[ F = 1 + (\alpha^2/(ZMAG^2 + \alpha^2))^{1/2}; \]

\[ \bar{w} = \bar{w} - \Omega \bar{Z}/F; \]

GSOP Specification

\[ \bar{Z} = \bar{w}^T \bar{b}; \]

\[ \omega^T = \frac{1}{Z^2 + \alpha^2} \bar{Z}^T \bar{w}^T; \]

\[ \delta X = \omega \delta Q; \]

\[ X = X' + \delta X; \]

\[ \omega^T = \frac{\omega \bar{Z}^T}{1 + \sqrt{\frac{\alpha^2}{Z^2 + \alpha^2}}}; \]

\[ W = W' - \frac{\omega \bar{Z}^T}{1 + \sqrt{\frac{\alpha^2}{Z^2 + \alpha^2}}}; \]

where \( \bar{b} \) \equiv geometry vector
\( W \) \equiv square root of covariance
\( \bar{\alpha}^2 \) \equiv measurement variance
\( X \) \equiv state vector
CONTROL, LOGIC AND COMPUTATION
(Cross product steering of Apollo vehicle)

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

XSTEER: IF TGO < 4 THEN DO;

\[ \Omega_{\text{CNB}} = 0; \]
\[ SW = \text{OFF}; \]

SCHEDULE ENGINE_OFF AT (TIME+TGO)

PRIORITY (20) E_OFF_ID;

GO TO START;

END;

\[ \Delta L = C \Delta L - \Delta V; \]
\[ \Omega_{\text{C}} = K(\bar{V}_G \Delta L)/(AVLAV (\bar{V}_G) AVLAV (\Delta L)); \]
\[ \Omega_{\text{CNB}} = \hat{\Sigma}_{\text{MN}} \hat{\text{REFSMAT}} \Omega_{\text{C}}; \]

GO TO START;

where TGO = "time-to-go"
\[ \bar{V}_G = "velocity-to-be-gained" \]
\[ \Omega_{\text{C}} = \text{rate command} \]
EXAMPLES OF MATRIX PARTITIONING

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

$$E = \begin{bmatrix}
  E_{p-p} & E_{p-v} & E_{p-l} \\
  E_{v-p} & E_{v-v} & E_{v-l} \\
  E_{l-p} & E_{l-v} & E_{l-l}
\end{bmatrix}$$

1. RMS Errors

$$\text{RMS\_POS} = \sqrt{\text{trace}(E_{1\text{ TO }3, 1\text{ TO }3})}$$

$$\text{RMS\_VEL} = \sqrt{\text{trace}(E_{4\text{ TO }6, 4\text{ TO }6})}$$

2. Initialize $E$ for new landmark

$$E_{1\text{ TO }6, 7\text{ TO }9} = 0;$$

$$E_{7\text{ TO }9, 1\text{ TO }6} = 0;$$

$$E_{7\text{ TO }9, 7\text{ TO }9} = \text{MATRIX}_{3,3} (A^2, 0, 0$$

$$0, B^2, 0$$

$$0, 0, C^2);$$
BIT AND CHARACTER MANIPULATIONS

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

```
SYSTEM_STATUS 1 0 1 1 1 0
```

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;

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;
         END;
      ELSE IF TRACKFLAG THEN 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 (R2, V2, (ABVAL (R2) - 30480), MU_EARTH,
   T_R_FLAG)) ASSIGN (TIME_32, R3, V3);

SUBROUTINE:

TIME_RADIUS: PROCEDURE (A,B,C,D,E) ASSIGN (F,G,H);

[Statements]

RETURN;

END TIME_RADIUS;

NOTE: "Call-By-Name", "Call-By-Value"
EXAMPLE OF A FUNCTION

ABLE: \[ N = \text{TRACER}(A+B); \]

GO TO BAKER;

TRACER: FUNCTION(Q);

DECLARE Q MATRIX(A,*);

IF trace(Q) > 100 THEN

RETURN \( Q^{-1} Q + Q^2 + Q^3 \)

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.
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

A: TASK; UPDATE;
M = N + P;
CLOSE A;

B: TASK; UPDATE;
N = X Y;
CLOSE B;

EXAMPLE 2: UPDATE CONFLICTS

A: TASK; UPDATE;
Y = Y - X;
CLOSE A;

B: TASK; UPDATE;
Y = Y - Z;
CLOSE B;

NOTES:
1. B "INTERRUPTS" A IN BOTH CASES
2. #1 TASK A RESUMES USING OLD AND NEW VALUES FOR N
3. #2 TASK A RESUMES "CLOBBERING" THE VALUE FOR Y SET BY TASK B
ERROR CONDITION STATEMENTS EXAMPLES:

ON ERROR

12  GO TO ABLE;

ON ERROR

1 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
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 - 11

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

=  NOT=

integer

<=  <=

bit

>=  >=

character

NOT <

NOT >

>  <
EXAMPLES OF DATA OPERATIONS - I

Arithmetic Operations

\[ I = (-J)^3 + K(3+J); \quad \text{all integers} \]

\[ A = P/R + R^P - \bar{V} \cdot \bar{M} \bar{V}; \quad \text{scalars, vectors, matrices} \]

\[ B = \bar{V} \cdot \bar{M} + 5 \bar{V} \bar{V} + A(\bar{V} \cdot \bar{F})^2 \bar{F} + \bar{F}/(\bar{V} \cdot \bar{V} \bar{M}); \quad \text{scalars, vectors, matrices} \]

\[ \bar{C} = -\bar{M} \bar{N} + \bar{M}^{-1} \bar{V} \bar{V}/A - (\bar{M} + \bar{N})^T; \quad \text{scalars, vectors, matrices} \]

\[ A = I/J + B J + J^{-6.4}; \quad \text{integers, scalars} \]
EXAMPLES OF STRING OPERATIONS

Bit String

\[ D = B \text{ AND } C; \]
\[ D = A \text{ OR } (B \text{ AND } C); \]
\[ A = D \text{ OR NOT } B \text{ OR } (B \text{ AND } C); \]

Character String

\[ C = '\text{PLEASE}' || '\text{HELP}'; \]
\[ D = '\text{THE ANS. IS}' || 'X' || '\text{N.M.}'; \]
EXAMPLES OF ARRAY OPERATIONS

"Two-array" operations:

- \([P]/[A]\), \([\bar{F}] * [\bar{V}]\), \([R]^{[P]}\), \([A]\) OR \([B]\)

One-array operations:

- \(-P/[A]\), \(\bar{F} * [\bar{V}]\), \(R^{[P]}\), \([A]\) OR \([B]\)
EXAMPLES OF COMPARISON OPERATIONS

IF \( I > J \) THEN ------

IF A NOT < \((X^2 - 5 \overline{V} \cdot \overline{V})\) THEN ------

DO WHILE S = (A AND B) ------

IF \( ^* M = ^* N \) 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:

<table>
<thead>
<tr>
<th>Organization</th>
<th>Type</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARRAY</td>
<td>INTEGER</td>
<td>PRECISION</td>
</tr>
<tr>
<td></td>
<td>SCALAR (optional)</td>
<td>INITIAL</td>
</tr>
<tr>
<td></td>
<td>VECTOR</td>
<td>CONSTANT</td>
</tr>
<tr>
<td></td>
<td>MATRIX</td>
<td>STATIC</td>
</tr>
<tr>
<td></td>
<td>BIT</td>
<td>AUTOMATIC</td>
</tr>
<tr>
<td></td>
<td>CHARACTER</td>
<td>LOCKTYPE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DENSE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ALIGNED</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VARYING</td>
</tr>
</tbody>
</table>
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 $\mathbf{V}$, $\mathbf{M}$)

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

$\mathbf{V}_1$ TO 5 ------- sub-vector,

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

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

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

$S_3$ -------- single bit,

$S_2$ TO 10 ------- sub-string, $S_6$ AT P ------- sub-string

$C_3$ TO # ------- sub-string,
EXAMPLES OF INDEXING - II

3. Arrays (given $\mathbf{A}$, a two-dimensional array of matrices)

$$\mathbf{A}^*_{1 \text{ TO } 4, \ 3 \text{ TO } 6} : \text{sub-array}$$

$$\mathbf{A}^p, q : \text{an array of scalar elements}$$

$$\mathbf{A}^*, *, *, 2 \equiv \mathbf{A}^*, 2 : \text{sub-array of vector elements}$$

4. Array of Bit Strings

$$\mathbf{A}^3 \text{ TO } 5, *: 1 \text{ TO } 6 : \text{sub-array of sub-strings}$$

$$\mathbf{A}^*_6, 4.3 : \text{one particular bit}$$
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;
    B = A = B;
    C = D;
    GO TO ABLE:
END BAKER;

ELSE CHARLIE: DO;
    \bar{Z} = \bar{M} \bar{V};
    \bar{F} = \bar{V} * \bar{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;  

\[ X = Y^2 + A_i; \]

\[ N = N - 0.006 X; \]

\[ P = 1; \quad S = 2; \quad L = 3; \]

END ABLE;

2. **Iterative DO-WHILE**

ABLE: DO WHILE (X > Y AND GO_FLAG = ON);  

\[ X = Y^2 + P \cdot \text{LOG}(Z); \]

\[ \text{GO_FLAG} = \text{TRAKFLAG OR NAV_FLAG}; \]

\[
\begin{bmatrix}
\text{Statements} \\
\vdots \\
\vdots \\
\vdots \\
\end{bmatrix}
\]

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;

\[ X = y^2; \]  \hspace{1cm} \text{CASE 1}

BAKER:  DO CASE P;

\[ F = A + B; \]  \hspace{1cm} \text{CASE 1}
\[ \overline{G} = \hat{M} \overline{V}; \]  \hspace{1cm} \text{CASE 2}

END;

GO TO CHARLIE;  \hspace{1cm} \text{CASE 3}

\[ \overline{Z} = \overline{W} + \overline{E}; \]  \hspace{1cm} \text{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 B OR C THEN Z = * M V;
   ELSE CHOICE: Z = *⁻¹ V;

3. More Complex:

   IF S = (A OR B) THEN IF X > 5 AND Y > 6 THEN GO TO OUT;
   ELSE IF [A] NOT= [B] THEN [A] = [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:

PROGRAM;
DECLARE A,B,C,D,E,F;
...;
...;

BAKER: PROCEDURE;
OUTER B,D,F;
DECLARE A,E;
...;
...

