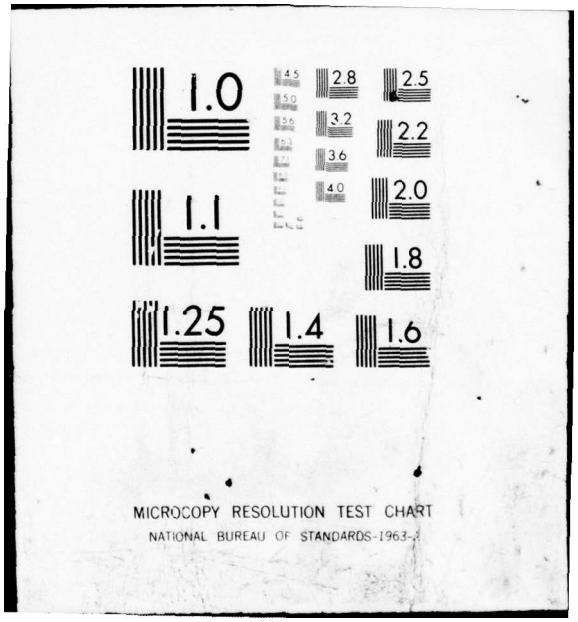
AD-A072 381 NORTHWESTERN UNIV EVANSTON IL DEPT OF ELECTRICAL ENG--ETC F/G 9/2
DYNAMIC MONITORING FOR LINEAR LIST DATA STRUCTURES.(U)
JUN 79 S S YAU, J L RAMEY F30602-76-C-0397
UNCLASSIFIED RADC -TR-79-128 NL

1 OF 2
AD A072381




1.0
1.1
1.25
1.4
1.6
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.8
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

DYNAMIC

AD A072381

RADC-TR-79-128
Interim Report
June 1979

DYNAMIC MONITORING FOR LINEAR LIST DATA STRUCTURES

Northwestern University

Stephen S. Yau
John L. Ramey

LEVEL II

DDC
RECEIVED
AUG 7 1979
C

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

DDC FILE COPY

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

79 08 06 029

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-79-128 has been reviewed and is approved for publication.

APPROVED: Rocco F. Iuorno

ROCCO F. IUORNO
Project Engineer

APPROVED: Wendall C. Bauman

WENDALL C. BAUMAN, Colonel, USAF
Chief, Information Sciences Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (ISIS) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: RADC-TR-79-128	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): DYNAMIC MONITORING FOR LINEAR LIST DATA STRUCTURES	5. TYPE OF REPORT & PERIOD COVERED: Interim Technical Report. 1 Aug 76 — 31 Jul 79
	6. PERFORMING ORG. REPORT NUMBER: N/A
7. AUTHOR(s): Stephen S. Yau, John L. Ramey	8. CONTRACT OR GRANT NUMBER(s): F30602-76-C-0397
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Northwestern University, Dept. of Electrical Engineering & ~~Computer Science~~, Evanston IL 60201	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62728F, 55810278
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (ISIS), Griffiss AFB NY 13441	12. REPORT DATE: June 1979
	13. NUMBER OF PAGES: 111
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Rocco Iuorno (ISIS)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Dynamic monitoring, assertion techniques, software maintenance, linear list data structures, arrays, record traversal scheme, implementation considerations, preprocessor.

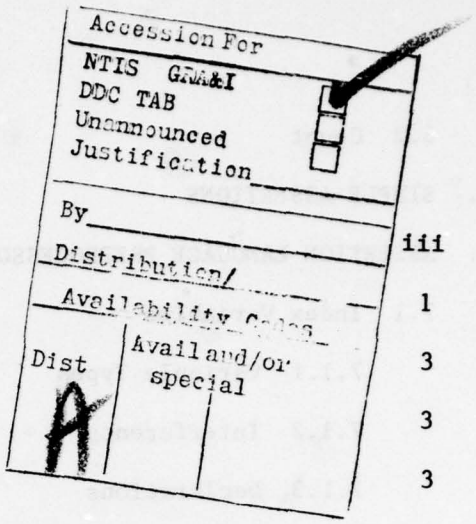
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

In this report, the contributions that dynamic monitoring can make in the area of software maintenance are discussed. New techniques that enable monitoring of most array-implemented linear list data structures are presented. The main advantage of these assertion techniques is the ability to construct a loop around a group of simple assertions. With this construct, the programmer can explicitly define the record traversal scheme for a linear list data structure implemented with either sequential or linked-list record allocation.

DD FORM 1 JAN 73 1473

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)



CONTENTS

LIST OF FIGURES iii

1. INTRODUCTION 1

2. ASSERTION CONCEPTS 3

2.1 The Assumption/Decision Model 3

2.2 A Software System Life Cycle 3

2.3 An Assertion Language 4

3. ARRAY DATA STRUCTURES 5

4. ASSERTION TECHNIQUES FOR LINEAR LIST DATA STRUCTURES 6

4.1 Record Oriented Monitoring 6

4.2 Linear Lists 6

4.3 Node Traversal Schemes 6

4.3.1 Sequential Allocation 7

4.3.2 Linked Allocation 8

4.4 Empty Data Structures 9

4.5 Examples 10

4.5.1 Stack - sequential allocation 10

4.5.2 Queue - sequential allocation 10

4.5.3 Queue - linked list allocation 11

4.5.4 Circular lists 12

4.5.5 Doubly-Linked list 12

4.5.6 Sequential variable - length records 13

4.6 Multi-dimensional Linear Structures 13

5. OPTIONAL INTEGRITY CHECKS 14

5.1 Bounds 14

5.2 Count 15

6. SIMPLE ASSERTIONS 15

7. ASSERTION LANGUAGE PREPROCESSOR 15

7.1 Index Variables 16

7.1.1 Variable Types 16

7.1.2 Interference 16

7.1.3 Declarations 17

7.1.4 JOVIAL conclusions 18

7.2 Integrity checks 18

7.2.1 COUNT 18

7.2.2 BOUNDS 19

7.2.3 A Complete Example 19

7.3 Command Stack 20

7.4 JOVIAL Automated Verification System (JAVS) 21

7.4.1 JAVS Directives 21

7.4.2 Directive Code Modifications 21

7.4.3 Assertion Violation Output 22

8. CONCLUSIONS 23

9. REFERENCES 25

APPENDIX I: JAVS Assertion Directives 27

APPENDIX II: Error Diagnostic Messages 28

APPENDIX III: JAVS Modification Documentation 29

APPENDIX IV: JAVS Modification Listings 50

APPENDIX V: Example JAVS run 85

LIST OF FIGURES

Page

Figure 1. Queue Implemented in Sequential Allocation..... 11

2. Array Used as Buffer for Variable-Length Records.... 13

1. INTRODUCTION

One of the major problems in maintaining large-scale software is the difficulty of understanding the dynamic behavior of the software system. In order to gain a good understanding of the dynamic behavior, it is necessary to have effective dynamic monitoring techniques which are applicable to large-scale software. One approach is to build self-metric software in which the software system has built into itself the mechanisms for measuring its own behavior. However, the large quantity of data collected must be analyzed. The software can also have these analysis mechanisms built into it giving the software system the capability of measuring and evaluating its own behavior. When this evaluation is used to determine whether the software is behaving properly, the software is called self-checking software.

A software system can be characterized as a collection of inter-related assumptions and decisions among modules. Each module makes assumptions about its operating environment. Based on these assumptions the module designer makes a number of decisions about the properties of the module. These decisions completely characterize the external behavior of the module. They become the properties expected of the module by other modules that interact with it. Thus, a module uses a collection of assumptions and makes one or more decisions which are in turn used as assumptions by other modules throughout the system.

During software modification, new errors are frequently introduced into the system in the following way. A modification is made to program module X which changes the definition of a decision D made in that module. Thinking that he has made a complete and successful modification, the programmer begins testing the new version. If everything seems to function correctly, the new version is considered correct and put into production usage. However, the problem arises because some module Y contains code which uses decision D as an assumption. D has been changed and module Y may no longer function properly. If the system is large and the modification relatively small, the testing was probably not very extensive with respect to complete testing. The necessary conditions for failure might not have been generated, or the malfunction might have been so subtle that it went undetected at the time it occurred. In either case, the new version of the software system contains errors due to the software modification.

The maintenance programmer works under a serious handicap. He must work without much knowledge of the execution behavior characteristics of the various control and data variables that comprise the software system. When looking at a particular block of code, it is difficult to determine what assumptions are used and what decisions are made and passed on to other parts of the system. If a decision is to be changed, it is difficult to identify all the other blocks of code where that decision is used as an assumption.

Significant progress is currently being made in research aimed at being able to trace all the possible decision/assumption paths eminating from any given block of code. However, there is a need for a practical tool to help ease the burden on the maintenance programmer. Dynamic monitoring techniques based on the use of assertions can provide that valuable tool. Well

formulated assertions will serve as explicit statements of many of the execution behavior characteristics relevant to each block of code in the system. Assertions will also detect assumption violations introduced during incorrect modifications to the software system.

The use of assertions was first introduced by Satterthwaite [1], who used the assertions to check the truth of logical expressions in a program. If an evaluation was false, the program was forced to abnormally terminate and a post-mortem analysis was initiated. Stucki [3] has presented a more sophisticated dynamic monitoring scheme which has been added to PET (Program Evaluation and Tester) [2]. Along with the assertions on logical expressions used by Satterthwaite, Stucki included assertions on legal and illegal values and ranges of variables. These techniques were still designed primarily for use on simple variables. Stucki [4] later described extensions to this language that makes it usable with simple array structures. But, even these extensions have several serious shortcomings with respect to monitoring array implementations of linear list data structures. The most serious problem is that the programmer can only specify the range for the index variable, but not the exact sequence of values the index should take. Even this range specification cannot be used to its full potential since it is a static definition and does not represent the true dynamic behavior of the data structure. Finally, if several assertions are to be made on the same array, the index specifications must laboriously be included with each assertion statement. Recently, Chow [5] has presented a method for using past as well as present program status in the assertions of logical expressions. While the ideas presented are interesting, they are not dealing with the monitoring of array structures.

In this report we will discuss dynamic monitoring with assertions and how it can be of benefit in the software maintenance process. We will also discuss the use of an assertion language compared to other possible alternatives. We will then present an advancement to the current assertion specification techniques that enables dynamic monitoring of most array-implemented linear list data structures. The main feature of our new techniques is the ability to construct a loop around a group of simple assertions. With this loop construct, the programmer can explicitly define the record traversal scheme for a linear list data structure implemented with either sequential or linked-list record allocation. Within the assertion loop, Stucki's simple logical expression and value assertions can then be used on the individual array items defined by the loop index.

We will give a number of examples to illustrate how these techniques may be used in typical JOVIAL tables and arrays. The example will include various linear list data structures, and both sequential and linked-list allocation schemes. We will also discuss the implementation of an assertion language preprocessor for a JOVIAL system. As a demonstration, we have modified JAVS [6-8] so that it recognizes and translates our proposed assertion constructs. We will also discuss how these techniques are applicable to other high level languages.

2. ASSERTION CONCEPTS

2.1 The Assumption/Decision Model

An ideal situation would be to have assertions written to verify all of the assumptions made by each of the modules in this system. This would provide complete documentation of expected run-time behavior as well as a means for each module to check if the outside environment is really as it is claimed to be. Some assumptions, such as legal values and ranges of input variables, readily lend themselves to specification as assertions. However, more complex assumptions concerning large data structures or relationships between various variables may be too complicated to be specified and thoroughly verified in a run-time situation.

In the same manner, assertions can be included for decisions made within each module. This will provide documentation as well as run-time verification of correct implementation of each decision. If a decision is incorrectly implemented, it may be detected and identified at virtually the location of the error. This will save lots of debugging time. As with the assumptions, not all of the decisions can be easily formulated into simple assertions.

More research will be required to determine what percentage of all the assumptions and decisions can be easily specified as dynamic assertions.

2.2 A Software System Life Cycle

Assertions should be a part of the entire life cycle of a large scale software system. They should be considered in the initial design and coding phases and will be of useful service as long as the system is being maintained.

During the initial design phase of the system, each module designer must identify the assumptions used and the decisions made by his module. These assumptions and decisions should then be included as assertions in the initial version of the coded module. From that point on, the assertions will serve as important documentation of the expected execution-time behavior characteristics of that module. The assertions included in each module will be useful when testing the individual modules and then the system as a whole. They will help detect and identify some errors earlier than otherwise might be the case. They may also detect errors that would otherwise go unnoticed. Thus, initial testing can be quicker and more thorough.

When the system has been thoroughly tested and is ready for use in the field, it would be recompiled without the assertions. The assertions would remain as comments but they would not be converted into executable code. Thus, no space or time overhead is added to the production system.

When it is necessary to perform a modification on the system, the assertions can again play a very helpful role. Before attempting any changes, the maintenance programmer must thoroughly examine the relevant modules and become familiar with both their static and dynamic characteristics. He must be aware of what assumptions are used in the system. Well formulated assertions provide the needed documentation.

But errors can still be made. To minimize the amount of damage done, it is important that these errors be detected as early as possible. By again compiling the assertions in the related parts of the system, the programmer has a powerful tool for detecting any abnormal system behavior, something that is otherwise very difficult to do. Errors that are introduced into the system during program modification usually occur through the decision/assumption relationship. A decision in one module may get changed making the assumption used consistent with the new decision from the first module, it may not function correctly. Thus, errors are introduced when the second module is left unchanged using an old assumption. The assumption assertion in the problem's second module would assist by noticing the assumption violation and identifying it to the maintenance programmer.

2.3 An Assertion Language

Thus far, our discussion of assertions has not been concerned with how they might be specified. For years, many good programmers have included input parameter checks in their routines reflecting a "I don't trust anyone" philosophy. These first assertions were coded along with the regular program and frequently remained in the finished product. This has an adverse effect on the space and time efficiency of the completed software system. We need assertions that can always remain in the source code, but yet don't always have to be compiled or executed.

The simplest method to achieve this desired control is to continue writing the assertions in the high-level language of the system, but include a special flag with each statement to indicate that it is part of an assertion. Then, prior to compilation the specially flagged statements could be converted to comments or left as executable code by a simple preprocessor.

A more sophisticated approach involves the use of a specialized assertion language. There are several reasons we feel that such an approach is desirable. First, an assertion language provides a simple syntax that is tailored to the checks that must be expressed. The assertions can be written in a much more concise manner than would otherwise be possible. For instance, the value assertion as defined by Stucki [2] is much simpler than the equivalent statements in any high-level language. The programmer is also spared from worrying with the output that must be generated if an assertion violation occurs. He knows that for each assertion, the preprocessor will generate statements to output a standard-form violation message. Also, this concise, tailored format means that each assertion statement is self-documenting; it is easily readable and readily stands apart from the remainder of the code.

Second, an assertion language overcomes a major problem associated with assertions written in the language of the software system. When assertions are written in the same language as the total software system, it is difficult to maintain the desired separation between the assertions and the remainder of the code. If the statements that make up an assertion are not flagged as assertions promptly when they are written, they will likely never be so flagged. This is because without an indepth study of the code, it is usually quite difficult to determine if a statement is part of an assertion or the real program. More than likely, the final program will have

both flagged and unflagged assertion statements. Then, when the assertions are to be disabled, these unflagged assertions may cause either compilation errors or additional time-and-space overhead, or both. An assertion language avoids this problem; the assertions always remain distinctly separate from the rest of the code.

Another benefit is the ease with which the programmer can control the enabling of the assertions. In the simple case, a preprocessor can either convert all the assertions to the appropriate high-level code or leave them as comments in that language. In a more powerful approach, the assertion language could include a complex hierarchy of control giving the programmer both compilation and execution-time control over the enabling of individual assertions [4].

All these reasons for using an assertion language basically come down to convenience and reliability. They are the same general reasons that the software is being written in a high-level language instead of in an assembly language. The programmer's time is valuable and we can expect a more reliable product.

However, there are some drawbacks to the use of an assertion language. First, a simple preprocessor must be written that will translate the assertion statements into the desired high level code. Since an assertion language can be kept simple, this does not appear to be a serious problem. Secondly, the assertion language is less flexible than the high level language that would otherwise be used. However, using Stucki's simple assertions and the advancement we propose in the remainder of this report, we feel that most desired assertions can be easily stated.

Overall, we feel that a powerful assertion language provides significant benefits in the specification of dynamic assertions.

3. ARRAY DATA STRUCTURES

The array is a commonly used data structure, one that is explicitly defined in almost all high-level languages. Yet current dynamic monitoring techniques are not usable with any but the simplest array structures.

An array is defined as a collection of homogeneous items that may all be referenced by the same name. Sometimes these items are merely elementary items of the language. Yet frequently they are themselves collections of other items that are non-homogeneous and/or logically related to each other. These latter collections are called records though each language seems to have its own unique terminology. Arrays of records are explicitly provided for in JOVIAL [9], PASCAL [10], PL/I [11], ALGOL 68 [12] and COBOL [13] although in JOVIAL these structures, called tables, are only one-dimensional.

FORTRAN [14] is a prominent high-level language that has no explicit record structure. There are two common techniques used to overcome this deficiency. The first method is to create a separate array for each individual item of the record. The other way is to implement m-dimensional records of an n-dimensional space using an n + m dimensional array. Either way, there is still the logical concept of an array of records.

In general, we can consider every array to be an array of records. These records may be elementary items, explicitly defined record structures or implicitly defined FORTRAN records. Thus, when specifying dynamic assertions for arrays, we want to treat the arrays as record-oriented data structures.

4. ASSERTION TECHNIQUES FOR LINEAR-LIST DATA STRUCTURES

4.1 Record-Oriented Monitoring

As noted above, most of the data structures explicitly defined in current programming languages are record-oriented organizations. We feel that dynamic assertion techniques for these structures ought to represent this record philosophy.

There are two things that the programmer must define in order to specify dynamic assertions for a record-oriented structure.

1) The group of simple assertions that is to be applied to each record in the structure.
2) A systematic means of accessing each record in the structure.

The simple assertions should be those already defined in the existing language for use with simple variables. A language construct must be defined that will have the assertions together as a group that will be applied to each record. The systematic access of records for the general class of all data structures is not a simple task. Therefore, for the rest of the report, we will limit our discussion to linear lists, a class of data structures in common use in today's programming practices.

4.2 Linear Lists

A linear list is a set of $n \geq \emptyset$ nodes $x[1], x[2],\ldots,x[n]$ whose structural properties essentially involve only the linear (one-dimensional) relative positions of the nodes [12]. Three special linear lists: stacks, queues and deques; allow deletions, additions and accesses to values only at the first or last node of the structure. Typically, these are implemented using a sequential storage allocation scheme. The more general class of linear lists allows additions, deletions and accesses to any node in the list. Typically, some sort of linked record allocation is used for these more general linear-list structures. Methods for specifying systematic traversal schemes for both types of allocation will be presented in the following section.

4.3 Node-Traversal Schemes

To specify the traversal of any data structure, three characteristics of the traversal must be defined:

1) the first node to look at
2) the method for determining the next node
3) the condition that signals the end of the traversal.

As is done within the program itself, an index will be used for speci-

fying the desired elements of the data structure. Our definitions for the first and next nodes will be values for the controlling index. The terminating condition will be any general Boolean expression.

The traversal scheme specification will take the form of a standard high-level program loop. The loop conditions will define the sequence of records to be monitored and within the loop bounds will be grouped the assertions for each record.

4.3.1 Sequential Allocation

If the records are stored in a sequential manner in the data structure, then the range for the index variable is sufficient to define the traversal. A simple loop definition using the range specification takes the following format:

```
LOOP(⟨variable name⟩) (⟨range⟩)
        .
        .
        .
END LOOP
```

For example, consider a JOVIAL table with N records stored in sequential order starting in record Ø. The following would define the assertions loop.

```
LOOP (I) (Ø...N-1)
      .
      .
      .
END LOOP
```

A JOVIAL assertion preprocessor would generate the following JOVIAL code and insert it into the original source code.

```
FOR I = Ø, 1, N-1  $
BEGIN
 .
 .
 .
END
```

This type of assertion can also be applied equally well to software systems written in other languages. For instance, the assertions on a FORTRAN array of N elements would look very similar to those written above for the JOVIAL example.

```
LOOP (I) (1...N)
   .
   .
   .
END LOOP
```

The translation to FORTRAN would then be as follows: where STMT denotes a

unique FORTRAN statement number.

```
          DO STMT I = 1, N
             .
             .
             .
STMT      CONTINUE
```

4.3.2 Linked Allocation

A more complicated situation exists when the records are stored in a linked manner scattered throughout the physical data structure. The initial value for the control index is generally taken from a variable maintained by the main program as a pointer to the first record. The next value for the index usually is taken from a link field within the current record. And the terminating condition is stated as a Boolean expression that when evaluated true, denotes that the end of the chain has been reached. Frequently this condition is that the link field of the current record, and subsequently the next value of the index variable, is an implementation-dependent null pointer. The syntax for this loop definition is:

```
CHAIN (⟨index variable⟩) INIT (⟨arith. expr.⟩)
NEXT (⟨arith. expr.⟩) END (⟨boolean expr.⟩)
                          .
                          .
                          .
END CHAIN
```

For example, consider a JOVIAL table that contains a linked list. Simple item PTR contains the index for the first record in the chain. The table contains item LINK which provides the link between records and is -1 in the last record of the chain. To loop through this linked list, the following directives would be used.

```
CHAIN (IX) INIT (PTR) NEXT (LINK($IX$)) END (IX EQ -1)
                          .
                          .
                          .
END CHAIN
```

The assertion preprocessor would then generate and insert the following JOVIAL source statements.

```
          IX = PTR $
Label.
             .
             .
             .
          IX = LINK($IX$) $
          IF NOT (IX EQ -1) $
          GOTO Label $
```

Label is used to denote a preprocessor-generated label that is unique to this

loop expansion. Again this type of assertion is also applicable to software systems written in other languages. For example, consider how the above example might look in a FORTRAN system. The linked list is stored in a 2-dimensional array, LLIST, with each row being a record of the logical structure. The first element of each row contains the link to the next logical record. Now our assertions would look as follows.

```
CHAIN (IX) INIT (PTR) NEXT (LLIST(IX,1)) END (IX .EQ. Ø)
        .
        .
        .
END CHAIN
```

The preprocessor might convert this to FORTRAN as indicated below, where again STMT is used to denote a uniquely generated statement number.

```
      IX = PTR
STMT  CONTINUE
        .
        .
        .
      IX = LLIST(IX,1)
      IF (.NOT. (IX .EQ. Ø)) GO TO STMT
```

4.4 Empty Data Structures

One important consideration that has not yet been accounted for is the possibility that the data structures might be empty. If that is the case, then the loop and the assertions defined within the loop should not be executed. To handle this situation, an optional EMPTY (⟨boolean expr.⟩) clause may be appended to either of the previously-defined loop-specification statements as follows:

```
LOOP (⟨variable⟩) (⟨range⟩) [EMPTY (⟨boolean expr.⟩)]

                    or

CHAIN (⟨variable⟩) INIT (...) NEXT (...) END (...)
          [EMPTY (⟨boolean expr.⟩)]
```

A true evaluation of the Boolean expression indicates that the data structure is empty and the assertions should be skipped. The JOVIAL preprocessor would group the whole assertion loop as a compound statement and precede it with an appropriate IF statement as shown below:

```
IF NOT (⟨boolean expression⟩) $
BEGIN
   Loop as defined previously
END
```

In the two examples discussed in Sections 4.3.1 and 4.3.2, the EMPTY Boolean expressions might be "N EQ Ø" and "PTR EQ -1".