- 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.  
\[ M S^I = S^I_X^I; \]  
\[ S^I = S^I_X^I \]  
becomes  
\[ S^I$(I) = S^I$(IX$(I)); \]

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

Ex.  
\[ E^2 E^2 I \]  
\[ M S^2 = X \]  
becomes  
\[ S^2**(2) = X**(I**(2)); \]

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 PROCESSCOMMENT (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.
STATE

1 MI  HALINHAL: PROGRAM;
2 CI  B P C D E F G
3 CI  THE FOLLOWING GROUP OF DECLARES ARE ONLY TEMPORARY
4 MI  REPLACE B BY 'REPLACE';
5 MI  R FALSE BY 'O'; R TRUE BY '1'; R BIT_1 BY 'INTEGER';
6 MI  R IF_NOT BY 'IF 0 ='; R IFF BY 'IF 1 =';
7 MI  R CUT BY 'WRITE(5)';
8 MI  R FOREVER BY 'WHILE 1 = 1';
9 MI  DEFAULT INTEGER;
10 MI  DECLARE XT0 CHARACTER(70) INITIAL [' '];
11 MI  DECLARE CHARACTER(255) VARYING, BUILT, BUILT_UP, BUILT_TOKEN, BUILT_O_P;
12 MI  DECLARE BLANK_FLAG BIT_1 INITIAL(FALSE);
13 MI  DECLARE INTEGER INITIAL(0),
14 MI  ERROR_COUNT, MACRO_POINT, MACRO_LIMIT,
15 MI  OLD_LEVEL, NEW_LEVEL,
16 MI  MAX_SEVERITY, STATEMENT_SEVERITY,
17 MI  SAVE_SEVERITY ARRAY(100),
18 MI  SAVE_LINE ARRAY(100), PREVIOUS_ERROR, I,
19 MI  DECLARE CCFILEING BIT_1 INITIAL(TPUF);
20 MI  DECLARE INTEGER, TIME1, TIME2;
21 MI  DECLARE MACRO_STREAM CHARACTER(255) VARYING;
22 MI  DECLARE MACRO_FOUND BIT_1 INITIAL(FALSE);
23 MI  DECLARE CARD_COUNT INTEGER INITIAL(-1);
24 MI  DECLARE DISASTER LABEL;
25 MI  DECLARE NEXT_CHAN CHARACTER(1);
DECLARE OVER_FUNCH CHARACTER(1):
DECLARE BLANK_COUNT INTEGER INITIAL(0):
DECLARE CONTROL ARRAY(10) BIT(37):
DECLARE TCGIFS CHARACTER(10) INITIAL('1234567890');

THE ARRAY CARD_TYPE IN XFL IS REPLACED BY THIS FUNCTION

FUNCTION(SELECT) INTEGER:
DECLARE SELECT CHARACTER(1):

IF SELECT = 'K' OR SELECT = ' ' THEN RETURN 2;
IF SELECT = 'S' THEN RETURN 3;
IF SELECT = 'F' THEN RETURN 1;
IF SELECT = 'C' OR SELECT = 'M' OR SELECT = 'H' THEN RETURN 4;
RETURN 0;

CLOSE CARD_TYPE:

THE CHAR_INDEX FUNCTION IS THE SAME AS THE INDEX BUILT IN FUNCTION

CHAR_INDEX: FUNCTION(STRING, PATTERN) INTEGER:
DECLARE CHARACTER(*), STRING, PATTERN;
DECLARE INTEGER, I, J, K;
J = LENGTH(STRING);
K = LENGTH(PATTERN);
IF K > J THEN RETURN 0;
ELSE DC FOR I = 1 TO J-K+1;
IF STRING*(K+1) = PATTERN THEN RETURN 1;
END;
HAL Compilation -- Phase 1 -- Internmetrics, Inc.

Statement

41:    RETURN 0;
42:    CLOSE CHAR_INDEX;
43:    HERE IS PAD A FUNCTION FROM HALPASS!
44:    PAD: FUNCTION (STRING,WIDTH) CHARACTER(255) VARYING;
45:    DECLARE CHARACTER(*), STRING;
46:    DECLARE CHARACTER(255) VARYING, TEMP_STRING;
47:    DECLARE INTEGER, L, WIDTH;
48:    L = LENGTH(STRING);
49:    IF L < WIDTH THEN
50:      TEMP_STRING = STRING || X'05'(1 TO WIDTH-L);
51:    ELSE TEMP_STRING = STRING;
52:    RETURN TEMP_STRING;
53:    CLOSE PAD;
54:    HERE IS THE FUNCTION I_FORMAT FROM HALPASS!
55:    I_FORMAT: FUNCTION (NUMBER,WIDTH) CHARACTER(*);
56:    DECLARE CHARACTER(255) VARYING, STRING;
57:    DECLARE INTEGER, L, NUMBER, WIDTH;
58:    STRING = NUMBER;
59:    L = LENGTH(STRING);
60:    IF L < WIDTH THEN
61:      STRING = X'05'(1 TO WIDTH-L) || STRING;
62:    RETURN STRING;
63:    CLOSE I_FORMAT;
64:    THE FOLLOWING IS THE ERROR PROCEDURE FROM PASS1.
62 \textbf{ERRORS: PROCEDURE (MESSAGE, SEVERITY):} \par
63 \textbf{PRINTS AND ACCOUNTS FOR ALL ERRORS} \par
64 \textbf{DECLARE MSG CHARACTER(255) VARYING;} \par
65 \textbf{DECLARE MESSAGE CHARACTER(*) ;} \par
66 \textbf{DECLARE SEVERITY INTEGER ;} \par
67 \textbf{ERROl-COUNT = ERROR-COUNT + 1 ;} \par
68 \textbf{MSG = "$\ldots$ ERROR # $\ldots$ ERROR-COUNT \ldots OF SEVERITY $\ldots$ ;} \par
69 \textbf{SEVERITY $\ldots$ MESSAGE ;} \par
70 \textbf{IF ERROR-COUNT > 1 THEN} \par
71 \textbf{MSG = MSG $\ldots$ LAST ERROR ON LINE $\ldots$ PREVIOUS_ERROR ;} \par
72 \textbf{WRITE(6) MSG $\ldots$ "$\ldots$ ;} \par
73 \textbf{SAVE_SEVERITY(SERROR-COUNT) = SEVERITY ;} \par
74 \textbf{SAVE_LINES(SERROR-COUNT) = CAPC_COUNT ;} \par
75 \textbf{PREVIOUS_ERROR = CAPC_COUNT ;} \par
76 \textbf{IF ERROR-COUNT >= 99 THEN GO ;} \par
77 \textbf{IF ERROR-COUNT = 100 THEN} \par
78 \textbf{WRITE(6) "$\ldots$ TOO MANY ERRORS: COMPILATION ABORTED . ;} \par
79 \textbf{IFF COMPILING THEN COMPILING = FALSE ;} \par
80 \textbf{ELSE GO TO DISASTER ; /* THEN EAC IT THERE */} \par
81 \textbf{END ;} \par
82 \textbf{IF SEVERITY $>$ MAX_SEVERITY THEN MAX_SEVERITY = SEVERITY ;} \par
83 \textbf{IF SEVERITY $>$ STATEMENT-SEVERITY THEN} \par
84 \textbf{STATEMENT-SEVERITY = SEVERITY ;} \par
85 \textbf{IF SEVERITY $>$ 2 THEN IFF COMPILING THEN} \par
86 \textbf{COMPILING = FALSE ;} \par
87 \textbf{ELSE GO TO DISASTER ;} \par
88 \textbf{END OF PASS1 ERROR} \par
CLOSE ERRORS: /* CLOSING ERRORS AND RETURNING. */

THIS NEXT PROCEDURE IS PRINT_SUMMARY

PRINT_SUMMARY: PROCEDURE;

WRITE(6) 'CARDS WERE PROCESSED';

IF ERR_COUNT = 0 THEN WRITE(6) 'NO ERRORS WERE DETECTED.';

ELSE IF ERR_COUNT > 1 THEN DC;

WRITE(6) 'ERRORS WERE DETECTED.';

THE LAST ERROR WAS ON LINE 11 PREVIOUS_ERROR;

WRITE(6) 'SUMMARY OF DETECTED ERRORS....';

FOR I = 1 TO ERR_COUNT;

WRITE(6) 'ERROR # I I I I ON LINE 11';

SAVE_LINE$ I I ' OF SEVERITY I I',

SAVE_SEVERITY$ I I';

END;

END;

ELSE WRITE(6) 'ONE ERROR WAS DETECTED WHICH OCCURRED ON LINE 11';

PREVIOUS_ERROR I I ' OF SEVERITY I I SAVE_SEVERITY$ I I';

TIME2 = TIME;

I = TIME2 - TIME1;

WRITE(6) 'CARD-PROCESSING RATE: I I 6000 CARD_COUNT/I I'

' CARDS PER MINUTE.';

WRITE(6) 'CLOCK TIME IS I I TIME2';

RETURN;

CLOSE PRINT_SUMMARY;

STREAM: PROCEDURE;

THIS PROCEDURE FILLS THE VARIABLES NEXT_CHAR,

ARROW, AND OVER_HUNCH.
I-NTER M-E-TR I-C-S, I-N C.

SOURC.E.

101 M
102 M
103 M
104 M
105 M
106 M
107 M
108 M
109 M
110 M
111 M
112 M
113 M
114 M
115 M
116 M
117 M
118 M
119 M
120 M
121 M
122 M
123 M
124 M
125 M
126 M
127 M
128 M
129 M
130 M
131 M
132 M
133 M
134 M
135 M
136 M
137 M
138 M
139 M
140 M
141 M
142 M
143 M
144 M
145 M
146 M
147 M
148 M
149 M
150 M
151 M
152 M
153 M
154 M
155 M
156 M
157 M
158 M
159 M
160 M
161 M
162 M
163 M
164 M
165 M
166 M
167 M
168 M
169 M
170 M
171 M
172 M
173 M
174 M
175 M
176 M
177 M
178 M
179 M
180 M
181 M
HAL COMPILATION -- PHASE I -- INTERMETRICS, INC.

```
<table>
<thead>
<tr>
<th>STMT</th>
<th>SOURCE</th>
<th>LINE</th>
<th>CURRENT SCOPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>115</td>
<td>GO TO STREAM_START;</td>
<td>182</td>
<td>STREAM</td>
</tr>
<tr>
<td>116</td>
<td>PROCESS_COMMENT: PROCEDURE;</td>
<td>183</td>
<td>STREAM</td>
</tr>
<tr>
<td>117</td>
<td>DECLARE K: CHARACTER(1);</td>
<td>184</td>
<td>PROCESS_COMMENT</td>
</tr>
<tr>
<td>118</td>
<td>DECLARE J: INTEGER;</td>
<td>185</td>
<td>PROCESS_COMMENT</td>
</tr>
<tr>
<td>119</td>
<td>IF CURRENT_CARDS1 = 'T' THEN</td>
<td>186</td>
<td>PROCESS_COMMENT</td>
</tr>
<tr>
<td>119</td>
<td>WRITE(A): LINE(1), CURRENT_CARDS(2 TO 4) /* ISSUE A NEW HEADER*/</td>
<td>187</td>
<td>PROCESS_COMMENT</td>
</tr>
<tr>
<td>120</td>
<td>ELSE</td>
<td>188</td>
<td>PROCESS_COMMENT</td>
</tr>
<tr>
<td>120</td>
<td>IF CURRENT_CARDS1 = 'C' THEN</td>
<td>189</td>
<td>PROCESS_COMMENT</td>
</tr>
<tr>
<td>120</td>
<td>DC FOR I=1 TO TEXT_LIMIT-1;</td>
<td>190</td>
<td>PROCESS_COMMENT</td>
</tr>
<tr>
<td>121</td>
<td>IF CURRENT_CARDS1 = ' ' THEN DO;</td>
<td>191</td>
<td>PROCESS_COMMENT</td>
</tr>
<tr>
<td>122</td>
<td>K = CURRENT_CARDS(I+1);</td>
<td>192</td>
<td>PROCESS_COMMENT</td>
</tr>
<tr>
<td>123</td>
<td>J = CHAR_INDEX(TOGLIES, K);</td>
<td>193</td>
<td>PROCESS_COMMENT</td>
</tr>
<tr>
<td>124</td>
<td>IF J = 0 THEN DO;</td>
<td>194</td>
<td>PROCESS_COMMENT</td>
</tr>
<tr>
<td>125</td>
<td>IF I &lt; TEXT_LIMIT-1 THEN</td>
<td>195</td>
<td>PROCESS_COMMENT</td>
</tr>
<tr>
<td>125</td>
<td>K = CURRENT_CARDS(I+2);</td>
<td>196</td>
<td>PROCESS_COMMENT</td>
</tr>
<tr>
<td>126</td>
<td>ELSE GO TO COMPLEMENT;</td>
<td>197</td>
<td>PROCESS_COMMENT</td>
</tr>
<tr>
<td>127</td>
<td>IF K = '*' THEN CONTROL = TRUE;</td>
<td>198</td>
<td>PROCESS_COMMENT</td>
</tr>
<tr>
<td>127</td>
<td>J:</td>
<td>199</td>
<td>PROCESS_COMMENT</td>
</tr>
<tr>
<td>129</td>
<td>ELSE IF K = '-' THEN CONTROL = FALSE;</td>
<td>200</td>
<td>PROCESS_COMMENT</td>
</tr>
<tr>
<td>129</td>
<td>J:</td>
<td>201</td>
<td>PROCESS_COMMENT</td>
</tr>
<tr>
<td>129</td>
<td>ELSE</td>
<td>202</td>
<td>PROCESS_COMMENT</td>
</tr>
<tr>
<td>129</td>
<td>COMPLEMENT: CONTROL = NOT CONTROL;</td>
<td>203</td>
<td>PROCESS_COMMENT</td>
</tr>
<tr>
<td>130</td>
<td>END;</td>
<td>204</td>
<td>PROCESS_COMMENT</td>
</tr>
<tr>
<td>130</td>
<td>END;</td>
<td>205</td>
<td>PROCESS_COMMENT</td>
</tr>
<tr>
<td>132</td>
<td>END;</td>
<td>206</td>
<td>PROCESS_COMMENT</td>
</tr>
<tr>
<td>133</td>
<td>CLOSE PROCESS_COMMENT;</td>
<td>207</td>
<td>PROCESS_COMMENT</td>
</tr>
</tbody>
</table>
```
STACK_RETURN_CHAR:

PROCEDURE(NUMBER, CHARD):

DECLARE INTEGER, NUMBER, I;

DECLARE CHAR, CHARACTER(1);

DC FOR I = 1 TO ?;

IF RETURN_CHAR = 0 THEN DO;
  I:

RETURN_CHAR = NUMBER;

TYPE_CHAR = CHAR;

I:

THE FOLLOWING STATEMENT IS FOR DEBUGGING PURPOSES ONLY.
GLT 'STACKED' || CHAR || '()' || NUMBER || ')' = INDEX(' || I || ');

RETURN;

END:

CLOSE STACK_RETURN_CHAR;

READ_CARD:

PROCEDURE:

DECLARE END_OF_INPUT BIT_1 INITIAL(FALSE);

IF END_OF_INPUT THEN DO:

CURRENT_CARD = INPUT_PAC;

RETURN;

END:

READ(CURRENT_CARD);

END_OF_INPUT = TRUE;

CARD_COUNT = CARD_COUNT + 1;
HAL COMPILATION -- PHASE 1 -- INTERMETRICS, INC.

LINE CURRENT SCOPE
1 243 READ_CARD
1 244 READ_CARD
1 245 READ_CARD
1 246 READ_CARD
1 247 READ_CARD
1 248 STREAM
1 249 STREAM
1 250 STREAM
1 251 STREAM
1 252 ORDER_OK
1 253 ORDER_OK
1 254 ORDER_OK
1 255 ORDER_OK CASE 1
1 256 ORDER_OK
1 257 ORDER_OK
1 258 ORDER_OK
1 259 ORDER_OK
1 260 ORDER_OK CASE 2
1 261 ORDER_OK
1 262 ORDER_OK
1 263 ORDER_OK
1 264 ORDER_OK
1 265 ORDER_OK
1 266 ORDER_OK
1 267 ORDER_OK CASE 3
1 268 ORDER_OK CASE 4
1 269 ORDER_OK
1 270 ORDER_OK

STMT  SOURCE
152 M1 IF CARD_COUNT NOT = 0 THEN DO;
153 M1 WRITE(6) I_FORMAT(CARD_COUNT, 4)
154 M1 I I I SAVE_CARD1 I I I SAVE_CARD2 TO #1 I I I CARD_COUNT;
155 M1 END;
156 M1 SAVE_CARD = CURRENT_CARD;
157 M1 CLOSE READ_CARD;
158 M1 ORDER_CK;
159 M1 FUNCTIONTYPE) BIT_1;
160 M1 DECLARE TYPE CHARACTER(1);
161 M1 CC CASE CARD_TYPE(CURRENT_CARD$1) + 1;
162 M1 DO: /*CASE 1--ILLEGAL CARD_TYPE*/
163 M1 END_GROUP = FALSE;
164 M1 RETURN FALSE;
165 M1 END: /*OF CASE 1*/
166 M1 /* CASE 2--E CARD*/
167 M1 E CARD:
168 M1 DD;
169 M1 IF CARD_TYPE(TYPE) = 2 OR CARD_TYPE(TYPE) = 3 THEN END_GROUP = TRUE;
170 M1 ELSE END_GROUP = FALSE;
171 M1 RETURN TRUE;
172 M1 END: /*CASE 2*/
173 M1 /*CASE 3--M CARD*/ GC TO E CARD;
174 M1 DD: /*CASE 4--S CARD*/
175 M1 END_GROUP = FALSE;
176 M1 IF CARD_TYPE(TYPE) = 2 OR
STMT      SOURCE
174 MI  ELSE RETURN FALSE;
175 MI  END: /* CASE */
176 MI  CC: /* CASE 5 /* COMMENT */
177 MI  IF CARD_TYPE(TYPE) = 2 OR CARD_TYPE(TYPE) = 3 THEN END_GROUP = TRUE;
178 MI  ELSE END_GROUP = FALSE;
179 MI  IF CURRENT_CARD$1 = 'P' THEN DO;
180 MI  IF TYPE = 'C' THEN RETURN TRUE;
181 MI  ELSE RETURN FALSE;
182 MI  ENC;
183 MI  ELSE DD;
184 MI  IF CARD_TYPE(TYPE) = 1 THEN RETURN FALSE;
185 MI  ELSE RETURN TRUE;
186 MI  ENC;
187 MI  ENC: /* CASE 5 */
188 MI  ENC: /* CF DD CASE */
189 MI  CLOSE ORDER_CK;
190 MI  COMMENT:
191 MI  FUNCTION BIT_1;
192 MI  IF CURRENT_CARD$CP = ';' THEN
193 MI  IF CP < TEXT_LIMIT THEN
194 MI  IF CURRENT_CARDS(CP + 1) = ';' THEN DD;
195 MI  LOC FOR CP = CP+2 TO TEXT_LIMIT;
196 MI  IF CURRENT_CARD$CP = ';' THEN
HAL COMPILATION -- PHASE 1 -- INTERMETRICS, INC.