4.5 Examples

In this section, we will show some examples detailing how the assertion loops could be used for typical linear-list structures and allocation schemes. At this point we are only concerned with the record-traversal scheme and thus will not try to show any specific assertions.

4.5.1 Stack - sequential allocation

Consider a JOVIAL table, STACK, that contains a stack of records. The stack is filled from the bottom (index = Ø) up and NENT(STACK) contains the number of entries in the stack. The loop and assertions, including an empty stack check, could be written as follows:

```
LOOP (I) (Ø...NENT(STACK)-1) EMPTY (NENT(STACK) EQ Ø)
   Assertions
END LOOP
```

The preprocessor would generate the following JOVIAL code:

```
IF NOT (NENT(STACK) EQ Ø) $
BEGIN
  FOR I = Ø, 1, NENT(STACK)-1 $
  BEGIN
     Assertions
  END
END
```

4.5.2. Queue - sequential allocation

To implement a first-come-first-serve queue, a large JOVIAL table, QUEUE, might be used. Records are added to one end of the sequential list and removed from the other. The queue moves progressively around the fixed length table. HEAD is used here to point to the oldest record in the queue, the one that will be serviced next. TAIL points to the next record to be filled at the end of the queue. When the queue is empty, HEAD = TAIL. QUELEN is the length of the table.

Because of its cyclic nature, the queue is always in one of three possible states as shown in Figure 1. Because of the difference of the relative values of HEAD and TAIL, these three cases cannot all be monitored using the same assertion-loop definition. Instead, we will use a JOVIAL IF-EITHER statement to separate case 3 from cases 1 and 2 as follows:

```
IFEITH HEAD LQ TAIL  $
   BEGIN "CASES 1 AND 2"
      LOOP (I) (HEAD...TAIL-1) EMPTY (HEAD EQ TAIL)
         Assertions
      END LOOP
   END "CASES 1 AND 2"
ORIF HEAD GT TAIL  $
   BEGIN "CASE 3"
```

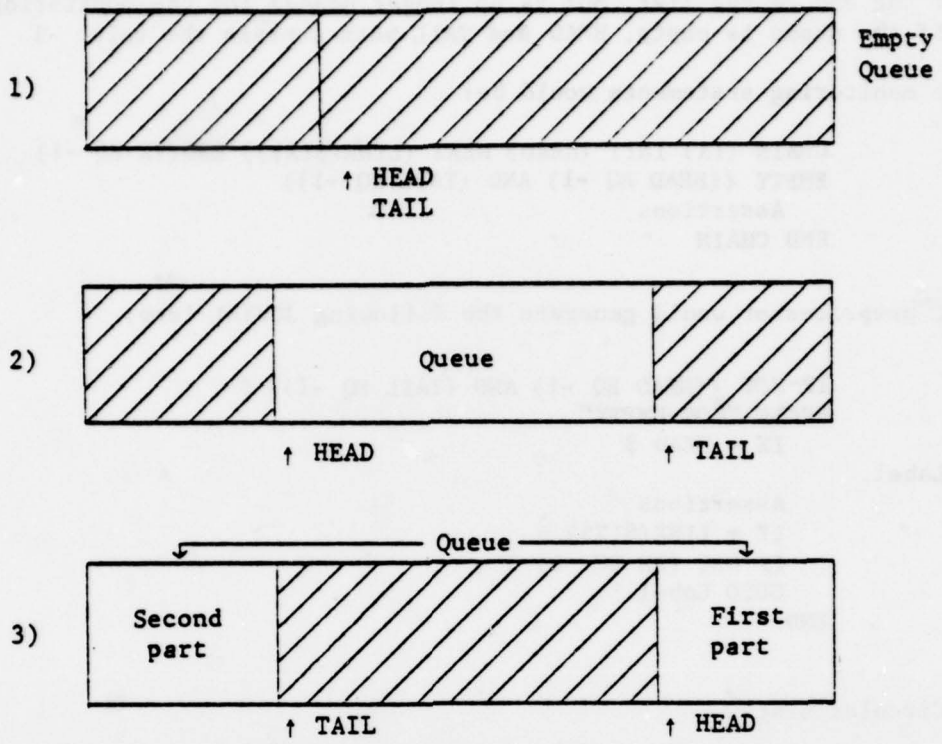


Fig. 1. Queue Implemented in Sequential Allocation
(The shaded areas represent empty records).

```
            LOOP (I) (HEAD...QUELEN -1)
               Assertions
            END LOOP
            LOOP (I) (Ø...TAIL -1)
               Assertions
            END LOOP
         END "CASE 3"
      END "IF EITHER"
```

For each of the assertion-loop definitions, code similar to that for the stack in Section 4.5.1 would be generated by the JOVIAL preprocessor.

4.5.3 Queue - linked list allocation

Consider again the queue in the previous section, only this time being implemented as a linked-list of records. Table item LINK is added and provides the proper linkage between the records. In the last record, the LINK

field contains the value -1. HEAD still points at the oldest record, what is now the first record in the list. TAIL points to the most recent record added at the end of the list, but is no longer needed for the monitoring process. If the queue is empty, HEAD and TAIL both contain the value -1.

The monitoring statements would be:

```
CHAIN (IX) INIT (HEAD) NEXT (LINK($IX$)) END(IX EQ -1)
EMPTY ((HEAD EQ -1) AND (TAIL EQ -1))
   Assertions
END CHAIN
```

A JOVIAL preprocessor would generate the following JOVIAL loop:

```
            IF NOT ((HEAD EQ -1) AND (TAIL EQ -1)) $
            BEGIN "NON-EMPTY"
               IX = HEAD $
      Label.
               Assertions
               IX = LINK($IX$) $
               IF NOT (IX EQ -1) $
               GOTO Label $
            END
```

4.5.4 Circular Lists

If the queue in Section 4.5.3. was to be implemented now as a circular list, the LINK value for the last record would have to point back to the first record in the list. This would cause a change to the terminating condition in the loop definition as shown below:

```
CHAIN (IX) INIT (HEAD) NEXT (LINK($IX$)) END (IX EQ HEAD)
EMPTY (HEAD EQ -1)
```

It is the circular list data structure that dictates the use of a DO-UNTIL loop in the JOVIAL expansion of the chain traversal definition. If a DO-WHILE loop was used, the terminating condition would be met immediately after the initial index value assignment. In this example, we can see that the initial assignment HEAD→IX would lead to a true evaluation of IX EQ HEAD and thus an exit from the DO-WHILE loop.

4.5.5. Doubly-Linked List

To traverse the records of a doubly-linked list, either one of the chains may be followed. Thus the doubly-linked list structure simplifies into either a linked-list or a circular list as far as the dynamic monitoring of the record values is concerned.

A totally complete set of assertions on a doubly-linked list should also verify that the two chains are consistent with each other. This would involve increased complexity in the loop specification language as well as in the source code generated. At the present time however, the need does not

seem to justify the added overhead and complexity that would be required.

4.5.6. Sequential Variable-Length Records

Frequently variable-length records are stored sequentially in a linear array, usually with the first word of each record containing the length of the record as shown in Figure 2. Typically the array acts as a buffer

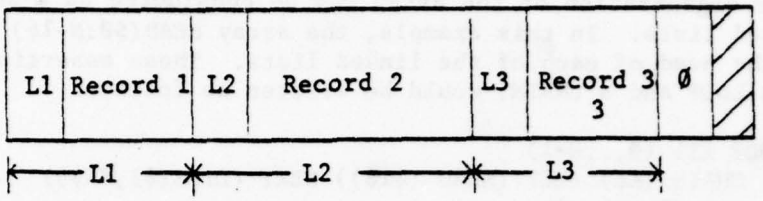


Fig. 2. Array Used as Buffer for Variable-Length Records
(The shaded areas represent empty space.)

for these records between main memory and some sort of secondary storage media.

Our chain-traversal definition is flexible enough to be of easy use with this sort of record organization. In the following example, BUFFER is a linear array. As indicated above, the first word of each record contains the length of that record. Following the last record is a Ø to indicate the end of the list. The loop definition would be written as follows.

```
CHAIN (IX) INIT (Ø) NEXT (IX + BUFFER($IX$)) END (BUFFER($IX$) EQ Ø)
EMPTY (BUFFER($Ø$) EQ Ø)
   Assertions
END CHAIN
```

4.6 Multi-dimensional Linear Structures

Until now we have only demonstrated our assertion approach with 1-dimensional linear lists. However, our techniques are also applicable to the general class of multi-dimensional linear structures. This is accomplished by the use of nested loops, one for each index of the array. Then inside the inner-most loop, the desired assertions can be written for the individual records.

The following examples illustrate the use of nested loops for multi-dimensional arrays. In each example, we will write assertions for an N by M array of records. This array can be viewed as a linear list of N linear lists of M records. Thus, what we must do is to specify the looping constructs for each of the two lists.

The simplest and most common view of our N x M array would consider both lists to be of a sequential nature. We would write the loops for such an organization as follows:

```
LOOP (I) (Ø...N-1)
   LOOP (K) (Ø...M-1)
      Assertions
   END LOOP
END LOOP
```

Another organization of the array may be considered as a sequential list of linked lists. In this example, the array HEAD($Ø:N-1$) contains the pointer to the head of each of the linked lists. These assertion loops, now using both a LOOP and a CHAIN, would be written as follows:

```
LOOP (I) (Ø...N-1)
   CHAIN (KK) INIT (HEAD ($I$)) NEXT (LINK($I,KK$))
   END (KK EQ -1)
      Assertions
   END CHAIN
END LOOP
```

We may also view the array as a linked list of linked lists. The array LISTLINK ($Ø:N-1$) provides the linkages in the list of lists while HEAD ($Ø:N-1$) again contains the head of list pointers for the lists of records. TOP points to the first list of records.

```
CHAIN (II) INIT (TOP) NEXT (LISTLINK($II$)) END (II EQ -1)
   CHAIN (KK) INIT (HEAD($II$)) NEXT (LINK($II,KK$))
   END (KK EQ -1)
      Assertions
   END CHAIN
END CHAIN
```

As demonstrated in these examples, our assertion techniques are applicable to any combination of allocation schemes that might be used in an array structure. They are simple and powerful enough to make assertion specification for multi-dimensional linear structures a simple task.

5. OPTIONAL INTEGRITY CHECKS

Thus far, we have discussed only integrity checks on the data values contained in the various linear data structures. However, with a linked-list structure, it is also important that we are able to check the integrity of certain aspects of its logical structure. Toward this end, we now define the optional BOUNDS and COUNT clauses to be appended to the chain-traversal loop definition as follows:

```
CHAIN (⟨variable⟩) INIT (...) NEXT (...) END (...)
[EMPTY (...)] [BOUNDS (⟨range⟩)] [COUNT (⟨range⟩)]
```

5.1 Bounds

With this clause, the programmer can make an assertion about the legal values that the index variable may take. If the index is out of range, it usually indicates a serious error in the logical data structure. For this

reason, after a BOUNDS assertion violation, the assertion loop should be immediately exited. This feature should provide detection of erroneous link field values and data structures that have simply overgrown their assigned memory space.

5.2 Count

This clause specifies an assertion about the legal number of records in the linked list. For instance, if the program maintains its own record count, this clause could monitor the correctness of that count. Another easy use might be to verify that the data structure is indeed empty or non-empty at a particular time. As with the BOUNDS clause, an error message is printed when an assertion violation is detected.

6. SIMPLE ASSERTIONS

In this section we will examine the actual assertions needed for simple and record-oriented items. Stucki [4] has defined one general and five specialized local assertion formats. Three of these seem important within the framework of the current discussion.

1) ASSERT (⟨boolean expression⟩)
2) ASSERT VALUE (⟨variable name⟩) (⟨list of legal values and/or ranges⟩)
3) ASSERT VALUE (⟨variable name⟩) NOT (⟨list of illegal values and/or ranges⟩)

The last two assertions could obviously be written using one or more of the general Boolean expression assertions. However, for clarity and ease of use, we feel that the value-type assertions should also be explicitly defined in the assertion language.

A brief example should adequately illustrate these assertions. Let us again consider the stack from Section 4.5.1, but this time we are interested in the actual assertions that we skipped over previously.

As defined before, I is the index variable, and we will use it to identify the record being examined. If A, B, C, D are items defined in the table, we could see the following assertions on these items.

```
ASSERT (A($I$) * B($I$) GT Ø)
ASSERT VALUE (C($I$)) (1, 5, 10,...15)
ASSERT VALUE (D($I$)) NOT (2, 4)
```

It should be noted here that currently JAVS [7] provides only the general assertion on a Boolean expression.

7. ASSERTION LANGUAGE PREPROCESSOR

In this section we will examine some of the important issues in the implementation of a preprocessor for our new assertion constructs in a JOVIAL system. Some of these implementation issues can effect the way the user may interface with the system. Our purpose here is to present the various alter-

natives that are available. We will also look at our implementation in JAVS of the preprocessor for an assertion language with these new constructs.

7.1 Index Variables

Both of our new looping constructs define the values for an index variable that is used to identify the desired array elements. Some of the more difficult implementation issues revolve around these indices, and particularly, the way they must be specified in JOVIAL. One of our main objectives must be to supply the user with the simplest possible tool. Yet, this objective will certainly have to be balanced against the realities of the preprocessor implementation.

7.1.1 Variable Types

One issue that is peculiar to JOVIAL systems and was covered over previously is the matter of index variable types. As the JOVIAL expansions for the LOOP and CHAIN constructs were previously defined, each requires the use of a different type of variable for the index. Since the LOOP command translates into a FOR statement, a one letter FOR variable must be used. On the other hand, the CHAIN command requires the use of a two to six character variable. The programmer must either remember which command requires which type of variable or be aware of the code to which the commands are translated. Neither alternative is very attractive.

An alternate translation for the LOOP construct may provide the best solution. Instead of using a FOR loop, the loop controls could be written explicitly with other JOVIAL structures. The translation for a LOOP command into JOVIAL would then be as follows:

```
LOOP (IX) (Ø...N-1)
.
.
.
END LOOP
```

translates to

```
    IX = Ø $
LABEL.
    .
    .
    .
    IX = IX + 1 $
    IF NOT (IX GR N-1)$
    GOTO LABEL $
```

The normal two- to six-character variable name could then be used for the index variable in both loop constructs.

7.1.2 Interference

It is important that the assertion statements not interfere with the

normal execution of the software system. The assertions must have access to all of the main program's variables, yet they must not be allowed to alter the values of those variables. The loop index variables are the only potential problem variables since they are the only ones to which new values are assigned. Ideally the preprocessor should protect the main program from interference by these index variables. Otherwise,the programmer will have to be responsible for avoiding harmful side effects from his assertions.

If the target language for the preprocessor is a block-structured language, such as ALGOL 60 [16], ALGOL 68 [12], PASCAL [10] or PL/I [11], the preprocessor can easily protect the main program from interference by the assertions. This is possible because of the way these languages define the scope of the variables. Any variable x is considered to be local to the block in which it is defined. Variable x can then be accessed as a global variable in blocks nested within the defining block unless the nested block defines its own variable x. In that case, the original x is unreachable yet left unharmed until the lower block is exited. The preprocessor needs only to translate each block of assertions into a block structure of the target language. Within this block the index variable could be declared and used as a local variable. Thus,the assertions have access to all the variables from higher level blocks, and yet the index variable does not interfere with any of the other variables already in use.

However, JOVIAL does not provide such facilities. The scope of a variable is either the entire program or routine in which it is defined. Thus, to avoid interference, the index variable must be unique relative to all the other currently live variables. One solution is for the preprocessor to "guarantee" this uniqueness. It could concatenate a short string onto the variable name to create a new unique name. For example, the string might be XXXX. Then, if the programmer used II for the index variable name, the preprocessor would generate the variable name IIXXXX. This assumes that the programmer knows to avoid using variable names ending with the string "XXXX". This solution may increase a lot of complexity in the preprocessor. A reasonable alternative seems to be to require the programmer to choose a unique index name for his assertions. A better approach may be to always use a particular distinctive name, such as ASSTIX, that would never be used elsewhere in the program.

7.1.3 Declarations

In most programming languages, the programmer must declare all of the variables he intends to use. Since the assertion loop indices are not part of the main program, we feel that the programmer should not need to include declarations for them. This means that the preprocessor must insert the necessary declarations.

In the block structured language approach described in the previous section, the declaration responsibility naturally falls on the preprocessor. Declarations are generated and inserted at the beginning of each assertion block.

For a JOVIAL implementation, the situation is not so well defined. If

the preprocessor is creating its own names, then it must also insert the variable declarations. However, if the programmer uses the same index variable name twice and the preprocessor includes two declarations, the compiler will issue a warning message. Certainly this does not represent an ideal solution to the problem. On the other hand, if the programmer is responsible for using unique index variable names, it would seem natural for him to also include the declarations.

7.1.4 JOVIAL conclusions

We feel that both loop constructs should use the same type of index variable as discussed in Section 7.1.1. Regarding variable interference and declarations, a realistic approach is to leave the index variable choice and declaration responsibilities with the programmer. These decisions provide an adequete interface with the programmer while holding down the complexity of the preprocessor.

7.2 Integrity checks

When the COUNT and BOUNDS options were previously introduced, no translations into JOVIAL code were given. The intent of the two clauses was evident without introducing the details of implementation. Here we will discuss those details.

The code for these options must be designed so that it can be easily inserted into the code already specified for a CHAIN loop. This will help keep the option translations a simple process.

7.2.1 COUNT

For the count check we will introduce a specific variable, ASTCNT, that will hold the count of records processed in any particular loop. ASTCNT can be automatically declared at either the beginning of the program or the time of the first count check. The later declaration requires the use of a separate flag to indicate whether or not ASTCNT has already been declared.

For a particular COUNT check, the preprocessor must generate code to initialize, increment and validate ASTCNT. Code must be inserted before, in the middle of, and after the basic assertion loop. For example,

```
CHAIN (IX) INIT (PTR) NEXT (LINK($IX$)) END (IX EQ -1)
COUNT (LOW...HIGH)
:
END CHAIN
```

would be translated to

```
     ASTCNT = Ø $
     IX = PTR $
LABEL.
     ASTCNT = ASTCNT + 1 $
     :
```

```
        IX = LINK($IX$) $
        IF NOT (IX EQ -1) $
        GOTO LABEL $
        IF NOT (LOW LQ ASTCNT LQ HIGH) $                    "COUNT"
        print error message $                               "COUNT"
```

The comment "COUNT" indicates the statements that were inserted to implement the COUNT option.

7.2.2 BOUNDS

The BOUNDS clause presents a slightly different problem. Now, the validity check must be made within the basic assertion loop and an exit taken if a violation is detected. To provide the exit point, the preprocessor must generate another unique variable name that will be placed immediately following the normal loop exit.

For example, the following is an assertion loop definition with a BOUNDS check and the generated JOVIAL code. All statements inserted for the BOUNDS clause are so marked.

```
        CHAIN (IX) INIT (PTR) NEXT (LINK($IX$)) END (IX EQ -1)
        BOUNDS (LOW...HIGH)
        .
        .
        .
        END CHAIN


        IX = PTR $
    LABEL.
        IF NOT (LOW LQ IX LQ HIGH) $                        "BOUNDS"
        BEGIN                                               "BOUNDS"
           print error message $                            "BOUNDS"
           GOTO ELABEL $                                    "BOUNDS"
        END                                                 "BOUNDS"
        .
        .
        .
        IX = LINK($IX$) $
        IF NOT (IX EQ -1) $
        GOTO LABEL $
    ELABEL.                                                 "BOUNDS"
```

7.2.3 A Complete Example

The following is a complete example designed to show how the code from each of the options; EMPTY, COUNT, and BOUNDS fits together. The JOVIAL statements from each of the options are again marked with comments.

```
CHAIN (IX) INIT (PTR) NEXT (LINK($IX$)) END (IX EQ -1) EMPTY (PTR EQ -1)
COUNT (LOCNT...HICNT) BOUNDS (LOBND...HIBND)
.
.
.

END CHAIN


IF NOT (PTR EQ -1) $                                          "EMPTY"
BEGIN
   ASTCNT = Ø $                                               "COUNT"
   IX = PTR $
LABEL.
   IF NOT (LOBND LQ IX LQ HIBND) $                            "BOUNDS"
   BEGIN                                                      "BOUNDS"
      print error message $                                   "BOUNDS"
      GOTO ELABEL$                                            "BOUNDS"
   END                                                        "BOUNDS"
   ASTCNT = ASTCNT + 1 $                                      "COUNT"
   .
   .
   .
   IX = LINK($IX$) $
   IF NOT (IX EQ -1) $
   GOTO LABEL $
ELABEL.                                                       "BOUNDS"
   IF NOT (LOCNT LQ ASTCNT LQ HICNT) $                        "COUNT"
   print error message $                                      "COUNT"
END
```

7.3 Command Stack

In the definition of an assertion loop, all of the loop parameters are specified at the top of the loop. Some of these parameters are used immediately by the preprocessor while others must be saved for usage at the end of the loop. As was discussed previously, assertion loops may be nested several levels deep. This suggests the use of a stack to retain the necessary information for each of the nested loops. Each element of the stack should be a record with fields for the different parameters to be retained. The following are the parameters that must be retained for a JOVIAL implementation as we have thus far described it.

1) Index variable
2) Algebraic expression for the next value of the index variable
3) Boolean condition that indicates exit from the loop
4) Label at the head of the loop
5) Label at the exit from the loop (optional entry-used with BOUNDS clause)
6) Legal range for record count (optional entry-used with COUNT clause)

7.4 JOVIAL Automated Verification System (JAVS)

The current version of JAVS provides the programmer with some assertion statements for use with Boolean expressions. We feel that the inclusion of our new assertion constructs, along with the Stucki-like VALUE assertion, would make JAVS a much more powerful system development and maintenance tool. Thus, we explored the feasibility of modifying JAVS to this means. First, it was decided that the set of directives recognized by JAVS could be easily expanded to suit our needs. And just as importantly, it was determined that JAVS already contained general statement parsing and text-manipulation routines to facilitate implementation of the new directives. None of the existing software would need to be changed, only new code added. The following sections briefly describe the modification work done on JAVS. (More detailed documentation and the modified JAVS source listing can be found in Appendices III and IV).