157 M1 IF CURRENT_CARES(CP+1) = '/' THEN
159 M1 RETURN_TRUE;
160 M1 END;
162 M1 END;
163 M1 GO TO RETURN_TRUE;
170 M1 RETURN_FALSE;
171 M1 DECLARE COMMENT:
173 M1 SCAN_CARD:
201 M1 PROCEDURE(TYPE,COUNT) ASSIGN(LIN,INDICATOR):
202 M1 DECLARE INDICATOR ARRAY(256) INTEGER;
204 M1 DECLARE INTEGER, TYPE, COUNT;
205 M1 DECLARE LIN CHARACTER(*);
206 M1 DO FOR CP = 2 TO TEXT_LIMIT;
207 M1 IF CURRENT_CARES_CP NOT = ' ' THEN GO;
209 M1 IF COMMENT THEN GO TO CONTINUE;
210 M1 IF LIN$CP NOT = ' ' THEN
211 M1 CALL ERRORS('OVERLAPPING E-LINE CHARACTERS',1);
212 M1 CALL ERRORS('OVERLAPPING S-LINE CHARACTERS',1);
213 M1 GO TO CONTINUE;
214 M1 END; /*CF CASE 2*/
PROCEDURE(TYPE) ASSIGN(LIN,INDICATOR,COUNT);
DECLARE PCINT CHARACTER(1);
DECLARE INTEGER, COUNT, TYPE;
DECLARE CHARACTER(#), LIN;
DECLARE ARRAY(256) INTEGER, INDICATOR;
IF TYPE = 1 THEN PCINT = 'S';
ELSE PCINT = 'E';
COUNT = 1;
DO FOREVER;
CALL SCAN_CARD(TYPE,COUNT) ASSIGN
(LIN,INDICATOR);
CALL READ_CARD;
IF CURRENT_CARD$1 NOT = PCINT THEN DO;
AC MORE OF THIS TYPE CARD.
IF NOT ORDER_CK(PCINT) THEN
CALL ERRORS(CRDER_ERR,1);
DO FOR CP = 2 TO TEXT_LIMIT;
IF LIN$CP = '' THEN
<table>
<thead>
<tr>
<th>STMT</th>
<th>SOURCE</th>
<th>LINE</th>
<th>CURRENT SCOPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>225</td>
<td>INDI HP $CP = 0;</td>
<td>355</td>
<td>COMP</td>
</tr>
<tr>
<td>226</td>
<td>ELSE IF TYPE = 0 THEN CC;</td>
<td>356</td>
<td>COMP</td>
</tr>
<tr>
<td>227</td>
<td>INDI HP $CP = COUNT - INDI HP $CP + 1;</td>
<td>357</td>
<td>COMP</td>
</tr>
<tr>
<td>228</td>
<td>END;</td>
<td>358</td>
<td>COMP</td>
</tr>
<tr>
<td>229</td>
<td>END;</td>
<td>359</td>
<td>COMP</td>
</tr>
<tr>
<td>240</td>
<td>RETURN;</td>
<td>360</td>
<td>COMP</td>
</tr>
<tr>
<td>241</td>
<td>END;</td>
<td>361</td>
<td>COMP</td>
</tr>
<tr>
<td>242</td>
<td>COUNT = COUNT + 1;</td>
<td>362</td>
<td>COMP</td>
</tr>
<tr>
<td>243</td>
<td>END;</td>
<td>363</td>
<td>COMP</td>
</tr>
<tr>
<td>244</td>
<td>CLCSF COMP;</td>
<td>364</td>
<td>STREAM</td>
</tr>
<tr>
<td>245</td>
<td>GFT_GROUP:</td>
<td>365</td>
<td>STREAM</td>
</tr>
<tr>
<td>246</td>
<td>PROCEDURE</td>
<td>366</td>
<td>STREAM</td>
</tr>
<tr>
<td>247</td>
<td>E_LINE = E_LINE$INDEX TO #)</td>
<td></td>
<td>BLANKS;</td>
</tr>
<tr>
<td>248</td>
<td>S_LINE = S_LINE$INDEX TO #)</td>
<td></td>
<td>BLANKS;</td>
</tr>
<tr>
<td>249</td>
<td>LAST_E_COUNT = E_COUNT;</td>
<td>369</td>
<td>GET_GROUP</td>
</tr>
<tr>
<td>250</td>
<td>LAST_S_COUNT = S_COUNT;</td>
<td>370</td>
<td>GET_GROUP</td>
</tr>
<tr>
<td>251</td>
<td>E_COUNT, S_COUNT = 0;</td>
<td>371</td>
<td>GET_GROUP</td>
</tr>
<tr>
<td>252</td>
<td>GO TO LOOP;</td>
<td>372</td>
<td>GET_GROUP</td>
</tr>
<tr>
<td>253</td>
<td>READ IT;</td>
<td>373</td>
<td>GET_GROUP</td>
</tr>
<tr>
<td>254</td>
<td>CALL READ_CARD;</td>
<td>374</td>
<td>GET_GROUP</td>
</tr>
<tr>
<td>255</td>
<td>IF NOT ORDER CHK(PREV_CARD) THEN CC;</td>
<td>375</td>
<td>GET_GROUP</td>
</tr>
<tr>
<td>256</td>
<td>CALL ERRCHK(ERRCHK_ERR, 1);</td>
<td>376</td>
<td>GET_GROUP</td>
</tr>
<tr>
<td>257</td>
<td>GO TO READ IT;</td>
<td>377</td>
<td>GET_GROUP</td>
</tr>
<tr>
<td>258</td>
<td>END;</td>
<td>378</td>
<td>GET_GROUP</td>
</tr>
<tr>
<td>259</td>
<td>LOOP;</td>
<td>379</td>
<td>GET_GROUP</td>
</tr>
<tr>
<td>260</td>
<td>IF END- GROUP THEN GO TO FCUND- GROUP;</td>
<td>380</td>
<td>GET_GROUP</td>
</tr>
<tr>
<td>261</td>
<td></td>
<td>381</td>
<td>GET_GROUP</td>
</tr>
<tr>
<td>262</td>
<td></td>
<td>382</td>
<td>GET_GROUP</td>
</tr>
</tbody>
</table>
259 M1 EN CASE CARD_TYPE(CURRENT_CARDS1) + 1;
260 M1 DO: /*CASE 1, A DUMMY*/ END;
261 M1 DC: /*CASE 2 E-LIN*/
262 M1 CALL CCMP(0) ASSIGN
263 M1 (E_LINE, E_INDICATOR, E_COUNT);
264 M1 CC TO LOOP;
265 M1 END: /*OF CASE 2*/
266 M1 DC: /*CASE 3 S-LIN*/
267 M1 M_LINE = M_LINES(INDEX TO 3) || CURRENT_CARDS(2 TO 3);
268 M1 SAVE_CARDS1 = 'Y';
269 M1 PREV_CARD = CURRENT_CARDS1;
270 M1 GO TO READ_IT;
271 M1 EN: /*OF CASE 3*/
272 M1 DC: /*CASE 4 S-LIN*/
273 M1 CALL CCMP(1) ASSIGN
274 M1 (S_LINE, S_INDICATOR, S_COUNT);
275 M1 GO TO LOOP;
276 M1 EN: /*OF CASE 4*/
277 M1 DC: /*CASE 5 COMMENT*/
278 M1 PREV_CARD = CURRENT_CARDS1;
279 M1 CALL PROCESS_COMMENT;
280 M1 GO TO READ_IT;
281 M1 EN: /*OF CASE 5*/
282 M1 DC: /*CASE 6 DO CASE*/
283 M1 CALL END_GROUP;
284 M1 WRITE(6) SKIP(1);
285 M1 END_GROUP = FALSE;
286 M1 E_LINE = E_LINES(1 TO LENGTH(M_LINE));
IN METRIS, ST VT 2846

H 8.5 M'

27'42

2'9

29 V,

20 AM 204

29;5

66 Ml

12

27

227

0.1

320

3o? 03

304

3Cc

THE FOLLOWING STATEMENTS ARE FOR DEBUGGING PURPOSES ONLY

OUT "E_LINE="|| E_LINE || "";

OUT "M_LINE="|| M_LINE || "";

OUT "S_LINE="|| S_LINE || "";

CLOSE GET_GROUP;

PROCEDURE;

PROCEDURE;

INDEX = INDEX + 1;

IF INDEX = TEXT_LIMIT THEN FC;

OUT OF DATA. GET MORE.

E_INDICATOR$1 = E_INDICATOR$TEXT_LIMIT;

S_INDICATOR$1 = S_INDICATOR$TEXT_LIMIT;

CALL GET_GROUP;

INDEX = 1;

ENDIF;
CLOSE CHOP;

PROCEDURE(TYPE, INDICATOR, LIN) ASSIGN
(INC, STACK, PP):

DECLARE INTEGER, TYPE, PP:

DECLARE CHARACTER(*), LIN, STACK:

DECLARE ARRAY(256) INTEGER, INDICATOR, IND:

IF PP < 1 THEN GO TO NOT_MULTIPLE;

IF LIN$INDX = ' ' THEN DO:

IF STACK$PP = ' ' THEN

IND$PP = IND$PP + 1:

ELSE GO TO NOT_MULTIPLE;

END;

ELSE DO:

NOT_MULTIPLE:

PP = PP + 1;

IF PP > 256 THEN DO CASE TYPE+1:

CALL ERRORS('EXPONENT STRING OVER FLOW', 3);

CALL ERRORS('SUBSCRIPT STRING OVER FLOW', 3);

END; /*CF DO CASE*/

STACK = STACK || LIN$INDX:

IND$PP = INDICATOR$INDX:

END;

CLOSE STACK:

BUILD_XSCRIPTS:
PROCEDURE;
E_STACK, S_STACK = ' ';
E_BLANKS, S_BLANKS = -1;
EP, SP = 0;
CHECK_M:
IF P_LINE$INDEX = ' ' THEN
PROCESS:
DO;
CALL STACK(0, E_INDICATOR, E_LINE) ASSIGN
(E_INC, E_STACK, EP):
CALL STACK(1, S_INDICATOR, S_LINE) ASSIGN
(S_INC, S_STACK, SP);
CALL CHOP;
GO TO CHECK_M;
END;
ELSE /* A NON BLANK ON S-LINE*/
DC;
IFF M_COMMENT THEN DC;
IF M_LINE$INDEX = '*#' THEN
IF M_LINE$(INDEX +1) = '/' THEN DC;
M_COMMENT = FALSE;
M_LINE$(INDEX +1) = ' ':
END;
GO TO PROCESS;
END;
IF M_LINE$INDEX = '/' THEN
IF M_LINE$(INDEX +1) = '*#' THEN DO;
M_COMMENT = TRUE;

SOURCE
LINE CURRENT SCOPE
470 BUILD_XSCRIPTS
471 BUILD_XSCRIPTS
472 BUILD_XSCRIPTS
473 BUILD_XSCRIPTS
474 BUILD_XSCRIPTS
475 BUILD_XSCRIPTS
476 BUILD_XSCRIPTS
477 BUILD_XSCRIPTS
478 BUILD_XSCRIPTS
479 BUILD_XSCRIPTS
480 BUILD_XSCRIPTS
481 BUILD_XSCRIPTS
482 BUILD_XSCRIPTS
483 BUILD_XSCRIPTS
484 BUILD_XSCRIPTS
485 BUILD_XSCRIPTS
486 BUILD_XSCRIPTS
487 BUILD_XSCRIPTS
488 BUILD_XSCRIPTS
489 BUILD_XSCRIPTS
490 BUILD_XSCRIPTS
491 BUILD_XSCRIPTS
492 BUILD_XSCRIPTS
493 BUILD_XSCRIPTS
494 BUILD_XSCRIPTS
495 BUILD_XSCRIPTS
496 BUILD_XSCRIPTS
497 BUILD_XSCRIPTS
498 BUILD_XSCRIPTS
499 BUILD_XSCRIPTS
500 BUILD_XSCRIPTS
STMT          SOURCE          LINE       CURRENT SCOPE
246 M1 M_LINE[1:(ADRX *1)] = "";
247 M1 GC TO PROCESS;
248 M1 END;
249 M1 IF S_STACK$SP = "" THEN DO;
250 M1 IF SP > 1 THEN S_STACK = S_STACK$(1 TO SP-1);
251 M1 ELSE S_STACK = "";
252 M1 S_BLANKS = S_IND$SP;
253 M1 END;
254 M1 IF E_STACK$EP = "" THEN DO:
255 M1 IF EP > 1 THEN E_STACK = E_STACK$(1 TO EP-1);
256 M1 ELSE E_STACK = "";
257 M1 E_BLANKS = E_IND$EP;
258 M1 END;
259 M1 IF E_BLANKS >= S_BLANKS THEN
260 M1 M_BLANKS = S_BLANKS;
261 M1 ELSE M_BLANKS = E_BLANKS;
262 M1 CLOSE BUILD_XSCRIPTS:
263 M1 STREAM_START:
264 M1 IF FIRST_CALL_TO_STREAM THEN DC;
265 M1 TIME1 = TIME;
266 M1 FIRST_CALL_TO_STREAM = FALSE;
267 M1 AGAIN:
268 M1 CALL READ_CARP;
STMT | SOURCE | LINE | CURRENT SCOPE
--- | --- | --- | ---
247 | IF_NOT CRDEP_CHK('C') THEN DO; | 527 | STREAM
248 | CALL ERRORS(CRDEP_ERR,1); | 528 | STREAM
249 | GO TO AGAIN; | 529 | STREAM
250 | FND; | 530 | STREAM
251 | TEXT_LIMIT = LENGTH(CURRENT_CARD); | 531 | STREAM
252 | RETURNING_M = TRLF; | 532 | STREAM
253 | N_LINE, E_LINE, S_LINE = ' '; | 533 | STREAM
254 | INDX = 1; | 534 | STREAM
255 | CALL GET_GROUP; | 535 | STREAM
256 | N_BLANKS = -1; | 536 | STREAM
257 | INDX = 2; | 537 | STREAM
258 | II = 1; | 538 | STREAM
259 | ENC; | 539 | STREAM
260 | IFF MACRO_FOUND THEN DC; | 540 | STREAM
261 | IF MACRO_POINT < MACRO_LIMIT THEN DO; | 541 | STREAM
262 | NEXT_CHAR = | 542 | STREAM
263 | MACROSTREAM$MACRO_POINT; | 543 | STREAM
264 | MACRO_POINT = MACRO_POINT +1; | 544 | STREAM
265 | RETURN; | 545 | STREAM
266 | ENC; | 546 | STREAM
267 | IF SAVE_BLANK_COUNT >= 0 THEN DC; | 547 | STREAM
268 | NEXT_CHAR = ' '; | 548 | STREAM
269 | BLANK_COUNT = SAVE_BLANK_COUNT; | 549 | STREAM
270 | SAVE_BLANK_COUNT = -1; | 550 | STREAM
271 | RETURN; | 551 | STREAM
272 | ENC; | 552 | STREAM
273 | MACRO_FOUND = FALSE; | 553 | STREAM
274 | NEXT_CHAR = SAVE_NEXT_CHAR; | 554 | STREAM
OVER_PUNCH = SAVE_OVER_PUNCH;
MACRO_STREAM = "";
RETURN;
FAI;
BLANK_COUNT = -1;
STACK_CHECK:
DC FOR II = II TC 3;/CHECK LIMIT/
IF RETURN_CHAR NOT = 0 THEN DO;
RETURN_CHAR = RETURN_CHAR -1;
ARROW_FLAG = TRUF;
RETURN_CHAR = RETURN_CHAR -1;
NEXT_CHAR = TYPE_CHAR ;
OVER_PUNCH = 0;
RETURN;
END;
END;
II = 1;
IFF ARROW_FLAG THEN DC;
ARROW_FLAG = FALSE;
NEXT_CHAR = SAVE_NEXT_CHAR;
FVF_PUNCH = SAVE_OVER_PUNCH;
BLANK_COUNT = SAVE_BLANK_COUNT;
RETURN;
END;
BEGINING:
IFF RETURNING THEN DC:
IFF M_BLANKS >= 0 THEN DO:
HAL COMPILATION -- PHASE 1 -- INTERMETRICS, INC.

<table>
<thead>
<tr>
<th>STATEMENT</th>
<th>SOURCE</th>
</tr>
</thead>
</table>
| 418 | NEXT_CHAR = ' ';
| 419 | ARROW = LAST_E_IND;
| 420 | LAST_E_IND = C;
| 421 | FLANK_COUNT = P_BLANKS;
| 422 | P_BLANKS = -1;
| 423 | GC TO FLANK_CHAR;
| 424 | GC;
| 425 | IF M_LINES(INDX NOT = ' ' THEN DO;
| 426 | IF M_LINES(INDX = '/' THEN
| 427 | IF M_LINES(INDX +1) = 'e' THEN DO;
| 428 | HERE IS A START OF A COMMENT
| 429 | M_COMMENT = TRUE;
| 430 | M_LINES(INDX = ' ';
| 431 | M_LINES(INDX +1) = ' ';
| 432 | GC TO BLANK;
| 433 | GC;
| 434 | IF E_COUNT = 0 THEN DO;
| 435 | IF E_LINES(INDX NOT = ' ' THEN DO:
| 436 | IF E_INDICATOR(INDX NOT = 1 THEN
| 437 | CALL ERRORS('E-LINE OVERLAPS M-LINE',1);
| 438 | ELSE OVER_PUNCH = E_LINES(INDX;
| 439 | ENC;
| 440 | ELSE OVER_PUNCH = 0;
| 441 | ENC;
| 442 | ELSE OVER_PUNCH = 0;
| 443 | IF S_COUNT = 0 THEN
| 444 | IF S_LINES(INDX NOT = ' ' THEN
| 445 | CALL ERRORS('S-LINE OVERLAPS M-LINE',1);

PAGE 21
HAL COMPILATION  --  PHASE 1  --  INTERMETRICS, INC.

STMT
441 M1 ARROW = LAST_E_IND;
442 M1 LAST_E_IND = C;
443 M1 NEXT_CHAR = $LINE\$TRX;
444 M1 CALL CHCP;
445 M1 GC TO FIND_CHAR;
446 M1 END;
447 M1 ELSE
448 M1 PLANK:
449 M1 DO:
450 M1 CALL BUILD_XSCRIPTS:
451 M1 OVER_PUNCH = 0;
452 M1 RETURNING_M = FALSE;
453 M1 LAST_S_IND = 0;
454 M1 RETURNING_S = TRUE;
455 M1 PCINTER = 1; /* THIS ONE MAY NOT BE NEEDED*/
456 M1 END;
457 M1 IFF RETURNING_S THEN DC;
458 M1 IF LENGTH(S_STACK) > 0 &
459 M1 PCINTER <= LENGTH(S_STACK) THEN DC;
460 M1 IF S_STACK$PCINTER = " " THEN DC;
461 M1 IF S_INDEX$POINTER = " " THEN MORE LEFT
462 M1 DC;
463 M1 NEXT_CHAR = " ";
464 M1 BLANK_COUNT = S_INDEX$POINTER;
465 M1 FCINTER = POINTER + 1;
466 M1 ARROW = LAST_S_IND - S_INDEX$POINTER;
LAST_S_IND = S_INDBPOINTER;
END;
FLSE DC; /*A NON BLANK*/
NEXT_CHAR = S_STACKPOINTER;
ARROW = LAST_S_IND - S_INDBPOINTER;
LAST_S_IND = S_INDBPOINTER;
POINTER = POINTER + 1;
END;
GO TO FOUND_CHAR;
END;
FLSE DC; /* CAN NOT RETURN*/
RETURNING_S = FALSE;
RETURNING_E = TRUE;
LAST_E_IND = -LAST_S_IND;
POINTER = 1;
END;
IFF RETURNING_E THEN CC;
IFF LENGTH(E_STACK) > 0 &
FCINTER <= LENGTH(E_STACK) THEN CC;
IFF E_STACKPOINTER = '=' THEN DO;
IFF E_INDBPOINTER >= 0 THEN
MORE TO GO.
DC:I
NEXT_CHAR = ' ';
BLANK_COUNT = E_INDBPOINTER;
POINTER = POINTER + 1;
HAL COMPILATION -- PHASE I -- INTERMETRICS, INC.

LINE CURRENT SCOPE

SOURCE

STM7

489 M1 ARROW = E_IND$POINTER - LAST_E_IND;

490 M1 LAST_E_IND = E_IND$POINTER;

491 M1 END;

492 M1 END;

493 M1 ELSE

CL A NON BLANK

497 M1 DO;

494 M1 NEXT_CHAR = E_STACK$POINTER;

495 M1 ARROW = E_IND$POINTER - LAST_E_IND;

496 M1 LAST_E_IND = E_IND$POINTER;

497 M1 POINTER = POINTER + 1;

498 M1 END;

499 M1 GC TO FOUND_CHAR;

500 M1 END;

501 M1 ELSE

CL CAN NOT RETURN

501 M1 DO:

502 M1 RETURNING_E = FALSE;

502 M1 RETURNING_M = TRUE;

503 M1 FND;

503 M1 FND;

504 M1 FND;

505 M1 END;

506 M1 GO TO BEGINING;

507 M1 FOUND_CHAR;

507 M1 IF ARROW NOT = 0 THEN DC;

508 M1 OLD_LEVEL = NEW_LEVEL;

509 M1 NEW_LEVEL = NEW_LEVEL + ARROW;

510 M1 SAVE_OVER_PUNCH = OVER_PUNCH;

511 M1 SAVE_NEXT_CHAR = NEXT_CHAR;

PAGE 24
HAL COMPILATION -- PHASE 1 -- INTERMETRICS, INC.