7.4.1 JAVS Directives

JAVS directives are statements which may be included in a JOVIAL source text but are only meaningful to the JAVS preprocessor. Some of these directives, such as JAVSTEXT, are used for naming and organizing various parts of a large software system. Others, the computational directives, cause JOVIAL code to be generated and inserted into the probe text. It is this subset that we are interested in expanding.

All the JAVS directives are written in the following basic format:

".⟨JAVS-directive⟩⟨any necessary parameters⟩"

with the character strings ". and " serving as directive delimiters. The choice of these symbols as delimiters insures that the directives can be left in the source code and will be interpreted as comments by the JOCIT compiler. (Double quotes (") serve as the JOVIAL comment delimiter.)

For our new assertion constructs, we added these four new JAVS computational directives: CHAIN, ENDCHAIN, LOOP and ENDLOOP. Each directive statement was given the appropriate syntax as defined previously. For instance, the LOOP directive syntax is as follows:

".LOOP (⟨variable name⟩) (⟨range⟩) [EMPTY (⟨boolean expression⟩)]

We also saw a need to expand the simple assertion power of JAVS as outlined in Section 6.0. In that section, we defined the following three formats for simple assertions.

ASSERT (⟨boolean expression⟩)
ASSERT VALUE (⟨variable name⟩) (⟨list of legal values and/or ranges⟩)
ASSERT VALUE (⟨variable name⟩) NOT (⟨list of illegal values and/or ranges⟩)

JAVS already supports an assertion directive of the following form

".ASSERT, ⟨JOVIAL boolean expression⟩".

We decided to retain this syntax for the expression assertion in order that all old JOVIAL software would be upward compatible with the new JAVS

system. Then, for the other two value assertions, we have added a VALUE directive with the following syntax.

".VALUE (⟨variable name⟩) [NOT] (⟨list of values and/or ranges⟩)"

In summary, we made a total of five additions to the set of computational directives that JAVS recognizes. Four of these are to support our new assertion constructs and the fifth is an enhancement to the simple assertion power of JAVS. A complete listing of the assertion directives recognized by the enhanced version of JAVS can be found in Appendix I.

7.4.2 Directive Code Modifications

Our first task was to make the JAVS source text analyzer recognize the new directives that we wished to add. This was a simple job. Procedure LOOK in component JAVS-2 contains a table, L1, of all the JAVS directives. We had to increment the table size by 5 (definition of NJDIR) and add entries for the new directives and their lengths to table L1.

The remainder of the modifications had to be made in component JAVS-5, the module instrumentation component. Procedure PRBDIR is the routine that is called each time a directive is encountered in the instrumentation phase. PRBDIR had to have additions made to it so that when any of the five new directives are encountered, it invokes the correct procedures.

New procedures had to be written for the directives CHAIN, ENDCHAIN, LOOP and VALUE. Since we decided to translate the LOOP directive into code similar to that for CHAIN (no FOR loop), we are able to use the ENDCHAIN procedure for both the ENDCHAIN and ENDLOOP directives. Each of these new routines was written in the style of the older directive routines, relying heavily on the already existing support routines of components JAVS-2, JAVS-5 and JAVS-11.

The compool for the JAVS-5 component also had to be updated. Procedure templates were added for each of the new procedures and for BALPAR, a JAVS-11 routine that had not previously been called from the JAVS-5 component. A common area, ASSERT, was also added to the compool. Its major contents are the stack and stack pointer for processing the assertion loops. It also contains a flag, DCLASSCNT, which indicates whether or not the variable ASSCNT (used for checking record count assertions) has yet been defined.

Like statements written in any other language, it is certainly possible for these assertion statements to be incorrectly written. For instance, the NEXT field might be missing in a CHAIN directive statement. Or, there might be an imbalance between the number of LOOP and ENDLOOP statements. Checks have been included in all the assertion procedures to detect these types of mistakes. In the event of such an error, the JAVS-11 ERROR procedure is invoked with an explanatory message. Appendix II contains a complete list of these error diagnostic messages.

7.4.3 Assertion-Violation Output

JAVS uses a scheme based on the JOCIT compiler's MONITOR primitive for outputting assertion-violation messages. This was done to add another level

of user control to the activation of the assertions. The violation output can be turned off at compile time independent of the output from the JAVS structural instrumentation [17].

To us, this scheme does not seem entirely satisfactory. The first problem is that it ties the assertion facility to the monitoring task. One can never have the assertions enabled without also having the monitors activated. Thus it is not possible to get any occasional assertion violation output without also getting the possibly voluminous monitor output. And to make matters even worse, output from both sources goes to the same output file.

However, the major problem is that under this scheme, the output messages must be fixed at the time of structural instrumentation. It is not possible to output the execution-time values of the pertinent variables along with a violation message. The programmer is virtually left in the dark when it comes to locating the cause of the violation. He is not given even the slightest bit of information from which to start his search.

We feel that a more powerful and flexible output scheme could be used by sacrificing the compile-time control. Instead of using the MONITOR capability, standard JOVIAL output statements could be inserted into the probe text. These statements would output the appropriate explanatory message and any pertinent variable values for each violation.

However, in keeping with our desire not to alter any of the existing JAVS system, we will retain the MONITOR scheme for use with our new assertions.

8. CONCLUSION

In this report we discussed the contributions that dynamic monitoring can make in the area of software maintenance. Assertions provide documentation that can help the maintenance programmer avoid modification errors. The assertions also assist in the detection of any errors that are introduced during modification. These benefits were previously not recognized since the initial work on dynamic monitoring had been done from a software reliability perspective.

The main thrust of the results presented in this report is the expansion of the already existing assertion concepts to include array data structures which have been virtually ignored in all previous work. Initially we developed a record-oriented approach toward array data structures. Based on this approach we proposed an assertion technique that will enable effective monitoring of most linear-list data structures that have been implemented as JAVS tables or arrays. We believe this is a big step in the development of monitoring as an effective software-development tool.

Finally, we discussed some important considerations relative to the implementation of our new assertion concepts for a JOVIAL system. As demonstration, modifications were made to JAVS so that it could serve as the preprocessor for an expanded assertion language containing our new constructs. Appendix V contains the output listing of an example run of this modified

version of JAVS.

More research is needed to study the use and benefits of dynamic monitoring throughout the whole software life cycle. We need to determine how easily assumptions and decisions can be defined and explicitly stated when designing software. Then, how many of these can effective assertions be written for? Assertions appear to be a promising tool for detecting errors made during software modification, but we need to research this area more. We must determine what kind of errors are typically made and how effective this type of dynamic monitoring will be in detecting them. Our belief is that dynamic monitoring will indeed be an effective and powerful tool over the whole software life cycle.

9. REFERENCES

[1] Satterthwaite, E., "Debugging Tools for High Level Languages," Software Practice and Experience, Vol. 2, 1972, pp. 192-217.

[2] Stucki, L. G., "Automatic Generation of Self-Metric Software," Proceedings of 1973 IEEE Symposium on Computer Software Reliability, 1973, pp. 84-100.

[3] Stucki, L. G. and Foshee, G. L., "New Assertion Concepts for Self-Metric Software Validation," Proceedings 1975 International Conference on Reliable Software, 1975, pp. 59-71.

[4] Stucki, L. G., "The Use of Dynamic Assertions to Improve Software Quality," Ph.D. Dissertation, UCLA Graduate School of Engineering, Computer Science Dept., June, 1976.

[5] Chow, T. S., "A Generalized Assertion Language," Second International Conference on Software Engineering, 1976, pp. 392-399.

[6] Cannon, C., Brooks, N. B., and Urban, R. J., JAVS Technical Report - User's Guide (RADC-TR-77-126, Vol I - AD A040103), General Research Corporation, Santa Barbara, California, April 1977.

[7] Cannon, C. and Brooks, N. B., JAVS Technical Report - Reference Manual (RADC-TR-77-126, Vol II - AD A040104)), General Research Corporation, Santa Barbara, California, April 1977.

[8] Cannon, C. and Brooks, N. B., JAVS Technical Report - Methodology Report (RADC-TR-77-126, Vol III - AD A041048), General Research Corporation, Santa Barbara, California, April 1977.

[9] JOCIT Compiler Users Manual, Computer Sciences Corporation under contract F30602-72-C-0467 with Rome Air Development Center.

[10] Jensen, Kathleen and Wirth, Niklaus, PASCAL User Manual and Report, Springer-Verlag, New York, 1974.

[11] Pollack, S. V. and Sterling, T. D., A Guide to PL/I, Holt, Rinehart, and Winston, Inc., New York, 1969.

[12] Van Wijngaarden, A., et.al., Revised Report on the Algorithmic Language Algol 68, Springer-Verlag, Berlin, 1976.

[13] Dock, V. Thomas, COBOL: American National Standard, Reston Publishing Company, Reston, Virginia, 1973.

[14] Pollack, S. V., A Guide to FORTRAN IV, Columbia University Press, New York, 1965.

[15] Knuth, D. E., The Art of Computer Programming, Vol. 1, Fundamental Algorithms, Addison-Wesley Publishing Company, Reading, Mass., 1973.

[16] Rutishauser, Heinz, ed., Description of Algol 60, Springer-Verlag, New York, 1967.

[17] Benson, J. B., Brooks, N. B., Gannon, C. and Urban, R. J., JAVS Computer Program Documentation: Vol. 1, System Design and Implementation Manual (CR-2-722), General Research Corporation, Santa Barbara, California, November, 1976, pg. 4-61.

APPENDIX I

JAVS Assertion Directives

".ASSERT, ⟨boolean expr.⟩"

".VALUE (⟨variable name⟩) [NOT] (⟨list of values and/or ranges⟩)"

".LOOP (⟨variable name⟩) (⟨range⟩) [EMPTY (⟨boolean expr.⟩)]

".ENDLOOP"

".CHAIN (⟨variable name⟩) INIT (⟨arith. expr.⟩) NEXT (⟨arith. expr.⟩) END (⟨boolean expr.⟩) [EMPTY (⟨boolean expr.⟩)] [BOUNDS (⟨range⟩)] [COUNT (⟨range⟩)]"

".ENDCHAIN"

APPENDIX II

Error Diagnostic Messages

This appendix describes the error diagnostic messages that JAVS generates as a result of improperly specified assertion directives. The standard JAVS ERROR procedure is used to output these messages to the JAVS output report. Given here with each error message is a brief explanation of the probable cause of the error and in what procedure it was detected.

ASSERTION STACK OVERFLOW

There is an overflow condition on the assertion command stack. The number of nested LOOP and CHAIN directives exceeds the permissable limit. This limit is defined as MAXCSTK (= 1Ø) in procedures PRBCHAIN and PRBLOOP and in common ASSERT. Error detected in procedure PRBCHAIN or PRBLOOP.

ASSERTION STACK UNDERFLOW

There is an underflow condition on the assertion command stack. For the ENDLOOP or ENDCHAIN directive being processed, there has been no matching LOOP or CHAIN directive. Error detected in procedure PRBENDC.

CHAIN ASSERTION ERROR - NO INIT FIELD

No INIT field specified for a CHAIN assertion directive. Error detected in procedure PRBCHAIN.

CHAIN ASSERTION ERROR - NO NEXT FIELD

No NEXT field specified for a CHAIN assertion directive. Error detected in procedure PRBCHAIN.

CHAIN ASSERTION ERROR - NO END FIELD

No END field specified for a CHAIN assertion directive. Error detected in procedure PRBCHAIN.

APPENDIX III

JAVS Modification Documentation

A. JOVIAL Automated Verification System
Documentation of changes made to JAVS to implement an expanded assertion specification language.

B. Work done at Northwestern University
by John L. Ramey
(312) 492-5248

Sponsored by Rome Air Development Center
Under Contract No. F30602-76-C-0397
Project Engineer - Mr. R. Iuorno

C. Abstract
This work is a modification to JAVS to enhance its dynamic assertion capabilities. More computational directives were added to the set that JAVS recognizes. These new directives give JAVS more assertion power both for simple data items and most linear list data structures. New routines were added to translate these assertions into the proper JOVIAL code during the instrumentation phase. All the source code was written in JOVIAL and tested on a HONEYWELL 6180.

D. Computer definition
The hardware requirements are the same as for the unmodified JAVS.

E. System Description
The modifications were run successfully under the H6180/GCOS Version H.1 operating system. No new special executive software is required.

F. Program Description
The general philosophy used when making the modifications was to use the same basic approaches to the organization and coding as were done in the original version of JAVS. All new routines were written using the same style as the older ones and previously written support routines were used whenever possible. Only enhancements were made so that all previously written programs and JAVS directives still work under the modified version. As was done with the previous directives, they are all recognized in the initial basic analysis of phase 2. Then during the instrumentation phase, routine PRBDIR distinguishes between the various possible directives calling the correct routine to handle each.

JPROBE - modified procedure in JAVS-5
An addition was made to this procedure so that it initialized both the assertion command stack pointer (CSTKPTR) and the variable indicating whether or not the count variable had yet been declared (DCLASSCNT).

LOOK - modified procedure in JAVS-2

Additions were made to the directive table in this procedure so that the five additional directives would be recognized during the basic analysis phase.

PRBAST - modified procedure in JAVS-5
This procedure was modified to make use of the new procedure PRBDCL instead of doing the same tasks internally.

PRBCHAIN - new procedure in JAVS-5
This procedure handles the CHAIN assertion directive. For each directive, it generates the required JOVIAL code for the top of the assertion loop and saves the necessary information on the top of the assertion command stack for use when the matching ENDCHAIN directive is encountered.

PRBCHKL - new procedure in JAVS-5
This procedure is used to keep from overflowing the TXTBUF buffer when building statements. Whenever there is a possibility that a new addition might not fit in the buffer, PRBCHKL is called before ADDTXT. If it will not fit, the contents of the buffer are written out as a statement and the buffer is cleared.

PRBDCL - new procedure in JAVS-5
This procedure makes sure that the MONITOR function is turned on for outputting violations from the assertions.

PRBDIR - modified procedure in JAVS-5
Additional entries were made to the IFEITHER statement in this routine so that when the new directives were encountered during the instrumentation phase, the appropriate procedures would be called.

PRBENDC - new procedure in JAVS-5
This procedure handles both the ENDCHAIN and the ENDLOOP directives. It gets the information from the record on top of the assertion command stack and generates the JOVIAL statements necessary for the termination of the assertion loop.

PRBLBL - new procedure in JAVS-5
This procedure generates a six-digit label based on the current statement number. Called by PRBCHAIN and PRBLOOP.

PRBLOOP - new procedure in JAVS-5
This procedure handles the LOOP assertion directive. For each directive, it generates the required JOVIAL code for the top of the assertion loop and saves the necessary information on the top of the assertion command stack for use when the matching ENDLOOP directive is encountered.

PRBVALUE - new procedure in JAVS-5
This procedure handles the VALUE assertions. It extracts the variable name and values from the assertion directive and generates

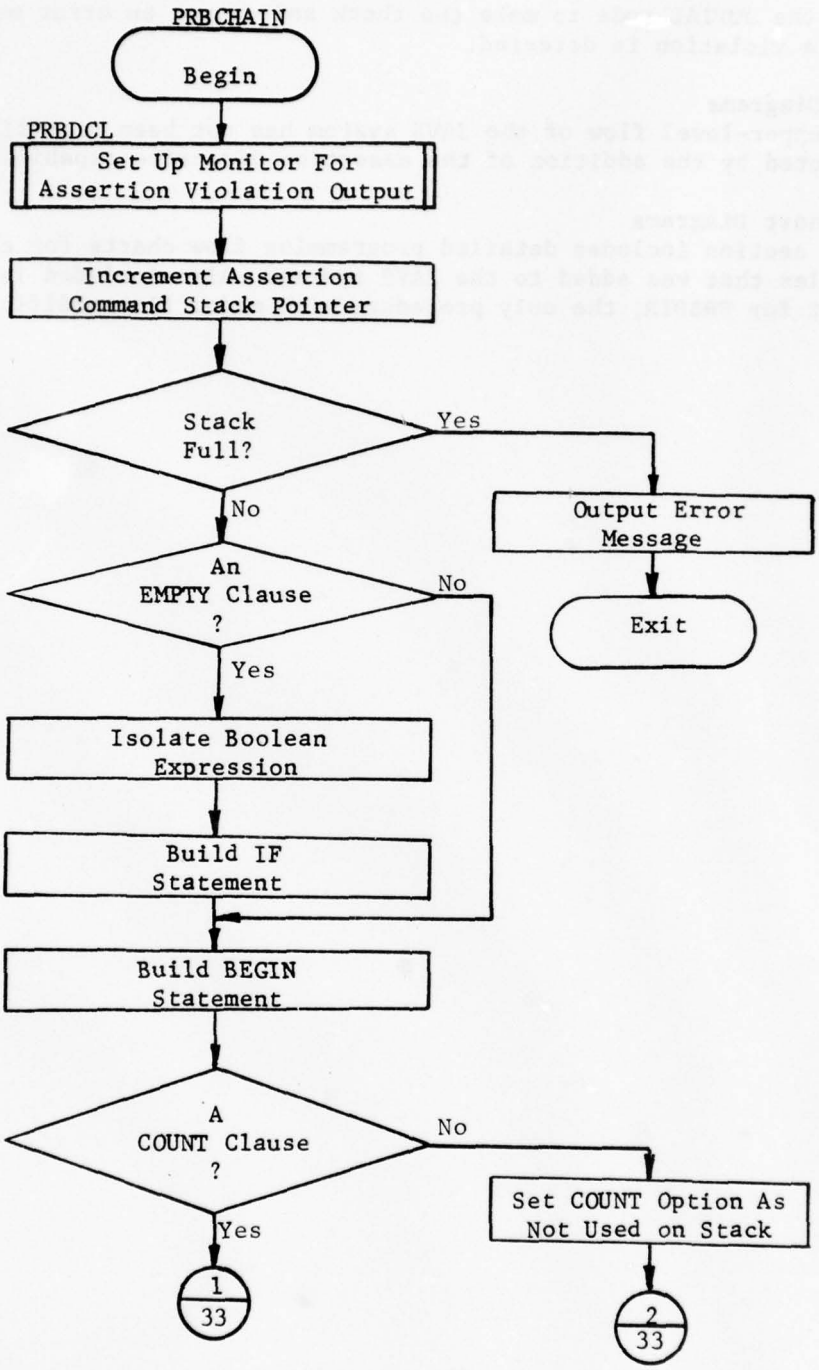
the JOVIAL code to make the check and output an error message if a violation is detected.

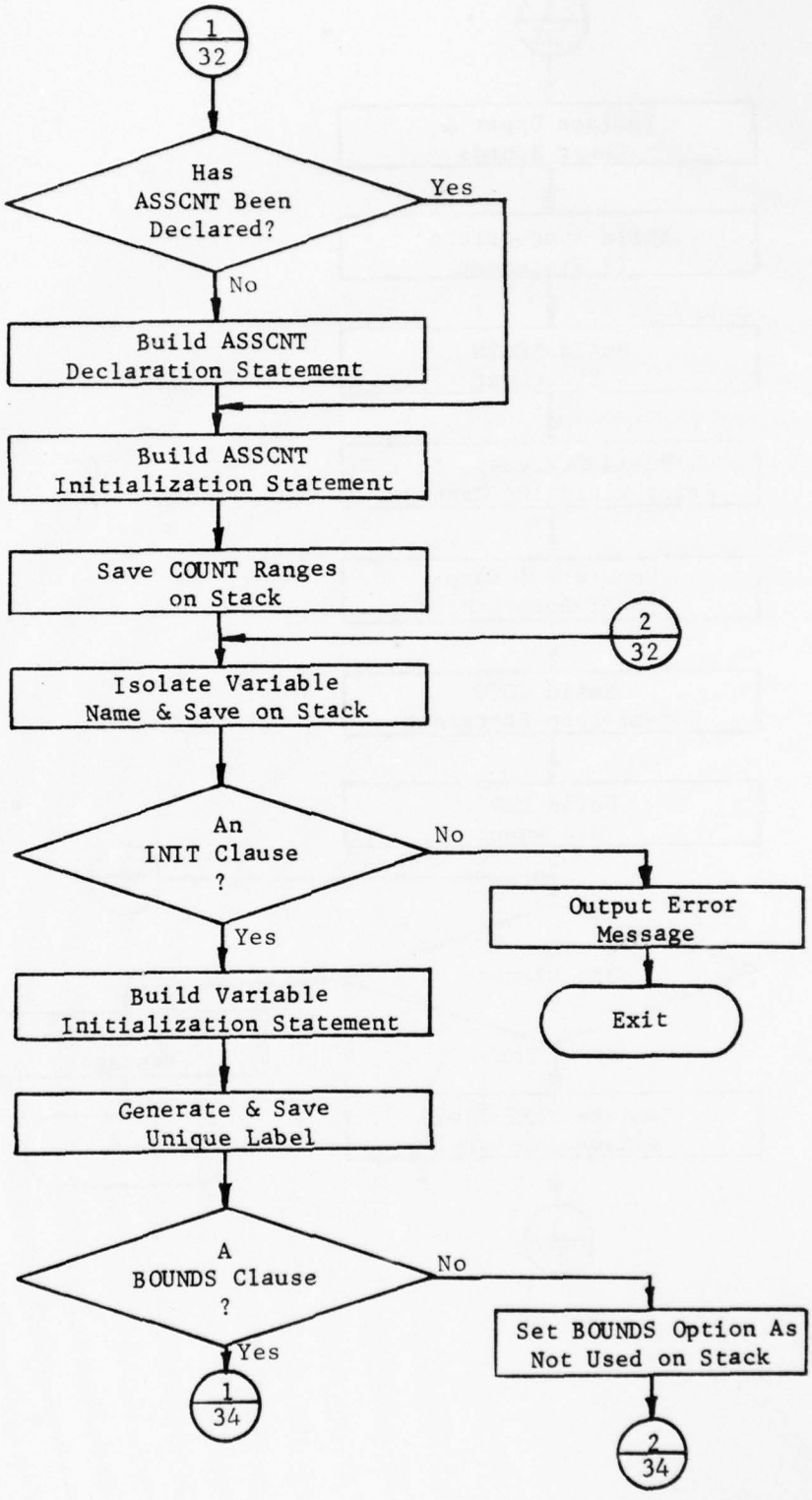
G. Logic Diagrams

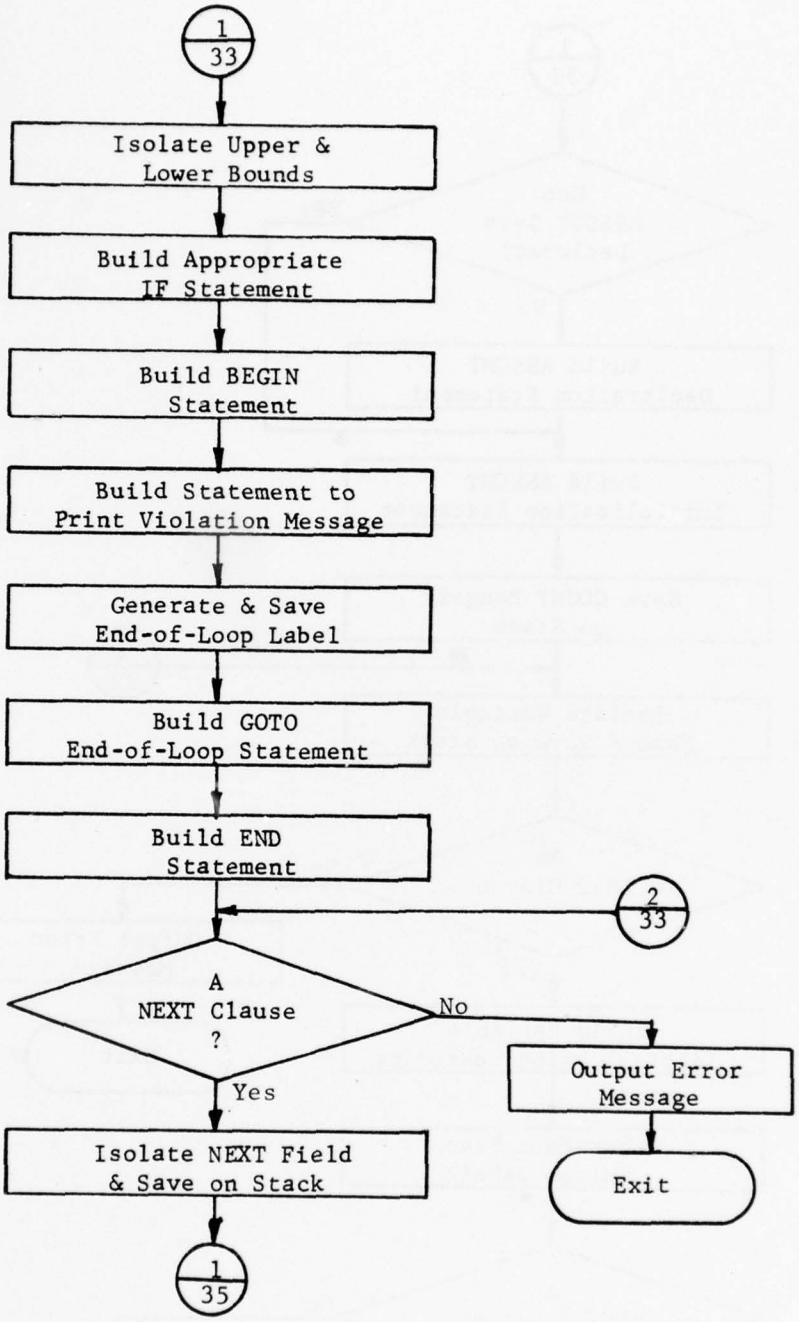
The upper-level flow of the JAVS system has not been significantly affected by the addition of the assertion directive capabilities.