STMT  SOURCE  LINE  CURRENT  SCOPE

512 M1  SAVE_BLANK_COUNT = BLANK_COUNT;  | 696  STREAM
513 M1  IF OLD_LEVEL > 0 THEN DC;  | 697  STREAM
514 M1  IF ARROW < 0 THEN  | 698  STREAM
514 M1  CALL STACK_RETURN_CHAR(ARROW,'');  | 699  STREAM
515 M1  ELSE  | 700  STREAM
515 M1  EXPONENT:  | 701  STREAM
515 M1  DU;  | 702  STREAM
516 M1  IF ARROW > 1 THEN  | 703  STREAM
516 M1  CALL ERRORS(LINE_ERR,1);  | 704  STREAM
517 M1  CALL STACK_RETURN_CHAR(2,'*');  | 705  STREAM
518 M1  CALL STACK_RETURN_CHAR(ARROW,'');  | 706  STREAM
519 M1  END;  | 707  STREAM
520 M1  END;  | 708  STREAM
521 M1  ELSE IF OLD_LEVEL = 0 THEN DO;  | 709  STREAM
522 M1  IF ARROW < 0 THEN  | 710  STREAM
522 M1  SUBS:  | 711  STREAM
522 M1  DU;  | 712  STREAM
523 M1  IF ARROW < -1 THEN CALL ERRORS(  | 713  STREAM
523 M1  'S-LINE CHARACTER MORE THAN 1 LINE BELOW PRECEDING CHARACTER',1);  | 714  STREAM
524 M1  CALL STACK_RETURN_CHAR(1,'$');  | 715  STREAM
525 M1  CALL STACK_RETURN_CHAR(ARROW,'');  | 716  STREAM
526 M1  END;  | 717  STREAM
527 M1  ELSE GC TO EXPONENT;  | 718  STREAM
528 M1  END;  | 719  STREAM
529 M1  ELSE /*OLD < 0*/ DO:  | 720  STREAM
530 M1  IF ARROW < 0 THEN GO TO SUBS;  | 721  STREAM
531 M1  IF NEW_LEVEL <= 0 THEN  | 722  STREAM
531 M1  CALL STACK_RETURN_CHAR(ARROW,'');  | 723  STREAM
ELSE DC;
532 M1 CALL STACK_RETURN_CHAR(-CLC_LEVEL,'');
533 M1 IF NEW_LEVEL > 1 THEN CALL ERRORS(E_LINE_ERR,1);
534 M1 CALL STACK_RETURN_CHAR(2,'*');
535 M1 CALL STACK_RETURN_CHAR(NEW_LEVEL,'(');
536 M1 END;
537 M1 END;
538 M1 ARROW = 0;
539 M1 GO TO STACK_CHECK;
540 M1 FNG;
541 M1 RETURN;
542 M1 CLOSE STREAM;
543 M1 PROGRAM TO TEST THE STREAM PROCEDURE
544 M1 MAIN_PROGRAM:
545 M1 WRITE(6) 'BEGIN TEST OF HAL IN HAL';
546 M1 BUILT, BUILT_UP, BUILT_TOKEN, BUILT_C_P='';
547 M1 MAIN_LOOP: DC FCREVER;
548 M1 CALL STREAM;
549 M1 IF NEXT_CHAR = '' THEN DC:
550 M1 IF BLANK_FLAG THEN GO TO MAIN_LOOP;
551 M1 ELSE BLANK_FLAG = TRUE;
552 M1 END;
553 M1 ELSE BLANK_FLAG = FALSE:
554 M1 IF OVER_PUNCH = '0' THEN DC;
555 M1 OUT 'STREAM RETURNS ' || NEXT_CHAR || '*** *** ** BLANKS=';
556 M1 || BLANK_COUNT;
Hal Compilation -- Phase 1 -- Intermetrics, Inc.

```
555 M] BUILT_UP = BUILT_UP || ' ';
556 M] END;
557 M] ELSE DO;
558 M] OUT 'STREAM_Returns "" || NEXT_CHAR || ";" - OVER_PUNCH "" || 
          " " - OVER_PUNCH || " " - BLANKS=" " BLANK_COUNT;
559 M] BUILT_UP = BUILT_UP || OVER_PUNCH;
560 M] END;
561 M] IF NEXT_CHAR NOT < 'A' THEN DO;
562 M] BUILT_TOKEN = BUILT_TOKEN || NEXT_CHAR;
563 M] END;
564 M] ELSE DO;
565 M] IF LENGTH(BUILT_TOKEN) NOT = 0 THEN DO;
566 M] IF BUILT_D_P = ' ' THEN WRITE(6) '***TOKEN=' || BUILT_TOKEN;
567 M] ELSE WRITE(6) '***TOKEN=' || BUILT_TOKEN || 'MARKER=' || BUILT_D_P;
568 M] BUILT_TOKEN, BUILT_C_P = ' ';
569 M] END;
570 M] IF NEXT_CHAR NOT = ' ' THEN WRITE(6) '***TOKEN=' || NEXT_CHAR;
571 M] ELSE WRITE(6) '***BLANKS=' || BLANK_COUNT + 1;
572 M] END;
573 M] IF NEXT_CHAR = ' ; ' THEN DO;
574 M] WRITE(6) SKIP(?), '***CVEPT**' || BUILT_UP;
575 M] WRITE(6) '***MARKER**' || BUILT, SKIP(?);
576 M] BUILT, BUILT_UP = ' ';
577 M] END;
578 M] IF NEXT_CHAR = '?' THEN GO TO DISASTER;
579 M] END: /* DO FOREVER */
580 M] DISASTER:
```
<table>
<thead>
<tr>
<th>STMT</th>
<th>SOURCE</th>
<th>LINE CURRENT SCOPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>580 M1 CALL PRINT_SUMMARY;</td>
<td></td>
<td>782 HALINHAL</td>
</tr>
<tr>
<td>581 M1 WRITE(6) ' THIS TEST IS NOW COMPLETE. ';</td>
<td></td>
<td>783 HALINHAL</td>
</tr>
<tr>
<td>582 M1 CLOSE HALINHAL;</td>
<td></td>
<td>784</td>
</tr>
<tr>
<td>G1 A B C D E F</td>
<td></td>
<td>785</td>
</tr>
<tr>
<td></td>
<td></td>
<td>786</td>
</tr>
<tr>
<td>LCF NAME</td>
<td>TYPE</td>
<td>CLASS</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>-------</td>
</tr>
<tr>
<td>FALINAL</td>
<td>PROG</td>
<td>LABEL</td>
</tr>
<tr>
<td>R</td>
<td>0</td>
<td>REPL</td>
</tr>
<tr>
<td>FALSE</td>
<td>0</td>
<td>REPL</td>
</tr>
<tr>
<td>TRUE</td>
<td>0</td>
<td>REPL</td>
</tr>
<tr>
<td>IF</td>
<td>0</td>
<td>REPL</td>
</tr>
<tr>
<td>ELSE</td>
<td>0</td>
<td>REPL</td>
</tr>
<tr>
<td>CUT</td>
<td>0</td>
<td>REPL</td>
</tr>
<tr>
<td>FOREVER</td>
<td>0</td>
<td>REPL</td>
</tr>
<tr>
<td>EXP</td>
<td>CHAR</td>
<td>VAR</td>
</tr>
<tr>
<td>BUILT</td>
<td>CHAR</td>
<td>VAR</td>
</tr>
<tr>
<td>BUILT_TOKEN</td>
<td>CHAR</td>
<td>VAR</td>
</tr>
<tr>
<td>PLANK_FLAG</td>
<td>INT</td>
<td>VAR</td>
</tr>
<tr>
<td>RECNO_CNT</td>
<td>INT</td>
<td>VAR</td>
</tr>
<tr>
<td>MACRO_PUNT</td>
<td>INT</td>
<td>VAR</td>
</tr>
<tr>
<td>MACRO_LIMT</td>
<td>INT</td>
<td>VAR</td>
</tr>
<tr>
<td>CLR_LVL</td>
<td>INT</td>
<td>VAR</td>
</tr>
<tr>
<td>MACRO_LVL</td>
<td>INT</td>
<td>VAR</td>
</tr>
<tr>
<td>MCRO_LVL2</td>
<td>INT</td>
<td>VAR</td>
</tr>
<tr>
<td>MAX_SEVERI</td>
<td>INT</td>
<td>VAR</td>
</tr>
<tr>
<td>STATEMENT_SEVERITY</td>
<td>INT</td>
<td>VAR</td>
</tr>
<tr>
<td>SAVE_SEVERI</td>
<td>INT</td>
<td>ARRAY</td>
</tr>
<tr>
<td>SAVE_LINE</td>
<td>INT</td>
<td>ARRAY</td>
</tr>
<tr>
<td>PREVICS_ERROR</td>
<td>INT</td>
<td>VAR</td>
</tr>
<tr>
<td>I</td>
<td>INT</td>
<td>VAR</td>
</tr>
<tr>
<td>T_DISABLE</td>
<td>INT</td>
<td>VAR</td>
</tr>
<tr>
<td>TIME</td>
<td>INT</td>
<td>VAR</td>
</tr>
<tr>
<td>TIME2</td>
<td>INT</td>
<td>VAR</td>
</tr>
<tr>
<td>MACRO_STREAM</td>
<td>CHAR</td>
<td>VAR</td>
</tr>
<tr>
<td>MACRO_FOUN</td>
<td>CHAR</td>
<td>VAR</td>
</tr>
<tr>
<td>CARD_COUN</td>
<td>CHAR</td>
<td>VAR</td>
</tr>
<tr>
<td>CANCELER</td>
<td>CHAR</td>
<td>VAR</td>
</tr>
<tr>
<td>CVR_PUNCH</td>
<td>CHAR</td>
<td>VAR</td>
</tr>
<tr>
<td>PLANK_CNTL</td>
<td>INT</td>
<td>VAR</td>
</tr>
<tr>
<td>CONTROL</td>
<td>BIT</td>
<td>ARRAY</td>
</tr>
<tr>
<td>TOGGLES</td>
<td>CHAR</td>
<td>VAR</td>
</tr>
<tr>
<td>CARD_TYPE</td>
<td>INT</td>
<td>FUNC</td>
</tr>
<tr>
<td>SELECT</td>
<td>CHAR</td>
<td>VAR</td>
</tr>
<tr>
<td>CHAR_INDEX</td>
<td>INT</td>
<td>FUNC</td>
</tr>
<tr>
<td>STRING</td>
<td>CHAR</td>
<td>VAR</td>
</tr>
<tr>
<td>PATTERN</td>
<td>CHAR</td>
<td>VAR</td>
</tr>
<tr>
<td>I</td>
<td>INT</td>
<td>VAR</td>
</tr>
<tr>
<td>J</td>
<td>INT</td>
<td>VAR</td>
</tr>
<tr>
<td>K</td>
<td>INT</td>
<td>VAR</td>
</tr>
<tr>
<td>PARG</td>
<td>CHAR</td>
<td>FUNC</td>
</tr>
<tr>
<td>STRING</td>
<td>CHAR</td>
<td>VAR</td>
</tr>
<tr>
<td>WIDTH</td>
<td>INT</td>
<td>VAR</td>
</tr>
<tr>
<td>TEMP_STRING</td>
<td>CHAR</td>
<td>VAR</td>
</tr>
<tr>
<td>I_FORMAT</td>
<td>CHAR</td>
<td>FUNC</td>
</tr>
<tr>
<td>LCC</td>
<td>NAME</td>
<td>TYPE</td>
</tr>
<tr>
<td>-----</td>
<td>---------------</td>
<td>--------</td>
</tr>
<tr>
<td>53</td>
<td>NUMBER</td>
<td>INT</td>
</tr>
<tr>
<td>54</td>
<td>WIDTH</td>
<td>INT</td>
</tr>
<tr>
<td>55</td>
<td>STRING</td>
<td>CHAR</td>
</tr>
<tr>
<td>56</td>
<td>L</td>
<td>INT</td>
</tr>
<tr>
<td>57</td>
<td>ERRORS</td>
<td>PROG</td>
</tr>
<tr>
<td>58</td>
<td>MESSAGE</td>
<td>CHAR</td>
</tr>
<tr>
<td>59</td>
<td>SEVERITY</td>
<td>INT</td>
</tr>
<tr>
<td>60</td>
<td>MSG</td>
<td>CHAR</td>
</tr>
<tr>
<td>61</td>
<td>DISASTER</td>
<td>T-ST.</td>
</tr>
<tr>
<td>62</td>
<td>PRINT_SUMMARY</td>
<td>PROG</td>
</tr>
<tr>
<td>63</td>
<td>STREAM</td>
<td>PROG</td>
</tr>
<tr>
<td>64</td>
<td>F_LINE</td>
<td>CHAR</td>
</tr>
<tr>
<td>65</td>
<td>E_LINE</td>
<td>CHAR</td>
</tr>
<tr>
<td>66</td>
<td>S_LINE</td>
<td>CHAR</td>
</tr>
<tr>
<td>67</td>
<td>SAVE_CARD</td>
<td>CHAR</td>
</tr>
<tr>
<td>68</td>
<td>CURREN_CARD</td>
<td>CHAR</td>
</tr>
<tr>
<td>69</td>
<td>BLANKS</td>
<td>CHAR</td>
</tr>
<tr>
<td>70</td>
<td>E_LINE.ERR</td>
<td>CHAR</td>
</tr>
<tr>
<td>71</td>
<td>CRER.FR</td>
<td>CHAR</td>
</tr>
<tr>
<td>72</td>
<td>PREV.CARD</td>
<td>CHAR</td>
</tr>
<tr>
<td>73</td>
<td>SAVE.NEXT.CHAR</td>
<td>CHAR</td>
</tr>
<tr>
<td>74</td>
<td>SAVE.OVER.PUNCH</td>
<td>CHAR</td>
</tr>
<tr>
<td>75</td>
<td>LAST.E_END</td>
<td>INT</td>
</tr>
<tr>
<td>76</td>
<td>E.PLANKS</td>
<td>INT</td>
</tr>
<tr>
<td>77</td>
<td>M.PLANKS</td>
<td>INT</td>
</tr>
<tr>
<td>78</td>
<td>N.PLANKS</td>
<td>INT</td>
</tr>
<tr>
<td>79</td>
<td>F.TEXT_LIMIT</td>
<td>INT</td>
</tr>
<tr>
<td>80</td>
<td>LAST.E.COUNT</td>
<td>INT</td>
</tr>
<tr>
<td>81</td>
<td>S.COUNT</td>
<td>INT</td>
</tr>
<tr>
<td>82</td>
<td>LAST.S.COUNT</td>
<td>INT</td>
</tr>
<tr>
<td>83</td>
<td>INDX</td>
<td>INT</td>
</tr>
<tr>
<td>84</td>
<td>EP</td>
<td>INT</td>
</tr>
<tr>
<td>85</td>
<td>PRINT</td>
<td>INT</td>
</tr>
<tr>
<td>86</td>
<td>ARROW</td>
<td>INT</td>
</tr>
<tr>
<td>87</td>
<td>I1</td>
<td>INT</td>
</tr>
<tr>
<td>88</td>
<td>F.IND</td>
<td>INT</td>
</tr>
<tr>
<td>89</td>
<td>S.IND</td>
<td>INT</td>
</tr>
<tr>
<td>90</td>
<td>E.INDICATOR</td>
<td>INT</td>
</tr>
<tr>
<td>91</td>
<td>S.INDICATOR</td>
<td>INT</td>
</tr>
<tr>
<td>92</td>
<td>RETURNING.P</td>
<td>INT</td>
</tr>
<tr>
<td>93</td>
<td>RETURNING.M</td>
<td>INT</td>
</tr>
<tr>
<td>94</td>
<td>END.GROUP</td>
<td>INT</td>
</tr>
<tr>
<td>95</td>
<td>F.COUNT</td>
<td>INT</td>
</tr>
<tr>
<td>96</td>
<td>ARGU.FLAG</td>
<td>INT</td>
</tr>
<tr>
<td>97</td>
<td>FIRST.CALL_TO_STREAM</td>
<td>CHAR</td>
</tr>
<tr>
<td>98</td>
<td>TYPE.CHAR</td>
<td>CHAR</td>
</tr>
<tr>
<td>99</td>
<td>E_STACK</td>
<td>CHAR</td>
</tr>
<tr>
<td>100</td>
<td>S_STACK</td>
<td>CHAR</td>
</tr>
<tr>
<td>101</td>
<td>INPUT_PAD</td>
<td>CHAR</td>
</tr>
<tr>
<td>102</td>
<td>STMT.LABEL</td>
<td>INT</td>
</tr>
<tr>
<td>103</td>
<td>START_START</td>
<td>CHAR</td>
</tr>
<tr>
<td>104</td>
<td>END_STACK</td>
<td>CHAR</td>
</tr>
<tr>
<td>105</td>
<td>COUNT.COUNT</td>
<td>INT</td>
</tr>
<tr>
<td>106</td>
<td>P.MSG</td>
<td>CHAR</td>
</tr>
<tr>
<td>107</td>
<td>P.RERU</td>
<td>CHAR</td>
</tr>
<tr>
<td>108</td>
<td>P.START</td>
<td>CHAR</td>
</tr>
<tr>
<td>LCC NAME</td>
<td>TYPE</td>
<td>CLASS</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>PROCESS_COMMENT</td>
<td>PRNC</td>
<td>LABEL</td>
</tr>
<tr>
<td>K</td>
<td>CHAR</td>
<td>VAR</td>
</tr>
<tr>
<td>J</td>
<td>INT</td>
<td>VAR</td>
</tr>
<tr>
<td>COMPLEMENT</td>
<td>STMT</td>
<td>LABEL</td>
</tr>
<tr>
<td>STACK_RETURN_CHAR</td>
<td>PROC</td>
<td>LABEL</td>
</tr>
<tr>
<td>NUMBER</td>
<td>INT</td>
<td>VAR</td>
</tr>
<tr>
<td>CHAIR</td>
<td>CHAR</td>
<td>VAR</td>
</tr>
<tr>
<td>T</td>
<td>INT</td>
<td>VAR</td>
</tr>
<tr>
<td>READ_CASE</td>
<td>PROC</td>
<td>LABEL</td>
</tr>
<tr>
<td>END_CE_INPUT</td>
<td>INT</td>
<td>VAR</td>
</tr>
<tr>
<td>PRIOR_CK</td>
<td>INT</td>
<td>FUNC</td>
</tr>
<tr>
<td>TYPE</td>
<td>CHAR</td>
<td>VAR</td>
</tr>
<tr>
<td>COMMENT</td>
<td>INT</td>
<td>VAR</td>
</tr>
<tr>
<td>RETURN-collapse</td>
<td>PROC</td>
<td>LABEL</td>
</tr>
<tr>
<td>SCAN_CASE</td>
<td>INT</td>
<td>VAR</td>
</tr>
<tr>
<td>TYPE</td>
<td>CHAR</td>
<td>VAR</td>
</tr>
<tr>
<td>LIA</td>
<td>INT</td>
<td>ARRAY</td>
</tr>
<tr>
<td>CONTINUE</td>
<td>STMT</td>
<td>LABEL</td>
</tr>
<tr>
<td>ERRORS</td>
<td>INT</td>
<td>VAR</td>
</tr>
<tr>
<td>Compilation</td>
<td>PROC</td>
<td>LABEL</td>
</tr>
<tr>
<td>GET_Group</td>
<td>STMT</td>
<td>LABEL</td>
</tr>
<tr>
<td>READ_IT</td>
<td>STMT</td>
<td>LABEL</td>
</tr>
<tr>
<td>READ_CASE</td>
<td>INT</td>
<td>VAR</td>
</tr>
<tr>
<td>PRCASES</td>
<td>INT</td>
<td>VAR</td>
</tr>
<tr>
<td>FOUND_GROUP</td>
<td>STMT</td>
<td>LABEL</td>
</tr>
<tr>
<td>CCMP</td>
<td>INT</td>
<td>LABEL</td>
</tr>
<tr>
<td>PROCESS_COMMENT</td>
<td>INT</td>
<td>ARRAY</td>
</tr>
<tr>
<td>CHOP</td>
<td>PROC</td>
<td>LABEL</td>
</tr>
<tr>
<td>GET_Group</td>
<td>INT</td>
<td>LABEL</td>
</tr>
<tr>
<td>STACK</td>
<td>STMT</td>
<td>LABEL</td>
</tr>
<tr>
<td>TYPE</td>
<td>INT</td>
<td>VAR</td>
</tr>
<tr>
<td>INDICATOR</td>
<td>CHAR</td>
<td>VAR</td>
</tr>
<tr>
<td>LIN</td>
<td>INT</td>
<td>ARRAY</td>
</tr>
<tr>
<td>STACK</td>
<td>CHAR</td>
<td>VAR</td>
</tr>
<tr>
<td>LIN</td>
<td>INT</td>
<td>ARRAY</td>
</tr>
<tr>
<td>STACK</td>
<td>CHAR</td>
<td>VAR</td>
</tr>
<tr>
<td>PP</td>
<td>INT</td>
<td>VAR</td>
</tr>
<tr>
<td>MULTIPLE</td>
<td>STMT</td>
<td>LABEL</td>
</tr>
<tr>
<td>ERRORS</td>
<td>INT</td>
<td>LABEL</td>
</tr>
<tr>
<td>BUILD_XSCRIPTS</td>
<td>PROC</td>
<td>LABEL</td>
</tr>
<tr>
<td>CHECK_M</td>
<td>STMT</td>
<td>LABEL</td>
</tr>
<tr>
<td>PROCESS</td>
<td>STMT</td>
<td>LABEL</td>
</tr>
<tr>
<td>STACK</td>
<td>INT</td>
<td>LABEL</td>
</tr>
<tr>
<td>CHOP</td>
<td>INT</td>
<td>LABEL</td>
</tr>
<tr>
<td>AGAIN</td>
<td>STMT</td>
<td>LABEL</td>
</tr>
</tbody>
</table>