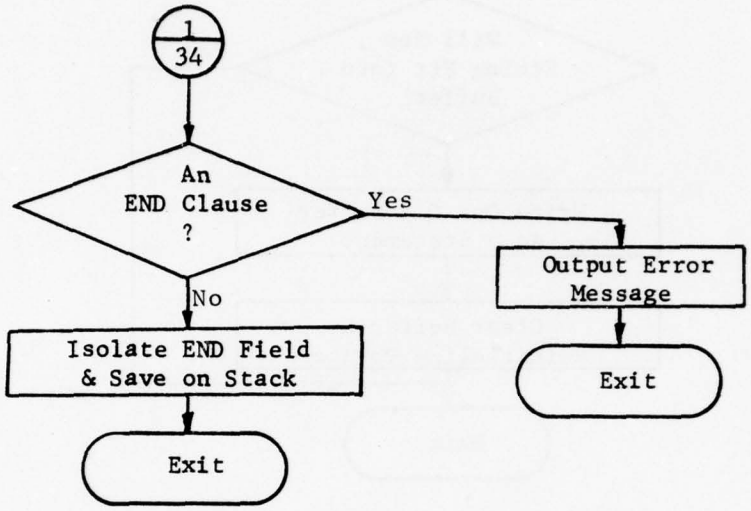
H. Flow Chart Diagrams

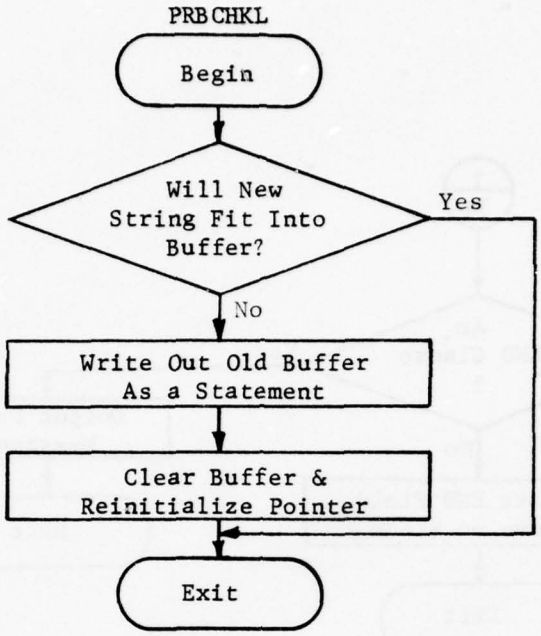
This section includes detailed programming flow charts for each of the modules that was added to the JAVS system. Also included is the flow chart for PRBDIR, the only procedure with major flow modifications.

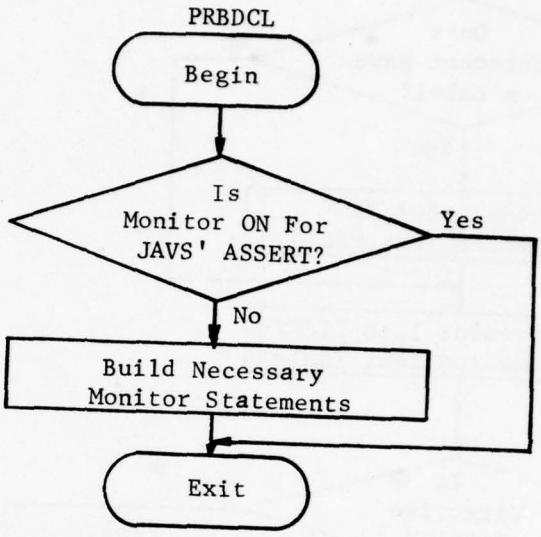

PRBCHAIN
Begin
PRBDCL
Set Up Monitor For Assertion Violation Output
Increment Assertion Command Stack Pointer
Stack Full?
Yes
Output Error Message
Exit
No
An EMPTY Clause ?
No
Yes
Isolate Boolean Expression
Build IF Statement
Build BEGIN Statement
A COUNT Clause ?
No
Set COUNT Option As Not Used on Stack
Yes
1
33
2
33

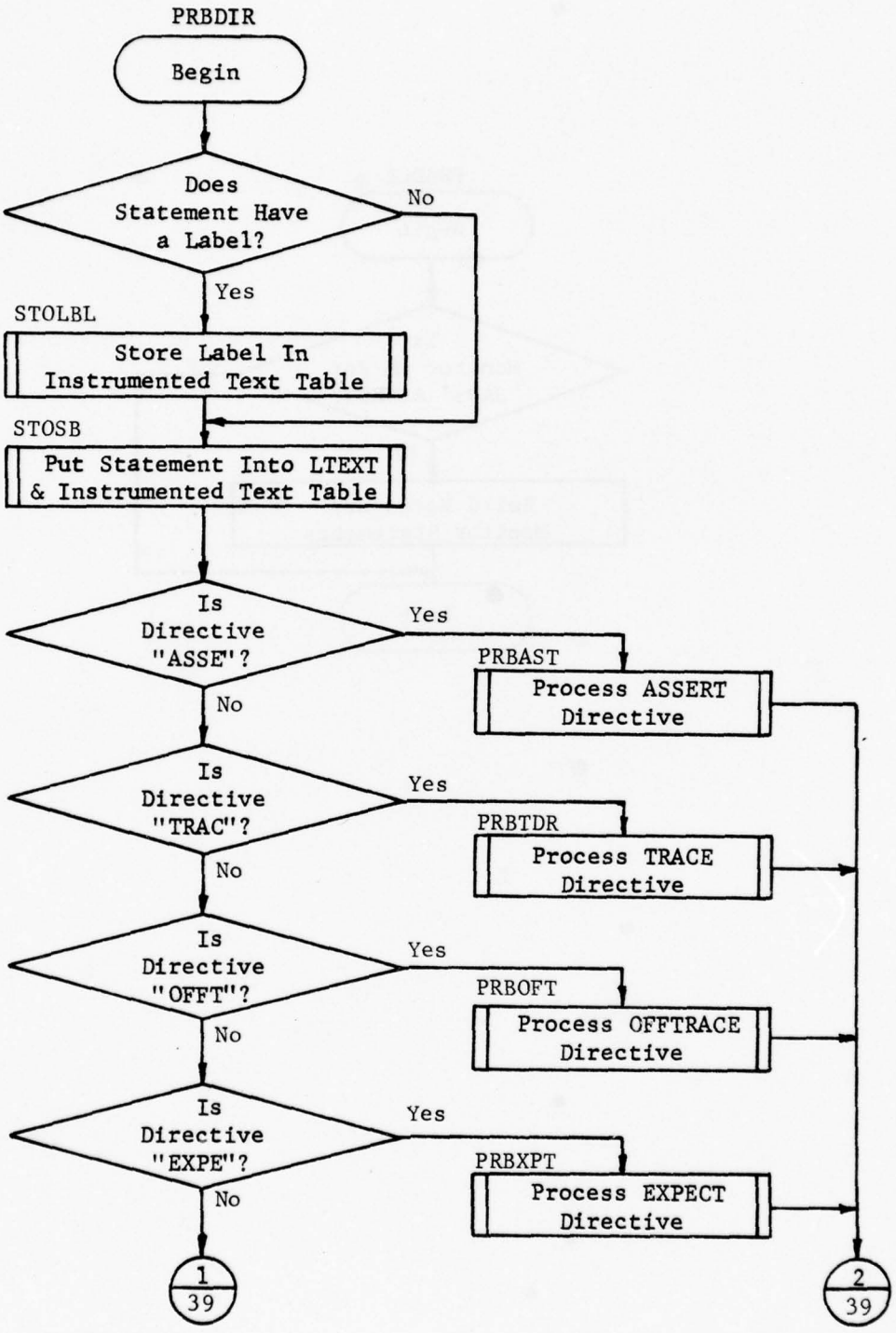

1
32
Has
ASSCNT Been
Declared?
Yes
No
Build ASSCNT
Declaration Statement
Build ASSCNT
Initialization Statement
Save COUNT Ranges
on Stack
2
32
Isolate Variable
Name & Save on Stack
An
INIT Clause
?
No
Output Error
Message
Exit
Yes
Build Variable
Initialization Statement
Generate & Save
Unique Label
A
BOUNDS Clause
?
No
Set BOUNDS Option As
Not Used on Stack
Yes
1
34
2
34

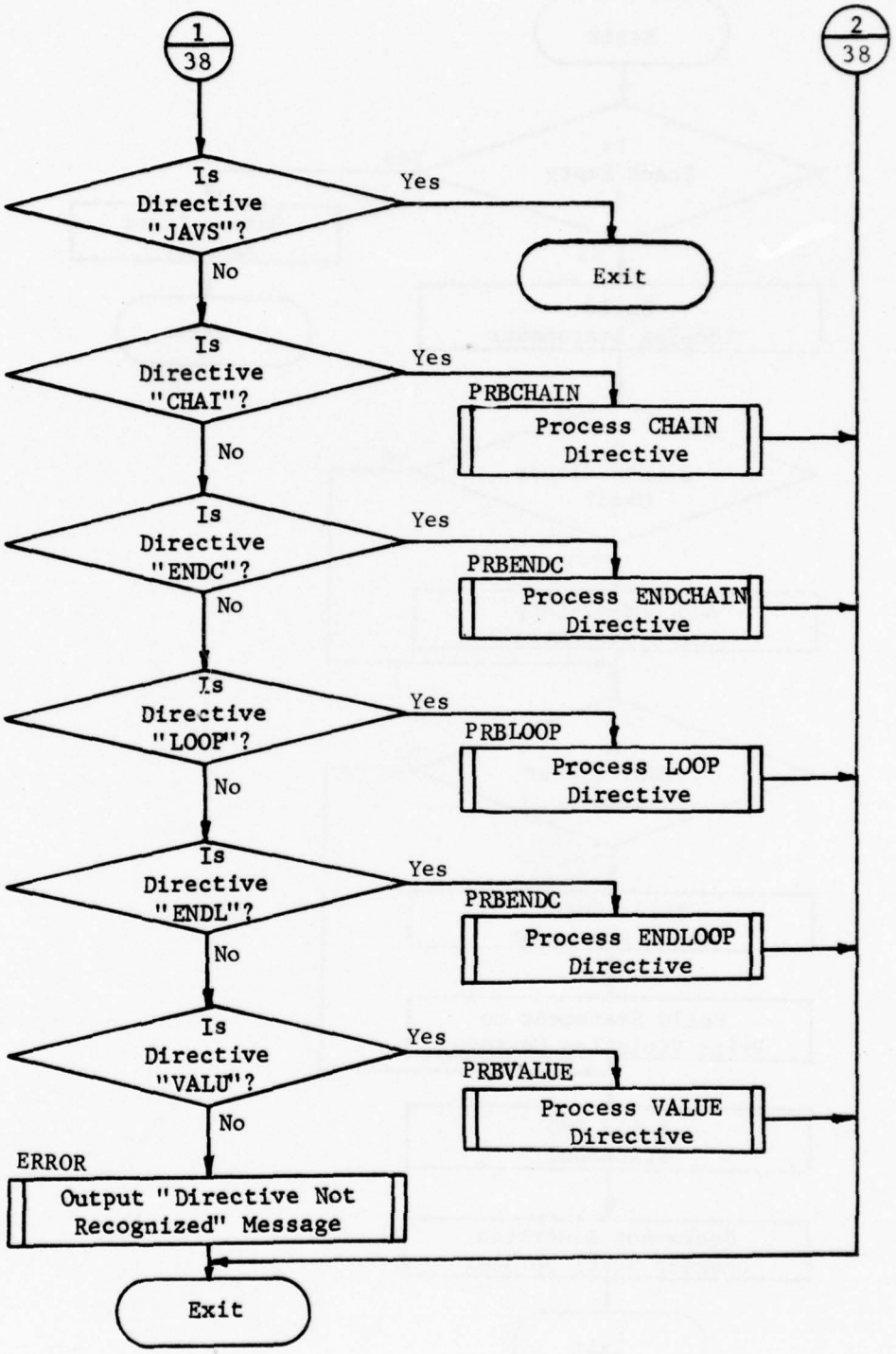

1
33
Isolate Upper &
Lower Bounds
Build Appropriate
IF Statement
Build BEGIN
Statement
Build Statement to
Print Violation Message
Generate & Save
End-of-Loop Label
Build GOTO
End-of-Loop Statement
Build END
Statement
2
33
A
NEXT Clause
?
No
Output Error
Message
Exit
Yes
Isolate NEXT Field
& Save on Stack
1
35

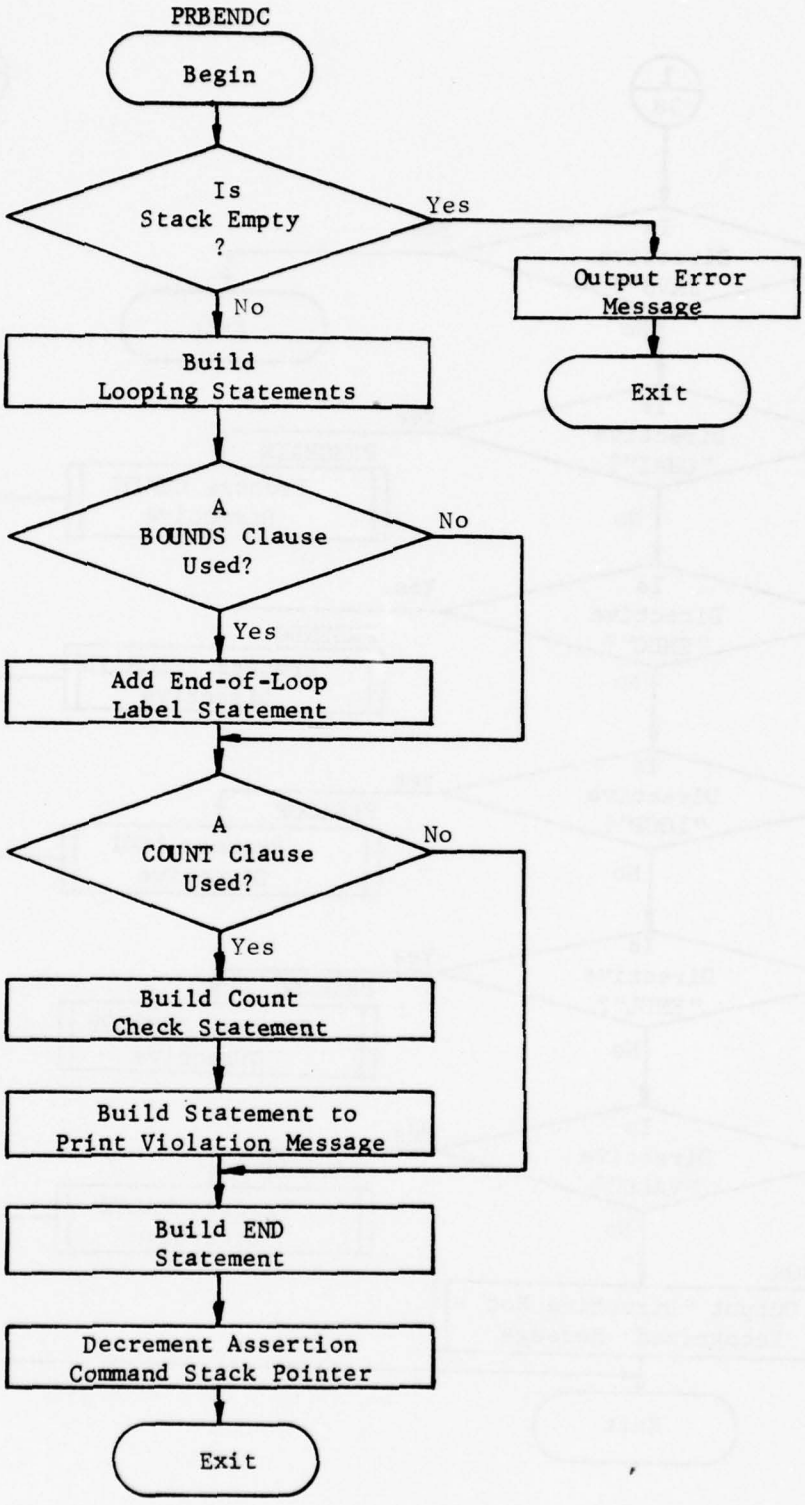

1
34
An
END Clause
?
Yes
No
Output Error
Message
Exit
Isolate END Field
& Save on Stack
Exit

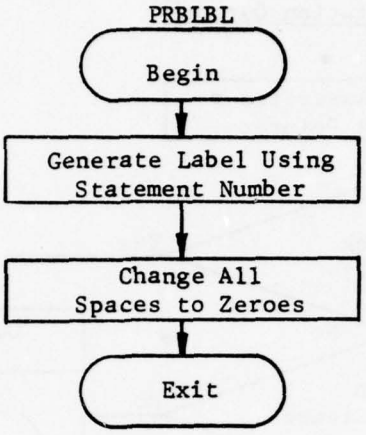

PRBCHKL
Begin
Will New String Fit Into Buffer?
Yes
No
Write Out Old Buffer As a Statement
Clear Buffer & Reinitialize Pointer
Exit

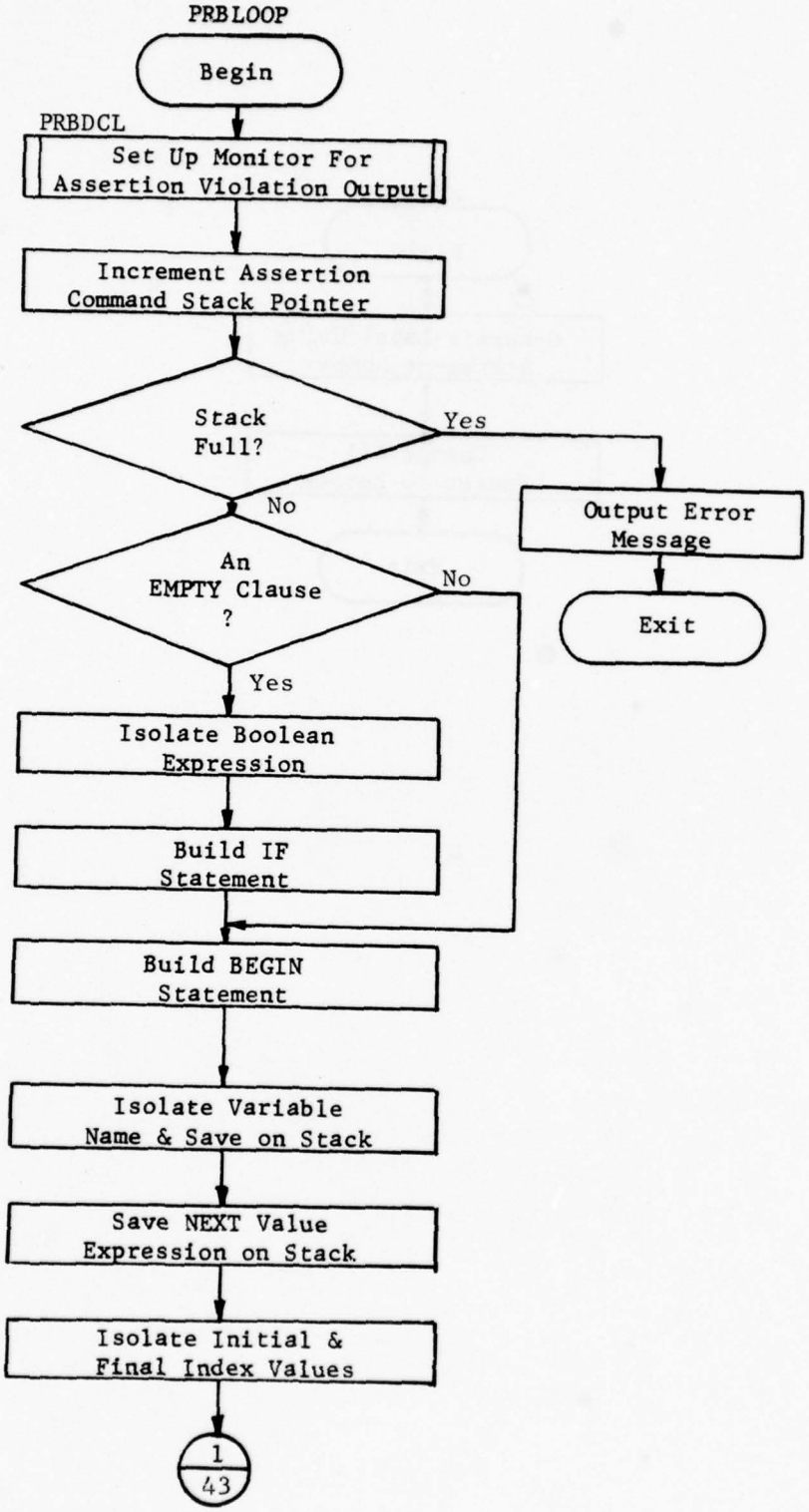

PRBDCL
Begin
Is
Monitor ON For
JAVS' ASSERT?
Yes
No
Build Necessary
Monitor Statements
Exit

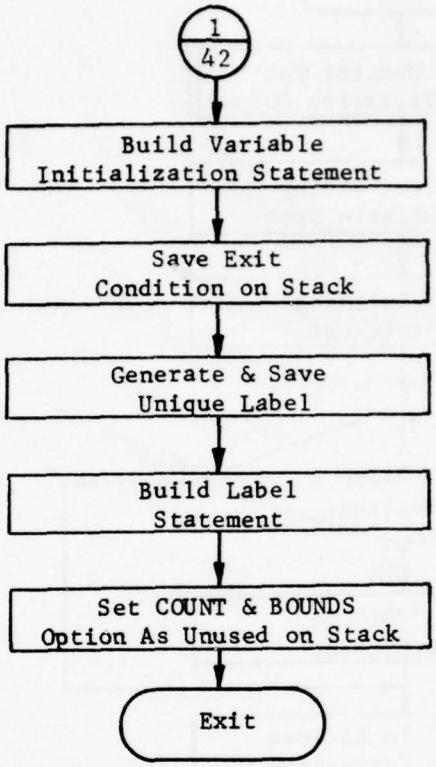

PRBDIR
Begin
Does Statement Have a Label?
No
Yes
STOLBL
Store Label In Instrumented Text Table
STOSB
Put Statement Into LTEXT & Instrumented Text Table
Is Directive "ASSE"?
Yes
PRBAST
Process ASSERT Directive
No
Is Directive "TRAC"?
Yes
PRBTDR
Process TRACE Directive
No
Is Directive "OFFT"?
Yes
PRBOFT
Process OFFTRACE Directive
No
Is Directive "EXPE"?
Yes
PRBXPT
Process EXPECT Directive
No
1/39
2/39

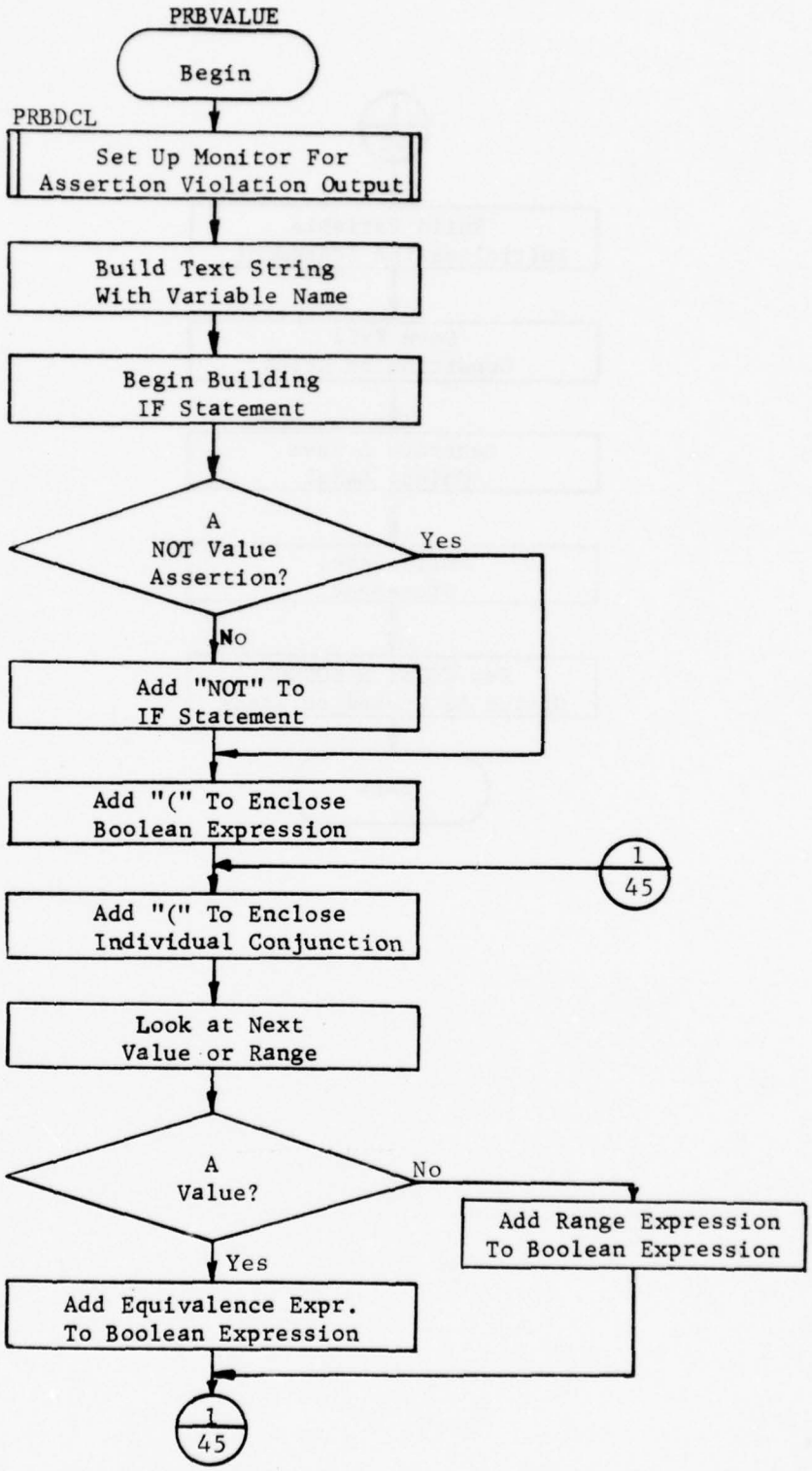

1
38
2
38
Is Directive "JAVS"?
Yes
Exit
No
Is Directive "CHAI"?
Yes
PRBCHAIN
Process CHAIN Directive
No
Is Directive "ENDC"?
Yes
PRBENDC
Process ENDCHAIN Directive
No
Is Directive "LOOP"?
Yes
PRBLOOP
Process LOOP Directive
No
Is Directive "ENDL"?
Yes
PRBENDC
Process ENDLOOP Directive
No
Is Directive "VALU"?
Yes
PRBVALUE
Process VALUE Directive
No
ERROR
Output "Directive Not Recognized" Message
Exit

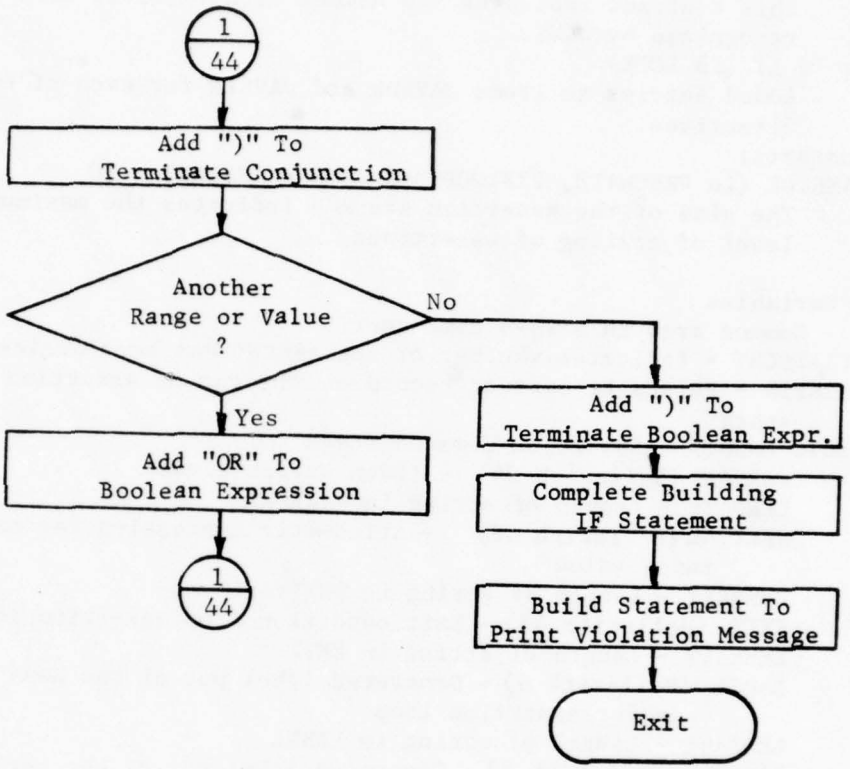

PRBENDC
Begin
Is
Stack Empty
?
Yes
Output Error
Message
Exit
No
Build
Looping Statements
A
BOUNDS Clause
Used?
No
Yes
Add End-of-Loop
Label Statement
A
COUNT Clause
Used?
No
Yes
Build Count
Check Statement
Build Statement to
Print Violation Message
Build END
Statement
Decrement Assertion
Command Stack Pointer
Exit


PRBLBL
Begin
Generate Label Using Statement Number
Change All Spaces to Zeroes
Exit


PRBLOOP
Begin
PRBDCL
Set Up Monitor For Assertion Violation Output
Increment Assertion Command Stack Pointer
Stack Full?
Yes
No
Output Error Message
Exit
An EMPTY Clause ?
No
Yes
Isolate Boolean Expression
Build IF Statement
Build BEGIN Statement
Isolate Variable Name & Save on Stack
Save NEXT Value Expression on Stack
Isolate Initial & Final Index Values
1
43


1
42
Build Variable Initialization Statement
Save Exit Condition on Stack
Generate & Save Unique Label
Build Label Statement
Set COUNT & BOUNDS Option As Unused on Stack
Exit


PRBVALUE
Begin
PRBDCL
Set Up Monitor For Assertion Violation Output
Build Text String With Variable Name
Begin Building IF Statement
A NOT Value Assertion?
Yes
No
Add "NOT" To IF Statement
Add "(" To Enclose Boolean Expression
1
45
Add "(" To Enclose Individual Conjunction
Look at Next Value or Range
A Value?
No
Add Range Expression To Boolean Expression
Yes
Add Equivalence Expr. To Boolean Expression
1
45


1
44
Add ")" To
Terminate Conjunction
Another
Range or Value
?
No
Yes
Add "OR" To
Boolean Expression
1
44
Add ")" To
Terminate Boolean Expr.
Complete Building
IF Statement
Build Statement To
Print Violation Message
Exit

I. Program Constants

Modified constants:

NJDIR (in LOOK) - changed from 11 to 16

This constant indicates the number of directives that are recognized by JAVS.

Table L1 (in LOOK)

Added entries to items JAVSDR and JAVSLN for each of the new directives.

New constants:

MAXSTCK (in PRBCHAIN, PRBLOOP and common ASSERT) = 10

The size of the assertion stack - indicates the maximum level of nesting of assertions

J. Program Variables

ASSERT - Common area in JAVS-5 component

DCLASSCNT - Indicates whether or not ASSCNT has been declared yet

CSTKPTR - Points to current record on the top of assertion command stack

Table CMDSTK - Assertion Command Stack

VARNAME (Hollerith 24) - Index variable name

LENNAME - Length of string in VARNAME

NEXTVAL (Hollerith 48) - Arithmetic expression for next index value

LENNEXT - Length of string in NEXTVAL

EXIT (Hollerith 48) - Exit condition from assertion loop

LENEXIT - Length of string in EXIT

LABEL (Hollerith 6) - Generated label put at the head of JOVIAL code for assertion loop

LENLABL - Length of string in LABEL

ELABEL (Hollerith 6) - Generated label put at the termination of JOVIAL code for assertion loop

LENELBL - Length of string in ELABEL

MINCNT (Hollerith 24) - Minimum of record COUNT range

LENMIN - Length of string in MINCNT

MAXCNT (Hollerith 24) - Maximum of record COUNT range

LENMAX - Length of string in MAXCNT

PRBCHAIN (MODULE, STMT) - Procedure in JAVS-5 component

ASSLEN - Holds the length of the string in ASSTXT

ASSTXT - String buffer used used for strings that are built from entries in the current statement block

FROM - Marks a position in the current SB.
Usually used for an opening parenthesis

KEYWRD - Marks a position in the current SB.
Used for marking various keywords

MIDL - Marks a position in the current SB.
Usually used for the range operator

MODULE - Number of the currently active module

STMT - Number of the current statement

TO - Marks a position in the current SB.
Usually used for the closing parenthesis

TMPTXT - Used as a temporary string buffer when calling routines
JUSTRT and LASTCH to determine the length of the string
in ASSTXT
TXTPTR - Contains the character count for the statement being
built in TXTBUF

PRBCHKL (MODULE, CHARS, LENGTH = NEW'LENGTH) - Procedure in JAVS-5
component
CHARS - Length of string to be added to TXTBUF
LENGTH - Length of current string in TXTBUF
MODULE - Number of the currently active module
NEW'LENGTH - Length of the string in TXTBUF at termination of
PRBCHKL

PRBDCL - Procedure in JAVS-5 component
No local variables

PRBENDC (MODULE, STMT) - Procedure in JAVS-5 component
ASSLEN - Holds the length of the string in ASSTXT
ASSTXT - String buffer used for strings that are built from entries
in the current statement block
MODULE - Number of the currently active module
STMT - Number of the current statement
TXTPTR - Contains the character count for the statement being
built in TXTBUF

PRBLBL (STMT, NO = LABEL) - Procedure in JAVS-5 component
LABEL - String buffer (length = 6) for newly generated label
NO - One digit used as the sixth character of the label
STMT - Number of the current statement

PRBLOOP (MODULE, STMT) - Procedure in JAVS-5 component
ASSLEN - Holds the length of the string in ASSTXT
ASSTXT - String buffer used for strings that are
built from entries in the current statement block
FROM - Marks a position in the current SB.
Usually used for an opening parenthesis
KEYWRD - Marks a position in the current SB.
Used for marking various keywords
MIDL - Marks a position in the current SB.
Usually used for the range operator
MODULE - Number of the currently active module
STMT - Number of the current statement
TO - Marks a position in the current SB.
Usually used for the closing parenthesis
TXTPTR - Contains the character count for the
statement being built in TXTBUF
VARLEN - Holds the length of the string in VARTXT
VARTXT - String buffer that is used to save the
variable name that is extracted from the assertion

PRBVALUE (MODULE, STMT) - Procedure in JAVS-5 component

ASSLEN - Holds the length of the string in ASSTXT
ASSTXT - String buffer used for strings that are
built from entries in the current statement block
FROM - Marks a position in the current SB.
Usually used for an opening parenthesis
MIDL - Marks a position in the current SB.
Usually used for the range operator
MODULE - Number of the currently active module
RPAREN - Marks the position of the right
parenthesis surrounding the list of values
in the assertion
STMT - Number of the current statement
TO - Marks a position in the current SB.
Usually used for the closing parenthesis
TXTPTR - Contains the character count for the
statement being built in TXTBUF
VARLEN - Holds the length of the string in VARTXT
VARTXT - String buffer that is used to save the
variable name that is extracted from the assertion

K. Inputs

All inputs are in the same form as before. All inputs that were previously recognized are still valid. The effect of this modification has been to expand the set of directives that JAVS would accept as input. The new JAVS directives that are acceptable as input have syntaxes as defined below:

".CHAIN (⟨variable name⟩) INIT (⟨arith. expr.⟩) NEXT (⟨arith. expr.⟩) END (⟨boolean expr.⟩) [EMPTY (⟨boolean expr.⟩)] [BOUNDS (⟨range⟩)] [COUNT (⟨range⟩)]"

".ENDCHAIN"

".ENDLOOP"

".LOOP (⟨variable name⟩) (⟨range⟩) [EMPTY (⟨boolean expr.⟩)]"

".VALUE (⟨variable name⟩) [NOT] (⟨list of values and/or ranges⟩)"

L. Output

All output generated by the additions to JAVS takes the form of that which is generated by the unmodified version of JAVS. Like the original JAVS computational directives, each new assertion directive causes the generation of JOVIAL code which is inserted into the probed text. All error diagnostics are output using the standard ERROR procedure so their form is also consistent with that of the error messages of the unmodified JAVS.

M. Error Messages

ASSERTION STACK OVERFLOW
There is an overflow condition on the assertion command stack. The

number of nested LOOP and CHAIN directives exceeds the permissable limit. This limit is defined as MAXCSTK (=10) in procedures PRBCHAIN and PRBLOOP and in common ASSERT. Error detected in procedure PRBCHAIN or PRBLOOP.

ASSERTION STACK UNDERFLOW
There is an underflow condition on the assertion command stack. For the ENDLOOP or ENDCHAIN directive being processed, there has been no matching LOOP or CHAIN directive. Error detected in procedure PRBENDC.

CHAIN ASSERTION ERROR - NO END FIELD
No END field specified for a CHAIN assertion directive. Error detected in procedure PRBCHAIN.

CHAIN ASSERTION ERROR - NO INIT FIELD
No INIT field specified for a CHAIN assertion directive. Error detected in procedure PRBCHAIN.

CHAIN ASSERTION ERROR - NO NEXT FIELD
No NEXT field specified for a CHAIN assertion directive. Error detected in procedure PRBCHAIN.

N. Operating Instructions

There are no new special operating instructions to go with these modifications.