PAGE 31
<table>
<thead>
<tr>
<th>LOC</th>
<th>NAME</th>
<th>TYPE</th>
<th>CLASS</th>
<th>LENGTH</th>
<th>PRECISION</th>
<th>NEST</th>
<th>OUTER_LEVEL</th>
<th>FLAGS</th>
<th>SYT_PTR</th>
</tr>
</thead>
<tbody>
<tr>
<td>165</td>
<td>ERRORS</td>
<td>I-CL</td>
<td>LABEL</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>3040</td>
<td>57</td>
</tr>
<tr>
<td>166</td>
<td>SAVE_BLANK_COUNT</td>
<td>TNT</td>
<td>VAR</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>8218</td>
<td>0</td>
</tr>
<tr>
<td>167</td>
<td>STACK_CHECK</td>
<td>STMT</td>
<td>LABEL</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>9040</td>
<td>0</td>
</tr>
<tr>
<td>168</td>
<td>BEGINNING</td>
<td>STMT</td>
<td>LABEL</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>9040</td>
<td>0</td>
</tr>
<tr>
<td>169</td>
<td>FOUND_CHAR</td>
<td>STMT</td>
<td>LABEL</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>9040</td>
<td>0</td>
</tr>
<tr>
<td>170</td>
<td>BLANK</td>
<td>STMT</td>
<td>LABEL</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>9040</td>
<td>0</td>
</tr>
<tr>
<td>171</td>
<td>EXPERIMENT</td>
<td>STMT</td>
<td>LABEL</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>9040</td>
<td>0</td>
</tr>
<tr>
<td>172</td>
<td>SLAS</td>
<td>STMT</td>
<td>LABEL</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>9040</td>
<td>0</td>
</tr>
<tr>
<td>173</td>
<td>MAIN_PROGRAM</td>
<td>STMT</td>
<td>LABEL</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>9040</td>
<td>0</td>
</tr>
<tr>
<td>174</td>
<td>MAIN_LOOP</td>
<td>STMT</td>
<td>LABEL</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>9040</td>
<td>0</td>
</tr>
<tr>
<td>LCC</td>
<td>STATEMENT NUMBER</td>
<td>FLAG</td>
<td>KEY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>-----------------</td>
<td>------</td>
<td>-----</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>00007</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The table lists the LCC (Library of Congress Call Number) and the statement number with the corresponding flags and keys.
MACRO TEXT LISTING:

LCC TEXT

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

STACKING DECISIONS = 13862
CALLS TO SCAN = 896
CALLS TO IDENTIFY = 896
NUMBER OF PREDICTIONS = 9727
MAX STACK SIZE = 25
MAX IND. STACK SIZE = 32
END IND. STACK SIZE = 17
MAX EXT_ARRAY INDEX = 2
XEFF LIST LENGTH = 970
STATEMENT COUNT = 893
MAX CUTUP_LIST_INDEX = 2
MAX NESTING DEPTH = 3
FREE STRING AREA = 55355

766 CARDS WERE PROCESSED.
NC ERRORS WERE DETECTED DURING PHASE 1.

TOTAL FLAPSEC TIME IN COMPILER 0:0:20.52.
FLAPSEC SET UP TIME 0:0:0.04.
ACTUAL FLAPSEC COMPILING TIME 0:0:14.33.
FLAPSEC CLEAN-UP TIME AT END 0:0:1.35.
PROCESSING RATE: 2463 CARES PER MINUTE.
BEGIN TEST CF HAL IN HAL

DECLARE M1$(3,3), M2$(8,8);
DECLARE II ARRAY(5) INTEGER INITIAL(3,2,5,4,1);

DECLARE M1$(3,3), M2$(8,8);
DECLARE II ARRAY(5) INTEGER INITIAL(3,2,5,4,1);
DECLARE II ARRAY(5) INTEGER INITIAL(3,2,5,4,1):

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

DECLARE II ARRAY(5) SCALAR, S1, S2:
FUNCTION DECLARATIONS
11 // FUNCTION DECLARATIONS
12 E
13 M1 PRO1: FUNCTION (P) MATRIX (*,*);
14 S1
**BLANKS=47
**TOKEN=P RCP1
**TOKEN:
**BLANKS=1
**TOKEN=FUNCTION
**BLANKS=2
**TOKEN=
**TOKEN=P MARKER=*
**TOKEN=*
**TOKEN=
**TOKEN=
**TOKEN=
**TOKEN=
**TOKEN=
**TOKEN:
**TOKEN:
**TOKEN:
**TOKEN:
**TOKEN:
**TOKEN:

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

15 E
16 M1 RETURN P

**BLANKS=45
**TOKEN=RETURN
**TOKEN:
**TOKEN=P MARKER=*
**TOKEN=* 
**TOKEN=* 
**TOKEN=
**TOKEN=
**TOKEN=
**TOKEN=*
**TOKEN:

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

17 M1 CLCSE RCP1:
**BLANKS=*
**TOKEN=CLCSE
**ELANKS=1**
**TCKEN=PP01**
**TCKEM=;**

***MAIN*** close-pro

19 M1  **PRO: FUNCTION (E) MATRIX (**,**);**

   **ELANKS=67**
   **TCKEN=PRO**
   **TCKEM=**
   **TCKEN=FUNCTION**
   **ELANKS=1**
   **TCKEM=**
   **TCKEN=**
   **TCKEM=**
**ELANKS=1**
**TCKEN=MATRIX**
**ELANKS=7**
**TCKEM=**
**TCKEN=**
**TCKEM=**

***MAIN*** close-pro:

16 M1  **DECLARE MATRIX (**,**);**

   **B**

17 M1  **DECLARE MATRIX (**,**);**

   **B**

20 M1  **RETURN PRO1(B,...);**

   **2 TO 4,1 TC 3**

17 **BLANKS=56**
RETURN **BLANKS=1
***TCKFA=PROC
***TCKEN=1
***TCKEN=U
***TCKEN=S
***TCKEN=C
***TCKEN=2
**BLANKS=1
***TCKEN=T0
**BLANKS=1
***TCKEN=T0
**BLANKS=1
***TCKEN=S
***TCKEN=0
***TCKEN=1
***TCKEN=2

**RETURN** RETURN PROI(AS(2 TO 4, 1 TO 3));

22 M1. CLOSE PRC;
**BLANKS=0
***TCKFA=CLOSE
**BLANKS=1
***TCKEN=PROC
***TCKEN=1

**RETURN** CLOSE PRC;
MAIN PROGRAM
22 HI MAIN PROGRAM
24 HI DC FOR I=1 TO 5;

***BLANKS=47
***TCKFN=N
***BLANKS=1
***TOKEN=FOR
***BLANKS=1
***TCKFN=I
***TCKFN=1
***TCKFN=1
***TCKFN=7
***TCKFN=1
***TOKFN=TO
***TOKFN=1
***TCKFN=5
***TCKFN=;

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

25 HI DC FOR J=1 TO 5;

***BLANKS=63
***TOKFN=DC
***BLANKS=1
***TCKFN=FOR
***BLANKS=1
***TOKFN=J
***TCKFN=*
***TCKFN=*
***TCKFN=*
***TENK=*
***TENK=*
***TENK=*
***TENK=*
***TENK=*

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

26 EI J
27 EL
23 M1 M2 = S1 :
25 S1 I, J T1
3C S1 I

***BLANKS=65
***TCKFN*Y
***TCKFN=*
***TCKFN=
***TCKFN=
***TENK=
***TENK=
***TENK=
***TENK=
***TENK=

***BLANKS=1
46\#1\**\*
**BLANKS=53
**TCFN=ACLSF
**BLANKS=1
**TCFN=TPARM
**TOKEN=1
**CVER***
**MAIN***
**CLOSE TPARM;

47\#1
**BLANKS=67
**TCFN=9
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.
CARDS PROCESSING RATE: 4.42652E+02 CARDS PER MINUTE,
CLOCK TIME IS 4361443
THIS TEST IS NOW COMPLETE.

2
M TPAM: PROGRAM;
M DECLARE M1 3,3;
M M2, S, E;
M DECLARE I1 ARRAY(5) INTEGER INITIAL(3,2,5,4,1);
M DECLARE ARRAY(5) SCALAR, S1, S2;
H FUNCTION DECLARATIONS
H PRO1: FUNCTION (P ) MATRIX (*,*)
H RETURN P !
H CLOSE PRO1;
H PRO: FUNCTION (B) MATRIX (*,*)
H DECLARE MATRIX (*,*) , B;
H RETURN PRO1(B);
H CLOSE PRO;
H MAIN PROGRAM
H DO FOR I=1 TO 5;
H DO FOR J=1 TO 5;
E 11$1
E M2 = S1 ;
S 1, J 11
E END;
E THE NEXT GROUP HAS AN OVERLAPPING E LINE
E 11$1
E S2 = S1 ; /* COMMENTS ARE BLANKS */
E 1 1
S 2
E THIS SHOULD DEMONSTRATE CARDS OUT OF ORDER
E MS2 = S2 ;
E 1 1
E AN S LINE OVERLAP ABOVE
E END;
E *
E M1 = PRO(M2 /* IGNORED */);
S 4 AT 3,4 AT 5
M CLOSE TPAM;