APPENDIX IV

JAVS Modification Listings

This section contains the partial compilation listings for the modified version of JAVS. JAVS is a large system which contains several hundred different modules. It is therefore impossible to include here the listings for each of those modules. What is included are listings for every new procedure and every procedure that was modified. The listing of the compool that contains the added common area is also included. For a description of the other modules in the system, refer to [17].

```
2076T 02  05-03-78   10.422     JOVIAL COMPILATION OF LOOK            JOCIT VERSION 042275            PAGE      1

    ALTER NO      ****5****1****5****2****5****3****5****4****5****5****5****6****5****7****5****8                ANS1

 1            ''.JAVSTEXT LOOK COMPUTE (J2P) = JAVS2 EXECUTABLE MODULES ''
 2             START
 3            PROC LOOK $                    ''PROCEDURE WHICH GETS NEXT PART
 4                                             OF SPEECH''
 5                  .!      JAVS-2 COMPONENT      ''
 6            ''.GROUP J2SB = STATEMENT AND STATEMENT DESCRIPTOR BLOCK CONSTRUCTION''
 7            ''.PROCESS SCANTEXT=JOVIAL SYMBOL ANALYSIS''
 8                        ''THIS PROCEDURE REPLACES THE ORIGINAL LOOK WHICH
 9                          IS RENAMED OLOOK.  RATHER THAN INSERT CODE AFTER
10                          EACH CALL TO LOOK, WE DID IT THIS WAY TO MINIMIZE
11                          THE NUMBER OF CHANGES.''
12            DEFINE COMBP  ''02DICT(900)   ''  $
13            DEFINE NJDIR ''16'' ''NUMBER OF JAVS DIRECTIVES'' $
14            ITEM T1 N 6 $
15                  DEFINE INTG '' I 24 S '' $
16                  DEFINE NLL '' N 4 '' $
17            ITEM T2 INTG $
18            TABLE L1 R NJDIR $ 2 $
19              BEGIN
20              ITEM JAVSDR T 6 0 0  $
21                BEGIN
22                6T(ASSERT)
23                6T(EXPECT)
24                6T(TOLERA)
25                6T(TRACE )
26                6T(OFFTRA)
27                6T(JAVSTE)
28                6T(GROUP )
29                6T(PROCES)
30                6T(FUNCTI)
31                6T(CLASS )
32                6T(CONTRO)
33                6T(LOOP  )        ''LOOP DIRECTIVE -- JLR, 2/16/78''
34                6T(ENDLOO)        ''ENDLOOP DIRECTIVE -- JLR, 2/16/78''
35                6T(CHAIN )        ''CHAIN DIRECTIVE -- JLR, 2/20/78''
36                6T(ENDCHA)        ''ENDCHAIN DIRECTIVE -- JLR, 2/20/78''
37                6T(VALUE )        ''VALUE DIRECTIVE -- JLR, 3/21/78''
38                END
39              ITEM JAVSLN I 24 U 1 0  $
40                BEGIN
41                      6 6 9 5
42                $
43            $ 5 7 8 5 7
44               4  ''LOOP''
45               7  ''ENDLOOP''
46               5  ''CHAIN''
47               8  ''ENDCHAIN''
48               5  ''VALUE''
49                END
50              END
```

```
ALTER NO      ...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8          ANZC

1             '!! JAVSTEXT J5POOL PRESET !!
2              START $                                                                    J4CMPL 2
3                   !!     JAVS-5 --  MODULE INSTRUMENTATION COMPONENT      !!         0729RU 6
4                   !!                                                                   TEMP   1
5                     DEFINE TRUE '' 1 '' $                                              BDEFINS2
6                     DEFINE FALSE '' 0 '' $                                             BDEFINS3
7                     DEFINE INTG '' I 24 S !! $                                         DEFINS 2
8                     DEFINE HLL '' H 4 '' $                                             DEFINS 3
9                     DEFINE DINTG '' I 48 S !! $                                        DDEFINS2
10                    DEFINE DHLL '' H 8 !! $                                            DDEFINS3
11                  !!   0....0....  LOCAL COMMON BLOCKS .0....0.0.0  !!                J4CMPL 6
12                  !!                                !!                               BLKSTO 2
13                  !!     BLOCK STORAGE COMMON       !!                               BLKSTO 3
14                  !!                                !!                               BLKSTO 4
15             COMMON BLKSTO $                                                         BLKSTO 5
16               BEGIN                                                                 BLKSTO 6
17                 ARRAY IX 100 INTG $                                                 BLKSTO 7
18                   ARRAY LX 100 HLL  $                                               BLKSTO 8
19                   OVERLAY IX = LX $                                                 BLKSTO 9
20               END                                                                   BLKSTO10
21                  !!                                          !!                     GLOBAL 2
22                  !!     GLOBAL SYMBOL DEFINITION COMMON      !!                     GLOBAL 3
23                  !!                                          !!                     GLOBAL 4
24             COMMON GLOBAL $                                                         GLOBAL 5
25               BEGIN                                                                 GLOBAL 6
26                 ITEM IGLOBLK HLL $                                                  GLOBAL 7
27                 ITEM IGLOUND HLL $                                                  GLOBAL 8
28                 ITEM LBLANK HLL $                                                   GLOBAL 9
29                 ITEM IBLANK INTG $                                                  GLOBAL10
30                 OVERLAY LBLANK = IBLANK $                                           GLOBAL11
31               END                                                                   GLOBAL12
32                  !!                                    !!                          MTHSTO 2
33                  !!     STATEMENT LIST STORAGE COMMON  !!                          MTHSTO 3
34                  !!                                    !!                          MTHSTO 4
35             COMMON MTHSTO $                                                         MTHSTO 5
36               BEGIN                                                                 MTHSTO 6
37                 ARRAY LLIST 2 250 HLL $                                             MTHSTO 7
38                 ARRAY ILIST 2 250 INTG $                                            MTHSTO 8
39                 OVERLAY LLIST = ILIST $                                             MTHSTO 9
40               END                                                                   MTHSTO10
41                  !!                                 !!                            TXTSTO 2
42                  !!     TEXT STORAGE COMMON         !!                            TXTSTO 3
43                  !!                                 !!                            TXTSTO 4
44               COMMON TXTSTO $                                                     TXTSTO 5
45               BEGIN                                                               TXTSTO 6
46                 ARRAY LTEXT 250 HLL $                                             TXTSTO 7
47                 ARRAY ITEXT 250 INTG $                                            TXTSTO 8
48                 OVERLAY LTEXT = ITEXT $                                           TXTSTO 9
49               END                                                                 TXTSTO10
50               COMMON BUFFER $                                                     J4CMPL13
```

```
ALTER NO      ....5....6....1....8....5....2....9....3....6....5....4....8....5....6....5....6....5....6....5....7....5....8          ANZC

                BEGIN                                                                                    J4CMPL14
                  ARRAY LBUFF 100 WLL $                                                                  J4CMPL15
                  ARRAY IBUFF 100 INTG $                                                                 J4CMPL16
                  OVERLAY LBUFF = IBUFF $                                                                J4CMPL17
                END                                                                                      J4CMPL18
                COMMON TABNOS $                                                                          J4CMPL19
                BEGIN                                                                                    J4CMPL20
                  ITEM DDPNUM INTG $                                                                     J4CMPL21
                  ITEM DSNUM  INTG $                                                                     J4CMPL22
                  ITEM PRBNUM INTG $                                                                     J4CMPL23
                  ITEM MDBNUM INTG $                                                                     J4CMPL24
                  ITEM SBNUM  INTG $                                                                     J4CMPL25
                  ITEM SDBNUM INTG $                                                                     J4CMPL26
                  ITEM SLTNUM INTG $                                                                     J4CMPL27
                  ITEM STBNUM INTG $                                                                     J4CMPL28
                END                                                                                      J4CMPL29
                COMMON DIRCOM $                                                                          0604JB 1
                BEGIN                                                                                    0604JB 2
                ''COMMON FOR DIRECTIVES GLOBAL VARIABLES''                                               0604JB 3
                  ARRAY STACK 20 INTG $  ''POINTERS TO START OF SYMBOL IN SB''                           0604JB 4
                  ITEM COMMA WLL P 4H(,  ) $                                                             0604JB 5
                  ITEM DECLARED B $                                                                      0604JB 6
                  ITEM FLAGVAR H 8 P 8H(FLEG(   )$                                                       C022JUL1
                  ITEM INEXT INTG $                                                                      0604JB 8
                ITEM MAXTXT INTG P 148 $                                                                 0806JB 1
                  ITEM STACKPTR INTG $                                                                   0604JB12
                  ITEM TABLPRESENT B P 0$                                                                0730JB 1
                  ITEM TVBNUM INTG $                                                                     0604JB15
                  ITEM TVINUM INTG $                                                                     0604JB16
                  ITEM TXTBUF W 148 $                                                                    0806JB 2
                END                                                                                      0604JB18
              COMMON ASSERT $
              BEGIN
              ''COMMON FOR ASSERTIONS COMMAND STACK,''
              ''      JOHN L. RAMEY -- 2/27/78''
                 DEFINE MAXCSTK ''10'' $
                 ITEM DCLASSCNT B $ ''INDICATES IF ASSCNT HAS BEEN DECLARED YET''
                 ITEM CSTKPTR INTG $ ''POINTS TO CURRENT TOP OF STACK''
                 TABLE CMDSTK R MAXCSTK $ ''ASSERTION COMMAND STACK''
                 BEGIN
                    ITEM VARNAME H  24 $  ''INDEX NAME''
                    ITEM LENNAME INTG $  ''LENGTH OF INDEX NAME''
                    ITEM NEXTVAL H 48 $  ''ARITH EXPR. FOR NEXT INDEX VALUE''
                    ITEM LENNEXT INTG $  ''LENGTH OF NEXTVAL''
                    ITEM EXIT    H 48 $  ''EXIT CONDITION''
                    ITEM LENEXIT INTG $  ''LENGTH OF EXIT''
                    ITEM LABEL   H  6 $  ''LABEL AT TOP OF LOOP''
                    ITEM LENLABL INTG $  ''LENGTH OF LABEL''
                    ITEM ELABEL  H  6 $  ''LABEL AT EXIT FROM LOOP''
                    ITEM LENELBL INTG $  ''LENGTH OF ELABEL''
```

```
ALTER NO      ....5....0....1....0....5....0....2....0....5....0....3....0....5....0....4....0....5....0....5....0....6....0....5....0....7....5....8        ANSC

101                 ITEM MINCNT  H  24 $  I''MIN OF RECORD CNT RANGE''
102                 ITEM LENMIN  INTG $  ''LENGTH OF MINCNT''
103                 ITEM MAXCNT  H  24 $  I''MAX OF RECORD CNT RANGE''
104                 ITEM LENMAX  INTG $ ''LENGTH OF MAXCNT''
105               END
106           END
107                 COMMON J4COM $                                                          J4CMPL30
108                 BEGIN                                                                   J4CMPL31
109             ARRAY ASTSTX 10 DHLL $  ''PROBE START TEST TEXTS''
110             ARRAY STRIES 10 DHLL $  ''PROBE START TEST NAMES''
111             ARRAY STRTFG 10 HLL $  ''PROBE START TEST TFLAG''
112             ARRAY ASTSMD 10 DHLL $   ''PROBE START TEST MODULE ''
113             ARRAY ASTSSM 10 INTG $   '' PROBE START TEST SMT. NO. ''
114             ITEM NPI INTG $          '' NO. OF START TEST PROBE CALLS ''
115                   ARRAY FBSTACK 100 INTG $  '' FALSE BRANCH STACK FOR IFTS ''       J4CMPL32
116                   ARRAY FBPATH 100 INTG $  '' FALSE BRANCH DD-PATH NUMBERS FOR      J4CMPL33
117                                             IF STATEMENTS ''                        J4CMPL34
118                   ARRAY FDDPS 100 INTG $  '' DD-PATH NUMBERS FOR FALSE BRANCH       J4CMPL35
119                                             IN IFEITH-ORIF-END STATEMENTS ''        J4CMPL36
120                   ARRAY IFWEST 100 INTG $  '' STACK OF FALSE BRANCHES FOR IFEITH    J4CMPL37
121                                             ORIF-END STATEMENTS ''                  J4CMPL38
122                   ITEM BEGST INTG $  ''STARTING WORD-PAIR OF STATEMENT IN BB''      J4CMPL39
123                   ITEM BLANKS H 150 P 150H                                          J4CMPL40
124                                                                                     J4CMPL41
125                                                  ) $                                J4CMPL42
126                   ITEM BLNR HLL P 4H(     ) $   '' HOLERITH BLANK ''              J4CMPL43
127                   ITEM BLOCK INTG $   ''INDEX OF CORRENT PROBE BLOCK BEING ''      J4CMPL44
128                                       ''STORED IN THE PROBE TABLE''                 J4CMPL45
129                   ITEM BUFOUT H 150 $   '' TEXT EXTRACTION BUFFER ''              J4CMPL46
130                   ITEM CFOR INTG $  ''DD NUMBER OF CONTROLLING FOR OF             J4CMPL47
131                                       PARALLEL FOR''                                J4CMPL48
132                 ITEM ENDMOD  DHLL $   ''MODULE WHICH ENDS TESTCASE''              J4CMPL50
133                 ITEM ENDSTMT HLL  $   ''STATEMENT WHICH ENDS TESTCASE''           J4CMPL51
134                   ITEM IENDST INTG  $          ''SAME AS ENDSTMT''                J4CMPL52
135                   OVERLAY IENDST 0 ENDSTMT $
136                 ITEM ENDTXT  DHLL $   ''JAVSTEXT WHICH ENDS TESTCASE''            J4CMPL54
137                   ITEM DEBUG B $  ''DEBUGGING OUTPUT FLAG''                       J4CMPL55
138                   ITEM FBLOCK INTG $  ''INDEX OF FIRST PROBE BLOCK FOR PROBED ''  J4CMPL56
139                                       ''STATEMENT''                               J4CMPL57
140                   ITEM FIRSTX INTG $  '' FIRST EXECUTABLE STATEMENT ''            J4CMPL58
141                   ''TEMPLATE FOR FIRST PROBE ''                                   J4CMPL59
142                   ITEM FRSTPRB H 52 P 52H(PROBE    ( 8H(MODULNAM) ; 8H(TEXTNAME) ,J4CMPL60
143             !!!! ) $   ) $                                                        J4CMPL61
144                   ITEM HDUMMY HLL $  ''I DUMMY HOLLERITH PARAMETER ''             J4CMPL62
145                   ITEM HTEMP HLL $  '' TEMPORARY HOLLERITH VARIABLE ''            J4CMPL63
146                   ITEM IDUMMY INTG $  '' DUMMY INTEGER PARAMETER ''               J4CMPL64
147                   ITEM ITYPE INTG $   ''INTEGER STATEMENT TYPE''                  J4CMPL65
148                   ITEM KBLOCK INTG $  ''INDEX OF PROBE STATEMENT BLOCK BEGIN''    J4CMPL67
149                                       ''STORED ''                                 J4CMPL68
150                   ITEM LASTDDP INTG $   '' LAST DD-PATH SEARCHED ''               J4CMPL69
```

```
2908† 82  05401-78   14.80‡     JOVIAL COMPILATION OF SORCE5          JOCI† VERSION 042275                PAGE      1

ALTER NO      ...1...2...3...4...5...6...7...8                                                          ANZ1

 1            '' JAVSTEXT STEP3 COMPUTE (USPOBL) ''
 2             START
 3
 4                  PROC STEP3 (MODULE; LNUM, UTABLE = !ANY) S                                    STEP3  3
 5                  ''     JAV8-5  --  MODULE INSTRUMENTATION COMPONENT      ''                 0729RU 6
 6                  ''     ''                                                                   TEMP   1
 7                  '' JAVS TEST CASE INSTRUMENTATION STEP ''                                   STEP3  4
 8                  ''REVISION OF APRIL 15, 1975''                                              STEP3  5
 9                    DEFINE INTG '' I 24 S ''S                                                 DEFINS 2
10                    DEFINE HLL '' H 4 '' S                                                    DEFINS 3
11                    DEFINE DINTG '' I 48 S ''S                                                DDEFINS2
12                    DEFINE DHLL '' H 8 ''I S                                                  DDEFINS3
13                    DEFINE TRUE '' 1 '' S                                                     BDEFINS2
14                    DEFINE FALSE '' 0 '' S                                                    BDEFINS3
15                    DEFINE NBYTWD '' 4 ''I S                                                  MACHDEP2
16                  ARRAY LTABLE 2 25 DHLL S                                                    STEP3  10
17                  ITEM !ANY INTG S                                                            STEP3  12
18                  ITEM LNUM INTG S                                                            STEP3  13
19                  ITEM MODULE INTG S                                                          STEP3  14
20                  ITEM LTES DHLL S
21                  ITEM HLFLG HLL S
22                  BEGIN                                                                       STEP3  15
23                    IF MCALLED5 EQ 0 S
24                      BEGIN                                                                   STEP3  18
25                        MCALLED5 = 1 S
26                        VPRBE = BH(PROBE    ) S                                             STEP3  20
27                        VPRBI = BH(PROBI    ) S                                             STEP3  21
28                        VPRBM = BH(PROBM    ) S                                             STEP3  22
29                        PMODE = PATHS S                                                     STEP3  23
30                        DEBUG = FALSE S                                                     STEP3  25
31                        ISTRST = -1S                                                        STEP3  27
32                        IENDST = -1S                                                        STEP3  28
33                       NPI = 0 S
34                      END                                                                   STEP3  29
35                    IANY = 1 S                                                              STEP3  30
36                    IFEITH LTABLE(S0; 0S) EQ BH(INSTRUME) S                                 STEP3  31
37                      BEGIN                                                                 STEP3  32
38                        IFEITH LNUM EQ 1 S                                                  STEP3  33
39                          BEGIN                                                             STEP3  34
40                    IF MODULE LQ 0 S                                                        STEP3  35
41                      BEGIN                                                                 STEP3  36
42                        ERROR (18H(BAD MODULE NUMBER;) ) S                                  STEP3  37
43                        IANY = 0 S                                                          STEP3  38
44                        RETURN S                                                            STEP3  39
45                      END                                                                   STEP3  40
46                           DDPNUM = NUMDDP (IBUMMY) S                                       STEP3  41
47                           MDBNUM = NUMMDB (IBUMMY) S                                       STEP3  42
48                           SBNUM = NUMSB (IBUMMY) S                                         STEP3  43
49                           SDBNUM = NUMSDB (IBUMMY) S                                       STEP3  44
50                           SLTNUM = NUMSLT (IBUMMY) S                                       STEP3  45
```

```
ALTER NO      ....5....1....5....2....5....3....5....4....5....5....5....6....5....7....5....8          ANZ1

                        $TBNUM = NUMSTB (IDUMMY) $                                          STEP3 46
                        $RBNUM = NUMPRB ( $DUMMY ) $                                        STEP3 47
                        $PROBE (MODULE) $                                                   STEP3 48
                        MCALLED$ = 0 $
                      END                                                                   STEP3 50
                    ORIF LTABLE ($ 0 , 1 $) EQ 8H(PROBE   ) $                               STEP3 51
                      BEGIN                                                                 STEP3 52
                        IFEITH LTABLE($0,2$) EQ 8H(DDPATH  ) $                              STEP3 53
                          VPRBE = LTABLE($$,8$) $                                           STEP3 54
                        ORIF   LTABLE($0,2$) EQ 8H(MODULE  ) $                              STEP3 55
                          VPRBM = LTABLE($$,8$) $                                           STEP3 56
                        ORIF   LTABLE($0,2$) EQ 8H(TEST    ) $                              STEP3 57
                          VPRBI = LTABLE($$,8$) $                                           STEP3 58
                        ORIF 1 $                                                            STEP3 59
                          IANY = 0 $                                                        STEP3 60
                        END                                                                 STEP3 61
                      END                                                                   STEP3 62
                    ORIF LTABLE($0,1$) EQ 8H(MODE    ) $                                    STEP3 63
                      BEGIN                                                                 STEP3 64
                        IFEITH LTABLE($1,1$) EQ 8H(FULL    ) $                              STEP3 65
                          PMODE = FULL $                                                    STEP3 66
                        ORIF LTABLE ($1,1$) EQ 8H(INVOCATI) $                               STEP3 67
                          PMODE = INVOKE $                                                  STEP3 68
                        ORIF LTABLE($1,1$) EQ 8H(DIRECTIV) $                                STEP3 69
                          PMODE = DIRTS $                                                   STEP3 70
                        ORIF LTABLE($1,1$) EQ 8H(DDPATHS ) $                                STEP3 71
                          PMODE = PATHS $                                                   STEP3 72
                        ORIF LTABLE($1,1$) EQ 8H(DD-PATHS) $                                STEP3 73
                          PMODE = PATHS $                                                   STEP3 74
                        ORIF 1 $                                                            STEP3 75
                          BEGIN                                                             STEP3 76
                            IANY = 0 $                                                      STEP3 77
                            BYTE($0,128$) (LINE) = BYTE($0,128$) (BLANKS) $                 STEP3 78
                            BYTE($0, 32$) (LINE) = 32H(DINSTRUMENTATION MODE BEGINNI STEP3 79
NG ) $                                                                                      STEP3 80
                            BYTE($32,8$) (LINE) = BYTE($6,8$) (LTABLE($1,1$)) $             STEP3 81
                            BYTE($40,21$) (LINE) = 21H( IS NOT A VALID MODE.) $             STEP3 82
                            OUTBUF (LENLIN, LINE) $                                         STEP3 83
                          END                                                               STEP3 84
                        END                                                                 STEP3 85
                      END                                                                   STEP3 86
                    ORIF LTABLE($0,1$) EQ 8H(STARTTES) $                                    STEP3 87
                      BEGIN                                                                 STEP3 88
                      NPI = NPI+1 $
                        STRMOD = LTABLE ($$,8$) $                                           STEP3 89
                        STRTXT = LTABLE($0,8$) $                                            STEP3 90
                IFSTRST = 0 $
                        BYTE($0,NBYTWD$) (STRSTMT) = BYTE($NBYTWD,NBYTWD$)                  STEP3 91
                                                     (LTABLE($0,3$)) $                      STEP3 92
                      IFEITH BYTE($0,8$)(LTABLE($0,4$)) EQ 8H(        ) $
```

```
 ALTER NO      ....5....1....5....2....5....3....5....4....5....5....5....6....5....7....5....8          ANZ1

101                     LTE$ = BW(CASE    ) $                                                    
102                   ORIF 1 $                                                                   
103                     LTE$ = LTABLE($0,4$) $                                                   
104                   END                                                                        
105                 BYTE($0,NBYTWD$)(MLFLG) = BYTE($NBYTWD,NBYTWD$)(LTABLE($0,5$)) $             
106                   IF NPI LQ 10 $                                                             
107                   BEGIN                                                                      
108                     ASTRT$(SNPI-1$) = S$RTXT $                                               
109                     ASTRMD(SNPI-1$) = S$RMOD $                                               
110                     ASTR$M(SNPI-1$) = I$TR$T $                                               
111                   STRTE$(SNPI-1$) = LTE$ $                                                   
112                   STRTFG(SNPI-1$) = MLFLG $                                                  
113                     END                                                                      
114                     END                                                                      STEP3 93
115                   ORIF LTABLE($0,1$) EQ BW(STOPTEST) $                                       STEP3 94
116                     BEGIN                                                                    STEP3 95
117                       ENDMOD  = LTABLE($$,$$) $                                              STEP3 96
118           IENDST = 0 $                                                                       
119                       ENDTXT  = LTABLE($0,$$) $                                              STEP3 97
120                       BYTE($0,NBYTWD$) (ENDSTMT) = BYTE($NBYTWD,NBYTWD$)                     STEP3 98
121                                                    (LTABLE($0,8$T) $                         STEP3 99
122                     END                                                                      STEP3100
123                   ORIF LTABLE ($0,1$) EQ BW(DEBUG   ) $                                      STEP3110
124                     DEBUG = TRUE $                                                           STEP3111
125                   ORIF 1 $                                                                   STEP3112
126                     IANY = 0 $                                                               STEP3113
127                   END                                                                        STEP3114
128                   END                                                                        STEP3115
129                   ORIF 1 $                                                                   STEP3116
130                     IANY = 0 $                                                               STEP3117
131                   END                                                                        STEP3118
132                 END                                                                          STEP3119
133                                                                                              
134                 PROC ADDTXT (TEXT'STRING, CHARS, LENGTH = NEW'LENGTH) $                      ADDTXT 3
135                 '!      JAVS-5  --  MODULE INSTRUMENTATION COMPONENT      !'                0729RU 6
136                 '!      !'                                                                   TEMP.  1
137                 '!ADDS TEXT TO THE CHARACTER STRING TXTBUF!'                                 ADDTXT 4
138                 DEFINE LEN'IN '(14$)' $                                                      0806JB 6
139                 ITEM TEXT'STRING H LEN'INS                                                   0725JB 1
140                   ITEM CHARS INTO $                                                          ADDTXT 8
141                   ITEM LENGTH INTO $                                                         ADDTXT 9
142                   ITEM NEW'LENGTH INTO $                                                     ADDTXT10
143                   BEGIN                                                                      ADDTXT11
144                 ''ASSUME THAT THE STRING IN TEXT'STRING IS RIGHT JUSTIFIED''                 0709CG18
145                 ''MOVE CHARACTERS FROM INPUT STRING TO TEXT BUFFER !'                        0709CG19
146                   IFEITH (LENGTH.CHARS) LQ MAXTXT $                                          0806JB 7
147                   BEGIN                                                                      0806JB 8
148                 BYTE($LENGTH+1,CHARS) (TXTBUF) = BYTE($LENIN-CHARS,CHARS$)                   0709CG20
149                 (TEXT'STRING)$                                                               0709CG21
150                 NEW'LENGTH = LENGTH-CHARS$                                                   0709CG22
```

```
ALTER NO  ...1...2...3...4...5...6...7...8  ANZ1
151                 END                                                    0806JB 9
152                 ORIE 1 $                                               0806JB10
153                   FATAL ( 31H(STRING LONGER THAN TEXT BUFFER?) ) $      0806JB11
154                 END                                                    0806JB12
155               END ''ADDTXT''                                           ADDTXT12
156
157               PROC BLDPRB (MODULE, DDPATH) $                           BLDPRB 3
158               ''     JAVB-5  --  MODULE INSTRUMENTATION COMPONENT     ''  0729RU 6
159               ''     ''                                                TEMP   1
160               '' BUILDS A PROBE INVOCATION IN LTEXT''                  BLDPRB 4
161               ITEM DDPATH INTG $                                       BLDPRB 7
162               ITEM MODP MLL $                                          BLDPRB 8
163               ITEM MODULE INTG $                                       BLDPRB 9
164               BEGIN                                                    BLDPRB10
165                 BYTE ($0,8$) (PRBUFR) = BYTE($0,8$) (VRRBE) $          BLDPRB11
166                 BYTE($13,8$) (PRBUFR) = BYTE($0,8$) (MODNAM) $         BLDPRB12
167                 BYTE($27,8$) (PRBUFR) = BYTE($0,8$) (TMTNAM) $         BLDPRB13
168                 MODP = ITSHOU ( DDPATH ) $                             BLDPRB14
169                 BYTE($36,4$) (PRBUFR) = BYTE($0,4$) (MODP) $           BLDPRB15
170                 FOR I = 0, 1; LENPRB-1 $                               BLDPRB16
171                 FOR U = 0, NBYTWD $                                    BLDPRB17
172                   BYTE($0,NBYTWD$) ( LTEXT($ I $) ) = BYTE($U,NBYTWD$) BLDPRB18
173                   (PRBUFR) $                                           BLDPRB19
174               END                                                      BLDPRB20
175
176               PROC BLDTXT (MODULE, STRNG$ INDENT = LENTXT) $           BLDTXT 3
177               ''     JAVB-5  --  MODULE INSTRUMENTATION COMPONENT     ''  0729RU 6
178               ''     ''                                                TEMP   1
179               DEFINE LENSTR '' 150 '' $                                BLDTXT 6
180               ITEM FIRST INTG $   '' FIRST NON-BLANK CHARACTER IN STRNG ''  BLDTXT 7
181               ITEM IDUMMY INTG $                                       BLDTXT 8
182               ITEM ITH INTG $                                          BLDTXT 9
183               ITEM RTH INTG $                                          BLDTXT10
184               ITEM MODULE INTG $                                       BLDTXT11
185               ITEM LASTBYT INTG $   '' NUMBER OF CHARACTERS IN LAST BYTE OF  BLDTXT12
186                                       STRNG ''                         BLDTXT13
187               ITEM NCHARS INTG $                                       BLDTXT14
188               ITEM NWORDS INTG $   '' NUMBER OF 4 CHARACTER WORDS IN STRNG ''  BLDTXT15
189               ITEM STRNG H LENSTR $                                    BLDTXT16
190               ITEM TEMP1 INTG $                                        BLDTXT17
191               ITEM INDENT INTG $                                       BLDTXT18
192               ITEM LENTXT INTG $                                       BLDTXT19
193               BEGIN     '' BLDTXT ''                                   BLDTXT20
194                 FOR I = 0, 1, 17 $   '' CLEAR BUFFER ''               BLDTXT21
195                   LTEXT($ I $) = BLNK $                                BLDTXT22
196                 FOR I = 0, 1; LENSTR-1 $   '' SCAN FOR FIRST NON-BLANK CHAR ''  BLDTXT23
197                   BEGIN                                                BLDTXT24
198                     IF BYTE($ I, 1$) (STRNG) NQ BYTE($0,1$) ( BLNK ) $ BLDTXT25
199                       BEGIN                                            BLDTXT26
200                         FIRST = I $                                    BLDTXT27
```

```
ALTER NO      8...5...6...1...8...5...2...9...3...8...5...4...5...5...5...6...6...7...5...8                ANI1

201                         GOTO MOVE $                                                    BLDTXT28
202                       END                                                              BLDTXT29
203                   END                                                                  BLDTXT30
204                 RETURN $   '' BLANK STRING ''                                          BLDTXT31
205               MOVE:                                                                    BLDTXT32
206                 NCHARS = LENSTR - FIRST $                                              BLDTXT33
207                 ''FIND INTGRAL NUMBER OF WORDS TO MOVE''                               BLDTXT34
208                 REMQUO (NCHARS, NBYTWD = NWORDS, LASTBYT) $                            BLDTXT35
209                 LENTXT = NWORDS * 1 $                                                  BLDTXT36
210                 KTH = 0 $                                                              BLDTXT37
211                 JTH = FIRST $                                                          BLDTXT38
212                 FOR I = 1, 1, NWORDS $                                                 BLDTXT39
213                   BEGIN                                                                BLDTXT40
214                     BYTE($0,NBYTWD$) (LTEXT($ KTH $)) =                                BLDTXT41
215                     BYTE($JTH,NBYTWD$) (STRNG) $                                       BLDTXT42
216                     JTH = JTH + NBYTWD $                                               BLDTXT43
217                     KTH = KTH + 1 $                                                    BLDTXT44
218                   END                                                                  BLDTXT45
219                 IF LASTBYT OR 0 $                                                      BLDTXT46
220                   BEGIN                                                                BLDTXT47
221                     BYTE($0,LASTBYT$) (LTEXT($ KTH $)) =                               BLDTXT48
222                     BYTE($JTH,LASTBYT$) (STRNG) $                                      BLDTXT49
223                     LENTXT = LENTXT + 1 $                                              BLDTXT50
224                   END                                                                  BLDTXT51
225               END   '' BLDTXT ''                                                       BLDTXT52
226
227               PROC CASE2 (MODULE, STMT) $                                              CASE2  3
228               ''      JAVS-5  --  MODULE INSTRUMENTATION COMPONENT      ''             0729RU 6
229               ''      ''                                                               TEMP   1
230               '' FINDS THE FIRST ORIF IN AN IFEITH-ORIF-END NEST BY ''                 CASE2  4
231               '' WALKING THE CHAIN OF POINTERS IN THE SDB UNTIL THE ''                 CASE2  5
232               '' IFEITH STATEMENT IS ENCOUNTERED AND THEN RETURNING ''                 CASE2  6
233               '' THE NEXT POINTER ''                                                   CASE2  7
234               DEFINE IFEIV ''4H(IFEI)'' $                                              CASE2  10
235               DEFINE ORIFV ''4H(ORIF)'' $                                              CASE2  11
236               DEFINE ENDV ''4H(END )'' $                                               CASE2  12
237               ITEM BASE2 INTG $       ''1ST ORIF STATEMENT NUMBER''                    CASE2  13
238               ITEM MODULE INTG $      ''MODULE NUMBER''                                CASE2  14
239               ITEM II INTG $          ''CURRENT STATEMENT IN CHAIN''                   CASE2  15
240               ITEM PTR INTG $         ''NEXT STATEMENT IN CHAIN''                      CASE2  16
241               ITEM STMT INTG $        ''CURRENT STATEMENT NUMBER''                     CASE2  17
242               ITEM TYPE MLL $         ''CURRENT CHAIN STATEMENT TYPE''                 CASE2  18
243               BEGIN                                                                    CASE2  19
244                 CASE 2 $ 0 $                                                           CASE2  20
245                 PTR = STMT $                                                           CASE2  21
246               LOOP:   II = PTR $      '' SEARCH FOR IFEITH STATEMENT ''                CASE2  22
247                     PTR = IGTWRD(MODULE, SBBNUM, II, 7) $                            CASE2  23
248                     IF PTR LQ 0 OR PTR GR BSTMTS $                                   CASE2  24
249                       BEGIN                                                          CASE2  25
250                         ERROR (21H(POINTER OUT OF RANGE.)) $                         CASE2  26
```

ALTER NO ANZ1

```
251                       RETURN $                                                    CASE2 37
252                     END                                                           CASE2 38
253                   HTEMP = IQTLIT(MODULE, SUBNUM, PTR; 5) $                        CASE2 39
254                   BYTE(S0,NBYTWD$) (TYPE) = BYTE(S0,NBYTWD$) (HTEMP) $            CASE2 30
255                   IF TYPE EQ ORIFV OR TYPE EQ ENDV $  GOTO LOOP $                 CASE2 31
256                 IF TYPE EQ IFEIV $                                                CASE2 32
257                   BEGIN                                                         CASE2 33
258                     II = PTR $                                                  CASE2 34
259                     PTR = IQTWRD(MODULE, SUBNUM, II, 7) $                       CASE2 35
260                     HTEMP = IQTLIT(MODULE, SUBNUM, PTR, 5) $                    CASE2 36
261                     BYTE(S0,NBYTWD$) (TYPE) = BYTE(S0,NBYTWD$) (HTEMP) $        CASE2 37
262                     IF TYPE EQ ORIFV $                                          CASE2 38
263                       BEGIN                                                     CASE2 39
264                         CASE2 = PTR $                                           CASE2 40
265                         RETURN $   'I NORMAL RETURN !!                          CASE2 49
266                       END                                                       CASE2 50
267                   END                                                           CASE2 51
268                       BYTE(S0,128$) (LINE) = BYTE(S0,128$) (BLANK$) $           CASE2 52
269                       BYTE(S0, 16$) (LINE) = 16H(POINTER IN SUB ) $             CASE2 53
270                 HTEMP = ITSHOL ( II) $                                          CASE2 54
271                 BYTE($16,4$) (LINE) = HTEMP $                                   CASE2 55
272                 BYTE($21,33$) (LINE) = 33H(POINTS TO INVALID STATEMENT TYPE ) $ CASE2 56
273                 BYTE($35,4$) (LINE) = BYTE(S0,4$) ( TYPE) $                     CASE2 57
274                 OUTBUF ( LENLIN, LINE ) $                                       CASE2 58
275                 ERROR ( 32H(INVALID STATEMENT TYPE IN CHAIN.) ) $               CASE2 59
276               END                                                               CASE2 60
277
278               PROC CONCAT ( LFTSTR, RTSTR ) $                                   CONCAT 3
279               !!     JAVS-5  --  MODULE INSTRUMENTATION COMPONENT     !!        0729RU 6
280               !!     !!                                                         TEMP   1
281               !! CONCATENATES RTSTR TO LFTSTR WITH 1 INTERVENING BLANK          CONCAT 4
282                  AND RETURNS THE RIGHTMOST LENSTR CHARACTERS !!                 CONCAT 5
283               DEFINE LENSTR !! 190 !! $                                         CONCAT 8
284               ITEM CONCAT H LENSTR $                                            CONCAT 9
285               ITEM RTSTR H LENSTR $  !! RIGHT SUBSTRING !!                      CONCAT10
286               ITEM LFTSTR H LENSTR $   !! LEFT SUBSTRING !!                     CONCAT11
287               ITEM RTBUFR H LENSTR $ !! RIGHT SUBSTRING BUFFER !!               CONCAT12
288               ITEM LFTBUFR H LENSTR $  !! LEFT SUBSTRING BUFFER !!              CONCAT13
289               ITEM II INTG $  !! FIRST NON-BLANK CHARACTER FROM LEFT            CONCAT14
290                                  IN RIGHT SUBSTRING !!                          CONCAT15
291               ITEM  NCHARS INTG $   !!NUMBER OF CHARACTERS TO MOVE FROM RTSTR!! CONCAT16
292               BEGIN                                                             CONCAT17
293                 RTBUFR = RTSTR $                                                CONCAT18
294                 LFTBUFR = LFTSTR $                                              CONCAT19
295                 CONCAT = BLNK $                                                 CONCAT20
296                 II = 0 $                                                        CONCAT21
297               LOOK: !!FOR FIRST NON-BLANK CHARACTER FROM LEFT IN RIGHT SUBSTRING!! CONCAT22
298                   !!  ASSERTION,  0<=II<LENSTR  !!                            CONCAT23
299                     IF  BYTE(II,1$) (RTBUFR) NQ BYTE(S0,1$) (BLNK)  $         CONCAT24
300                       GOTO FOUND $                                            CONCAT25
```

```
 ALTER NO      ....5....6....1....8....5....2....9....3....4....5....4....5....5....6....5....7....5....8          ANZ1

301                    II = II + 1 $                                                          CONCAT26
302                  IF II LS LENSTR $ GOTO LOOK $                                            CONCAT27
303          FOUND.                                                                           CONCAT28
304                  IFEITH II GR 0 AND II LS LENSTR $                                        CONCAT29
305                    BEGIN                                                                  CONCAT30
306                      NCHARS = LENSTR - II $                                               CONCAT31
307                      BYTE($II,NCHARS$) (CONCAT) = BYTE($II,NCHARS$) (RTBUFR) $            CONCAT32
308                      BYTE($0,II-1$) (CONCAT) = BYTE($NCHARS+1,II-1$) (LFTBUFR) $          CONCAT33
309                    END                                                                    CONCAT34
310                  ORIF II 0 LENSTR $                                                       CONCAT35
311                    CONCAT = LFTBUFR $                                                     CONCAT36
312                  ORIF 1 $                                                                 CONCAT37
313                    CONCAT = RTBUFR $                                                      CONCAT38
314                  END                                                                      CONCAT39
315                END ''CONCAT''                                                             CONCAT40
316
317                PROC EXTSB (FIRST, LAST, LSTART = LENTXT) $                              EXTSB 10
318                ''      JAVS-5  --  MODULE INSTRUMENTATION COMPONENT      ''             0729RU 6
319                ''      ''                                                               TEMP  1
320                ''EXTRACT TEXT FROM STATEMENT BLOCK TO LTEXT ARRAY''                     EXTSB 11
321                DEFINE BUFRLEN '' 72'' $                                                 0710JB 1
322                ITEM BUFFER H 72 $                                                       0722JB31
323                ITEM FIRST INTG $  ''WHERE TO START IN SB''                              EXTSB 16
324                ITEM LAST INTG $  ''WHERE TO END IN SB''                                 EXTSB 17
325                ITEM LSTART INTG $  ''WHERE TO START IN LTEXT''                          EXTSB 18
326                ITEM LENTXT INTG $  ''LENGTH OF LTEXT''                                  EXTSB 19
327                ITEM IFIRST INTG $  ''NEXT STARTING WORD IN SB''                         EXTSB 20
328                ITEM ILSTRT INTG $  ''NEXT WORD TO PUT IN LTEXT''                        EXTSB 21
329                ITEM NEXT INTG $  ''NEXT WORD TO GET FROM SB''                           EXTSB 22
330                ITEM NWORDS INTG $  ''NUMBER OF WORDS GOTTEN FROM SB''                   EXTSB 23
331                BEGIN  ''EXTSB1''                                                        EXTSB 24
332                IFIRST = FIRST $                                                         EXTSB 25
333                ILSTRT = LSTART $                                                        EXTSB 26
334                LENTXT = LSTART $                                                        0710JB 2
335          GET;                                                                           EXTSB 28
336                ''EXTRACT TEXT FROM STATEMENT BLOCK''                                    EXTSB 29
337                BLDLIB (IFIRST, LAST, LLIST, 0, BUFRLEN = NBYT, BUFOUT) $
338                ''DETERMINE LENGTH OF BUFFER''                                           EXTSB 31
339                  BYTE($0,72$) (BUFFER) = BYTE($0,72$) (BUFOUT) $                        0722JB32
340                  BUFFER = JUSTR ( BUFFER ) $                                            0722JB33
341                  NWORDS = (LASTCH(BUFFER) / NBYTWD) + 1 $                               0722JB34
342                ''MOVE TEXT TO LTEXT ARRAY''                                             EXTSB 33
343                FOR I = ILSTRT, 1, NWORDS / ILSTRT $                                     EXTSB 34
344                  FOR J = BUFRLEN - (NWORDS * NBYTWD ), NBYTWD $                         0723JB 1
345                  BYTE($0, NBYTWD$) (LTEXT($ I $) ) = BYTE($J,NBYTWD$) (BUFFER) $        0722JB35
346                LENTXT = LENTXT + NWORDS $                                               EXTSB 37
347                IF NEXT LQ LAST $                                                        EXTSB 38
348                BEGIN                                                                    EXTSB 39
349                  ILSTRT = NWORDS + ILSTRT $                                             EXTSB 40
350                  IFIRST = NEXT $                                                        EXTSB 41
```

```
ALTER NO      0....5....1....2....3....4....5....6....7....8                    ANZ1

351                 GOTO GET $                                                  EXTSB 42
352               END                                                           EXTSB 43
353               END ''EXTSB''                                                 EXTSB 44
354
355               PROC EXTEND (MODULE, TEXT, NCHARS, STRT # LENTXT) $           EXTEND 3
356               ''     JAVS-5  --  MODULE INSTRUMENTATION COMPONENT     ''    0729RU 6
357               ''     ''                                                    TEMP   1
358               ''ADDS TEXT TO THE END OF THE LTEXT ARRAY AND UPDATES LENTXT  EXTEND 4
359                 WHICH IS THE CURRENT LENGTH OF THE TEXT IN THE ARRAY''      EXTEND 5
360               DEFINE MAXWORDS ''37'' $                                      0718JB12
361               DEFINE MAXTEXT '' 72''$                                       0724JB 2
362               ITEM LASTBYT INTG M  ''NUMBER OF CHARACTERS REMAINING AFTER   EXTEND10
363                                      NWORDS HAVE BEEN MOVED''               EXTEND11
364               ITEM LENTXT INTG $   ''LENGTH OF LTEXT ARRAY ON RETURN''      EXTEND12
365               ITEM MODULE INTG $                                            EXTEND13
366               ITEM NCHARS INTG $   ''LENGTH OF INPUT CHARATER STRING''      EXTEND14
367               ITEM NWORDS INTG $   ''INTEGRAL  NUMBER OF WORDS TO MOVE''    0718JB14
368               ITEM STRT INTG $     ''STARTING WORD IN LTEXT''               EXTEND16
369               ITEM TEXT M MAXTEXT $   $''INPUT CHARACTER STRING''           EXTEND17
370               BEGIN   ''EXTEND''                                            EXTEND18
371                 IF NCHARS GR MAXTEXT $                                      0714JB11
372                 BEGIN                                                       EXTEND20
373                   ERROR ( 44H(INPUT STRING TOO LONG--TRUNCATED FROM HEAD.)) $ EXTEND21
374                   NCHARS = MAXTEXT $                                        EXTEND22
375                 END                                                         EXTEND23
376                 ''CALCULATE INTEGRAL NUMBER OF WORDS TO MOVE AND REMAINING  EXTEND24
377                   CHARACTERS''                                              EXTEND25
378                 REMQUO (NCHARS, NBYTWD = NWORDS, LASTBYT) $                 EXTEND26
379                 IF NWORDS GR 0 $                                            0721JB 1
380                 BEGIN                                                       0806JB13
381                 FOR I = STRT, 1, STRT+NWORDS-1 $                            0806JB14
382                 FOR J = MAXTEXT-NCHARS, NBYTWD $                            0806JB15
383                   BYTE($J,NBYTWDS) (LTEXT($ I $)) = BYTE($J,NBYTWDS) (TEXT) $ 0806JB16
384                 END                                                         0806JB17
385                 LENTXT = STRT + NWORDS $                                    EXTEND30
386                 IF LASTBYT GR 0 $                                           EXTEND31
387                 BEGIN                                                       EXTEND32
388                   BYTE($0,NBYTWDS) (LTEXT($ LENTXT $)) = BYTE($0,NBYTWDS)   0724JB 3
389               (BLOK) $                                                      0724JB 4
390                   BYTE($0,LASTBYTS) (LTEXT($ LENTXT $)) = BYTE($MAXTEXT-LASTBYT EXTEND34
391               ,LASTBYTS) (TEXT) $                                           EXTEND35
392                   LENTXT = LENTXT + 1 $                                     0721JB31
393                 END                                                         EXTEND36
394               END ''EXTEND''                                                EXTEND37
395
396               PROC INTVB (MODULE,SYMBOL,SYMLEN)$                            J80724 1
397               ''     JAVS-5  --  MODULE INSTRUMENTATION COMPONENT     ''    0729RU 6
398               ''     ''                                                    TEMP   1
399               ''RETURNS THE INDEX OF SYMBOL IN THE TRACE VARIABLE BLOCK OR 0 IF INTVB  4
400                 THE SYMBOL IS NOT PRESENT''                                 INTVB  5
```

```
ALTER NO  ....5....1....2....3....4....5....6....7....8                    ANZ1

401                ITEM INTVB INTG $                                       INTVB  8
402               ITEM MODULE INTG$                                        0709CG24
403               ITEM SYMLEN INTG$                                        0709CG25
404               ITEM ISYMLEN INTG$                                       J80724 2
405               ITEM SYMBOL H 32$                                        0709CG26
406               ITEM TABSYM H 32$                                        0709CG27
407               ITEM CYMBOL H 32$                                        CG29JUL2
408               ITEM TABCYM H 32$                                        CG29JUL3
409               ITEM IBEG INTG$                                          0709CG28
410                BEGIN                                                   INTVB 10
411               IBEG = 0$ ''INITIALIZE STARTING POINT OF SEARCH TO START OF TABLE''0709CG29
412               ISYMLEN = ((SYMLEN-1)/NBYTWB)+1$                         J80724 3
413               IF LENTVI GR 0$                                          0709CG30
414               BEGIN                                                    0709CG31
415            SEARCHITVI;                                                 0709CG32
416               IBEG = IBEG+1$                                           0709CG33
417               INTVB = ISRTAB (MODULE,TVINUM,IBEG,LENTVI/1,18,YMLEN)$   J80724 4
418               IF INTVB LQ 0$   RETURN$   ''SYMBOL NOT FOUND''          0709CG35
419               CYMBOL = SYMBOL$                                         CG29JUL5
420               GETTVB (MODULE,INTVB,SYMLEN&TABSYM)$                     CG29JUL6
421               TABCYM = TABSYM$                                         CG29JUL7
422               IF BYTE(S32-SYMLEN,SYMLEN$)(SYMBOL) EQ BYTE(S32,SYMLEN,SYMLEN$) 0728JB 3
423               (TABSYM)$                                                0728JB 4
424                  RETURN$   ''INDEX OF SYMBOL IN TABLE''                0709CG38
425               IBEG = INTVB$   ''UPDATE STARTING POINT OF SEARCH''      0709CG39
426               IF IBEG LS LENTVI$                                       0728JB 5
427               GOTO SEARCHITVI$                                         0709CG41
428               END                                                      0709CG42
429                  INTVB = 0 $                                           0728JB 6
430                END ''INTVB''                                           INTVB 11
431
432                PROC JPROBE ( MODULE ) $                              JPROBE 3
433                ''      JAVS-5  --  MODULE INSTRUMENTATION COMPONENT      ''  0729RU 6
434                ''      ''                                            TEMP   1
435                ''REVISION OF APRIL 14, 1995''                        JPROBE 4
436                ITEM LINE M 128 $                                     JPROBE 9
437                ITEM MODULE INTG $                                    JPROBE10
438                  ITEM MTYPE HLL $
439                BEGIN                                                 JPROBE11
440               DECLARED = FALSE$                                      J80724 6
441               DCLASSENT = FALSE $
442               CSTKPTR = -1 $ ''INITIALIZE ASSERTION STACK PTR''
443                  TOP = 0 $                                          JPROBE12
444                  FBSTACK($ 0 $) = 0 $                               JPROBE13
445                  PDDES ($ 0 $) = 0 $                                JPROBE14
446                  LASTIF = 0 $                                       JPROBE15
447                  LASTDOP = 0 $                                      JPROBE16
448                  MXDOP = IGTWRD (MODULE, MDBNUM, 1, 25) $           JPROBE17
449                  MODNAM = MDBMOD ( MODULE ) $                       JPROBE18
450                  TXTNAM = MDBTXT (MODULE) $                         JPROBE19
```

ALTER NO ...5...6...7...8 ANZ1

```
451                 BYTE($0,128$) (LINE) = BYTE($0,128$) (BLANKS) $                    JPROBE20
452                 BYTE($0,61$) (LINE) = 61H(OJOVIAL AUTOMATED VERIFICATION SYSTEM JPROBE21
453           ### INSTRUMENTATION ###) $                                               JPROBE22
454                 OUTBUF ( LENLIN, LINE ) $                                        JPROBE23
455                 BYTE($0,128$) (LINE) = BYTE($0,128$) (BLANKS) $                  JPROBE24
456                 BYTE($0,24$) (LINE) = 24H(OOPTIONS IN EFFECT . . .) $            JPROBE25
457                 OUTBUF (LENLIN, LINE) $                                          JPROBE26
458                 BYTE($0,128$) (LINE) = BYTE($0,128$) (BLANKS) $                  JPROBE27
459                 BYTE($30,17$) (LINE) = 17H( DD-PATH PROBE # ) $                  JPROBE28
460                 BYTE($57,8$) (LINE) = BYTE($0,8$) (VPROBE) $                     JPROBE29
461                 OUTBUF (LENLIN, LINE) $                                          JPROBE30
462                 BYTE($0,128$) (LINE) = BYTE($0,128$) (BLANKS) $                  JPROBE31
463                 BYTE($30,17$) (LINE) = 17H( MODULE  PROBE # ) $                  JPROBE32
464                 BYTE($57,8$) (LINE) = BYTE($0,8$) (VPROM) $                      JPROBE33
465                 OUTBUF (LENLIN, LINE) $                                          JPROBE34
466                 BYTE($0,128$) (LINE) = BYTE($0,128$) (BLANKS) $                  JPROBE35
467                 BYTE($30,17$) (LINE) = 17H( TEST    PROBE # ) $                  JPROBE36
468                 BYTE($57,8$) (LINE) = BYTE($0,8$) (VPROB) $                      JPROBE37
469                 OUTBUF (LENLIN, LINE) $                                          JPROBE38
470                 BYTE($0,128$) (LINE) = BYTE($0,128$) (BLANKS) $                  JPROBE39
471                 BYTE($0,15$) (LINE) = 15H(@INSTRUMENTING ) $                     JPROBE40
472                 IF†H PMODE EQ INVOKE $                                           JPROBE41
473                   BYTE($15,24$) (LINE) = 24H(ENTRY POINTS AND RETURNS) $         JPROBE42
474                 ORIF PMODE EQ PATHS $                                            JPROBE43
475                   BYTE($15,34$) (LINE) =34H(ENTRY POINTS; RETURNS AND DD-PATHS)$ JPROBE44
476                 ORIF PMODE EQ DIRTS $                                            JPROBE45
477                   BYTE($15,10$) (LINE) = 10H(DIRECTIVES) $                       JPROBE46
478                 ORIF PMODE EQ FULL $
479                   BYTE($15,23$) (LINE) = 23H(DD-PATHS AND DIRECTIVES) $
480                 END                                                              JPROBE47
481                 BYTE($49,12$) (LINE) = 12H( OF MODULE #) $                       JPROBE48
482                 BYTE($61,8$) (LINE) = BYTE($0,8$) (MODNAM) $                     JPROBE49
483                 BYTE($69,15$) (LINE) = 15H(> OF JAVSTEXT <) #                    JPROBE50
484                 BYTE($84,8$) (LINE) = BYTE($0,8$) (TXTNAM) $                     JPROBE51
485                 BYTE($92,1$) (LINE) = $H(>) $                                    JPROBE52
486                 OUTBUF (LENLIN, LINE) $                                          JPROBE53
487                 NSTMTS = IGTWRD (MODULE, MODNUM, 1, 10) $                        JPROBE54
488                 FIRSTX = IGTWRD(MODULE, MODNUM, 1, 12) $                         JPROBE55
489                 IF FIRSTX LQ 0 $                                                 JPROBE56
490                   BEGIN                                                          JPROBE57
491                     BYTE($0,128$) (LINE) = BYTE($0,128$) (BLANKS) $              JPROBE58
492                     BYTE($0, 47$) (LINE) = 47H(OTHERE ARE NO EXECUTABLE STATEMENJPROBE59
493           TS IN MODULE <) $                                                      JPROBE60
494                     BYTE($47,8$)  (LINE) = BYTE($0,8$) (MODNAM) $                JPROBE61
495                     BYTE($55,15$) ( LINE) = 15H(> OF JAVSTEXT <) $               JPROBE62
496                     BYTE($70, 8$) (LINE) = BYTE($0,8$) (TXTNAM) $                JPROBE63
497                     BYTE($78, 2$) (LINE) = 2H(>.) $                              JPROBE64
498                     OUTBUF (LENLIN, LINE) $                                      JPROBE65
499                     RETURN $                                                     JPROBE66
500                   END                                                            JPROBE67
```

```
ALTER NO      ....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8      ANZ1
501                  ''FIND THE LAST EXECUTABLE STATEMENT IN THE MODULE ASSUMING    ''JPROBE48
502                ''  THAT IT IS NOT A DECLARATION INCLUDING AN END               ''JPROBE49
503                  HTEMP = IGTLIT(MODULE, SDBNUM, NSTMTS, 5) $                    JPROBE70
504                  BYTE($0,NBYTWDS) (STYPE) = BYTE($0,NBYTWDS) (HTEMP) $          JPROBE71
505                  HTEMP = IGTLIT ( MODULE, MDBNUM, 1, 3 ) $
506                  BYTE($0,NBYTWDS) (MTYPE) = BYTE($0,NBYTWDS) (HTEMP) $
507                  IFEITH STYPE EQ ENDV $                                         JPROBE72
508                    LASTX = NSTMTS $                                             JPROBE73
509                  ORIF (STYPE EQ TERMV) AND (MTYPE EQ #HIPROG)) $
510                    LASTX = NSTMTS $
511                  ORIF   STYPE EQ TERMV $                                        JPROBE74
512                    BEGIN                                                        JPROBE75
513                    FOR I = NSTMTS-1, -1, 0 $
514                      BEGIN
515                      HTEMP = IGTLIT ( MODULE, SDBNUM, I, 3 ) $
516                      BYTE($0,NBYTWDS) (STYPE) = BYTE($0,NBYTWDS) (HTEMP) $
517                      IF STYPE EQ ENDV $
518                        BEGIN
519                        LASTX = I $
520                        GOTO FOUND $
521                        END
522                      END
523                    ERROR ( 35H(NO EXECUTABLE STATEMENTS IN MODULE;) ) $
524                    RETURN $
525              FOUND:
526                    END                                                          JPROBE84
527                  ORIF 1 $                                                       JPROBE85
528                    BEGIN                                                        JPROBE86
529                      ERROR (44H(LAST STATEMENT IN MODULE IS NOT TERM OR END.)) $ JPROBE87
530                      RETURN $                                                   JPROBE88
531                    END                                                          JPROBE89
532                  END                                                            JPROBE90
533                  BLOCK = 0 $                                                    JPROBE91
534                  CFOR = 0 $                                                     JPROBE92
535                  BYTE($0,8$) (TESTPRB) = BYTE($0,8$) (VARBL) $                  JPROBE93
536                IFEITH (PMODE EQ PATHS) OR (PMODE EQ DIRTS) OR (PMODE EQ FULL) $ 0604JB68
537                    PROBJ3 (MODULE) $                                            JPROBE95
538                  ORIF   PMODE EQ INVORE $                                       JPROBE96
539                    PARENT ( MODULE) $                                           JPROBE97
540                  ORIF 1 $              '1MODE NOT IMPLEMENTED!'                 JPROBE98
541                    BEGIN                                                        JPROBE99
542                      BYTE($0,128$) (LINE) = BYTE($0,128$) (BLANKS) $            JPROB100
543                      BYTE($0, 23$) (LINE) = 23H(INSTRUMENTATION MODE  ) $       JPROB101
544                      BYTE($23, 4$) (LINE) = BYTE($0,4$) (PMODE) $               JPROB102
545                      BYTE($27,22$) (LINE) = 22H(  NOT YET IMPLEMENTED.) $       JPROB103
546                      OUTBUF (LENLIN, LINE) $                                    JPROB104
547                    BLOCK = 1 $                                                  JPROB105
548                    END                                                          JPROB106
549                  END                                                            JPROB107
550                  ''STORE TOTAL NUMBER OF PROBE STATEMENT BLOCKS''               JPROB108
```

```
ALTER NO      ....5....6....1....8....5....2....3....4....5....6....7....8....5....8          ANZ1

551                 IF BLOCK GR 0$                                                    0730JB 2
552                 PUTBRD (MODULE, MDBNUM, 1, 24, BLOCK-1) $                         JPROB109
553               END                                                                 JPROB110
554
555               PROC JUSTRT ( STRNG) $                                              JUSTRT 3
556               !!     JAVS-5  --  MODULE INSTRUMENTATION COMPONENT     !!         0729RU 6
557               !!     !!                                                          TEMP   1
558               !!RIGHT JUSTIFY A LEFT JUSTIFIED CHARACTER STRING WITH LEADING     JUSTRT 4
559                 BLANKS!!                                                          JUSTRT 5
560               DEFINE LENSTR !! 150 !! $                                           JUSTRT 6
561               ITEM II INTG $                                                      JUSTRT 9
562               ITEM JUSTRT H LENSTR $                                              JUSTRT10
563               ITEM STRNG H LENSTR $                                               JUSTRT11
564               BEGIN   !!JUSTRT!!                                                  JUSTRT12
565                 !!CLEAR RETURN STRING!!                                           JUSTRT13
566                 JUSTRT = BLNK $                                                   JUSTRT14
567                 !!SEARCH FOR FIRST NON-BLANK CHARACTER FROM RIGHT TO LEFT IN       JUSTRT15
568                   STRING!!                                                        JUSTRT16
569                 II = LENSTR $                                                     JUSTRT17
570           LOOP!                                                                   JUSTRT18
571                   II = II - 1 $                                                   JUSTRT19
572                 IF BYTE(SII,1S) (STRNG) NQ BYTE(80,1S) (BLNK) $ GOTO MOVE $       JUSTRT20
573                 IF II GR 0 $  GOTO LOOP $                                         JUSTRT21
574                 RETURN$  !!ALL BLANKS!!                                           JUSTRT22
575           MOVE!   !!CHARACTERS FROM INPUT TO OUTPUT STRING!!                      JUSTRT23
576                 BYTE(SLENSTR-II-1,II+1S) (JUSTRT) = BYTE(S0,II+1S) (STRNG) $      JUSTRT24
577               END  !!JUSTRT!!                                                     JUSTRT25
578
579               PROC LASTCH ( BUFFER ) $                                            LASTCH 3
580               !!     JAVS-5  --  MODULE INSTRUMENTATION COMPONENT     !!         0729RU 6
581               !!     !!                                                          TEMP   1
582               !!*** ORIGINAL SPECIFICATION ***!!                                 0722JB11
583               !!INTEGER FUNCTION -- RETURNS THE BYTE NUMBER OF THE LAST !!       LASTCH 4
584               !!NON-BLANK CHARACTER IN THE INPUT BUFFER     !!                   LASTCH 5
585               !!*** CHANGED SPECIFICATION *** !!                                0722JB12
586               !!RETURNS THE LENGTH OF THE INPUT STRING BY SCANNING FROM LEFT TO  0722JB13
587                 RIGHT AND IGNORING PRECEDING BLANKS!!                           0722JB14
588             !!WARNING --- LASTCH ASSUMES THAT THE INPUT CHARACTER                0722JB15
589                           STRING IS RIGHT JUSTIFIED !!                          0722JB16
590               ITEM LASTCH INTG $                                                LASTCH 8
591               DEFINE MXBUF !!150!!$                                             0722JB17
592               ITEM BUFFER H MXBUF$                                              0722JB18
593               BEGIN                                                             LASTCH10
594                 FOR I = 0, 1; MXBUF-1 $                                         0722JB19
595                   BEGIN !!SEARCHING FOR FIRST NON-BLANK CHARACTER FROM LEFT!!   0722JB10
596                     IF BYTE(SI,1S) (BUFFER) NQ BYTE(S0,1S) (BLNK) $             LASTCH13
597                       BEGIN                                                     LASTCH14
598                         LASTCH = MXBUF - I $                                    0722JB11
599                         RETURN $                                                LASTCH16
600                       END                                                       LASTCH17
```

```
ALTER NO      ....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8      ANZ1

601                   END                                                                    LASTCH18
602                 LASTCH # 0 $                                                             LASTCH19
603               END ''LASTCH''                                                             LASTCH20
604
605               PROC MAKE (MODULE, STRNG , NCHARS, INDENT # LENTXT) $                      MAKE   3
606               !!     JAVS-5  --  MODULE INSTRUMENTATION COMPONENT      !!                0729RU 6
607               !!     !!                                                                 TEMP   1
608               DEFINE MXSTRG !!148!! $ !! MAXIMUM LENGTH OF STRING !!                    0806JB18
609               ITEM INDENT INTG $      !! NUMBER OF CHARACTERS TO INDENT !!              MAKE   7
610               ITEM JJ INTG $          !! CHARACTER INDEX !!                             MAKE   8
611               ITEM KK INTG $          !! WORD INDEX !!                                  MAKE   9
612               ITEM LASTBYT INTG $     !! NUMBER OF CHARACTERS IN LAST WORD OF           MAKE  10
613                                       LTEXT ARRAY !!                                    MAKE  11
614               ITEM LNSTRG INTG $      !! TOTAL STRING LENGTH INCLUDING INDENT !!        MAKE  12
615               ITEM LENTXT INTG $      !! LENGTH OF OUTPUT ARRAY LTEXT !!                MAKE  13
616               ITEM MODULE INTG $      !! MODULE NUMBER !!                               MAKE  14
617               ITEM NCHARS INTG $      !! NUMBER OF CHARACTERS IN INPUT STRING !!        MAKE  15
618               ITEM NWORDS INTG $      !! NUMBER OF FULLY PACKED WORDS IN STRING !!      MAKE  16
619               ITEM STRNG H MXSTRG $   !! INPUT CHARACTER STRING !!                      0806JB19
620               BEGIN                                                                     MAKE  18
621                 !!....CAUTION--MAKE EXPECTS THE CHARACTER STRING IN STRNG              0718JB 1
622                                 TO BE RIGHT JUSTIFIED !!                                0718JB 2
623                 !!                                                  !!                  0718JB 3
624                 LNSTRG # NCHARS # INDENT $                                            MAKE  19
625                 IF LNSTRG GR MXSTRG $                                                 MAKE  20
626                   BEGIN                                                               MAKE  21
627                     ERROR (44H(INPUT STRING TOO LONG--TRUNCATED FROM HEAD.)) $         MAKE  22
628                     LNSTRG = MXSTRG $                                                 MAKE  23
629                   END                                                                 MAKE  24
630                 FOR I = 0,1, (LNSTRG -1) / NBYTWD $                                   0806JB30
631                   LTEXT(I I 1) = BLNK $                                               MAKE  26
632                 REMOVE (LNSTRG, NBYTWD = NWORDS, LASTBYT) $                           MAKE  27
633                 JJ # MXSTRG - LNSTRG $                                                MAKE  28
634                 KK # 0 $                                                              MAKE  29
635               LOOP:  IF KK EQ NWORDS # GOTO LAST $                                    MAKE  30
636                   BYTE(S0,NBYTWDS) (LTEXT(S KK S1) =                                  MAKE  31
637                   BYTE(SJJ/NBYTWBS) (STRNG ) $                                        MAKE  32
638                   KK = KK + 1 $                                                       MAKE  33
639                   JJ = JJ + NBYTWD $                                                  MAKE  34
640                 GOTO LOOP $                                                           MAKE  35
641               LAST.                                                                   0715JB 1
642                 LENTXT # KK $                                                         0715JB 2
643                 IF LASTBYT GR 0 $                                                     0715JB 3
644                 BEGIN                                                                 0715JB 4
645                   BYTE(S0,LASTBYTS) (LTEXT(SKKS)) # BYTE(SJJ/LASTBYTS) (STRNG) $      0715JB 5
646                   LENTXT = LENTXT + 1 $                                               0715JB 6
647                 END                                                                   0715JB 7
648               END                                                                     MAKE  39
649
650               PROC PRBAST (MODULE, STMT) $                                            PRBAST 3
```

```
ALTER NO  ....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8   ANZ1
  651          '!     JAVS-5  --  MODULE INSTRUMENTATION COMPONENT     !!          0729RU  6
  652          '!     !!                                                             TEMP    1
  653          !!PROBE ASSERT DIRECTIVE!!                                           PRBAST  4
  654          DEFINE DELI !!4H(DELI)!!  $
  655          DEFINE KEYW !!4H(KEYW)!!  $
  656          ITEM ASSLEN INTG $                                                   PRBAST  8
  657          ITEM ASSTXT H 72$                                                    J8072410
  658          ITEM FROM INTG $                                                     PRBAST  9
  659          ITEM MODULE INTG $                                                   PRBAST10
  660          ITEM STMT INTG $                                                     PRBAST11
  661          ITEM TXTPTR INTG $                                                   PRBAST12
  662          BEGIN                                                                PRBAST13
  663             PROCL $   !!J.L; RAMEY, 4/1/78!!
  664          FROM = SCANB (3,LENGTH+1,COMMA|DELI)$                                J8072411
  665             BLDLIN(FROM+2, LENGTH-5, LIST, 0, 72 , INEXT, BUFOUT) $           PRBAST23
  666          BYTE (80,72) (ASSTXT) = BYTE (80,72) (BUFOUT)$                       J8072412
  667          ASSTXT = JUSTRT (ASSTXT)$                                            J8072413
  668          ASSLEN = LASTCH (ASSTXT)$                                            J8072414
  669             TXTPTR = 0 $                                                      PRBAST26
  670             BYTE(80,MAXTXTS) (TXTBUF) = BYTE(80,MAXTXTS) (BLANKS) $           0728JB  8
  671          ADDTXT (8H(IF NOT (),8,TXTPTR,TXTPTR)$                               J8072415
  672          ADDTXT (ASSTXT,ASSLEN,TXTPTR,TXTPTR)$                                J8072416
  673          ADDTXT(3H() $),3,TXTPTR,TXTPTR)$                                     0728JB  9
  674             TXTBUF = JUSTRT ( TXTBUF ) $                                      0721JB11
  675             MAKE (MODULE, TXTBUF, TXTPTR, 0 , LENTXT) $                       PRBAST33
  676             STORE (MODULE, DIRTY, LENTXT, LTEXT) $                            PRBAST34
  677          TXTPTR = 0$                                                          0709CG99
  678          BYTE(80,MAXTXTS) (TXTBUF) = BYTE(80,MAXTXTS) (BLANKS)$               0709CG80
  679          ADDTXT (14H(JAVS'ASSERT = )$14,TXTPTR,TXTPTR)$                       J8072416
  680             ADDTXT ( ITSWOL(ASSLEN+24), NBYTWD, TXTPTR = TXTPTR) $            0728JB10
  681          ADDTXT (2H(W(),2,TXTPTR,TXTPTR)$                                     J8072420
  682          ADDTXT (ASSTXT,ASSLEN,TXTPTR,TXTPTR)$                                J8072421
  683          ADDTXT (10H( AT STMT, )$10,TXTPTR,TXTPTR)$                           J8072422
  684          ADDTXT (ITSWOL(STMT),NBYTWD,TXTPTR,TXTPTR)$                          J8072423
  685          ADDTXT (13H( NOT TRUE.) $)$13,TXTPTR=TXTPTR)$                        0725JB  9
  686             TXTBUF = JUSTRT ( TXTBUF ) $                                      0721JB12
  687          MAKE (MODULE,TXTBUF,TXTPTR,0,LENTXT)$                                0709CG79
  688          STOPRB (MODULE,DIRTY,LENTXT,LTEXT)$                                  0709CG80
  689           END !!PRBAST!!                                                      PRBAST35
  690
  691       PROC PROCHAIN (MODULE, STMT) $
  692       !!     JAVS-5  --  MODULE INSTRUMENTATION COMPONENT!!
  693       !!     !!
  694       !!PROBE CHAIN DIRECTIVE  --  J.L. RAMEY, 2/24/78!!
  695
  696       DEFINE IDENT !!4W(IDEN)!! $
  697       DEFINE DELI  !!4W(DELI)!! $
  698       DEFINE KEYW  !!4W(KEYW)!! $
  699       DEFINE RANGE !!4W(:  )!! $
  700       DEFINE LPAREN !!4H((   )!! $
```

```
ALTER NO      ....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8    ANZI
701           DEFINE MAXCSTK ''10'' $  ''SIZE OF ASSERTION COMMAND STACK''
702           ITEM ASSLEN INTG $
703           ITEM ASSTXT H 72 $
704           ITEM FROM INTG $
705           ITEM KEYWRD INTG $
706           ITEM MIDL INTG $
707           ITEM MODULE INTG $
708           ITEM STMT INTG $
709           ITEM TO INTG $
710           ITEM TMPTXT H 72 $
711           ITEM TXTPTR INTG $
712
713           BEGIN
714              ''SET UP MONITOR FOR ASSERTION VIOLATION OUTPUT''
715              PROBOL $
716
717              ''INCREMENT AND CHECK COMMAND STACK PTR''
718              CSTKPTR = CSTKPTR + 1 $
719              IF CSTKPTR GQ MAXCSTK $
720                 BEGIN
721                    ERROR (33H(ASSERTION COMMAND STACK OVERFLOW?)) $
722                    RETURN $
723                 END
724
725              ''CHECK FOR AND HANDLE EMPTY CLAUSE''
726              KEYWRD = SCANSB (3, LENGTH-5, 4H(EMPT), IDENT) $
727              IF (KEYWRD NQ -1) AND (LLIST(SO,KEYWRD+1S) EQ 4H(T   )) $
728                 BEGIN ''FOUND AN EMPTY CLAUSE''
729                 ''ISOLATE BOOLEAN EXPRESSION''
730                    FROM = SCANSB (KEYWRD$2, LENGTH$1, LPARENV DELI) $
731                    TO = BALPAR (FROM+1, LENGTH, LLIST) $
732                    BLOLIN (FROM+2, TO-1, LOIST, 0, 72 , INEXT, BUFOUT) $
733                    BYTE (S0,72S) (ASSTXT) = BYTE (S0,72S) (BUFOUT) $
734                    ASSTXT = JUSTRT (ASSTXT) $
735
736                 ''BUILD IF NOT STMT''
737                    TXTPTR = 0 $
738                    BYTE (S0,MAXTXTS) (TXTBUF) = BYTE (S0,MAXTXTS) (BLANKS) $
739                    ADDTXT (8H(IF NOT (), 8, TXTPTR = TXTPTR) $
740                    ASSLEN = LASTCH (ASSTXT) $
741                    ADDTXT (ASSTXT, ASSLEN, TXTPTR , TXTPTR) $
742                    ADDTXT (3H() $), 3, TXTPTR = TXTPTR) $
743                    TXTBUF = JUSTRT (TXTBUF) $
744                    MAKE (MODULE, TXTBUF, TXTPTR, 0 = LENTXT) $
745                    STOPRB (MODULE, DIRTY, LENTXT, LTEXT) $
746                 END
747
748              ''GENERATE BEGIN STMT''
749              MAKE (MODULE, 5H(BEGIN), 5, 0 = LENTXT) $
750              STOPRB (MODULE, DIRTY, LENTXT, LTEXT) $
```

```
ALTER NO      ....5....1....2....3....4....5....6....7....8                ANZ1
751
752             '(CHECK FOR AND HANDLE COUNT CLAUSE)'
753             KEYWRD = SCANSB (3, LENGTH-1, 4H(COUN), IDENT) $
754             IF(ITH (KEYWRD NQ -1) AND (ULIST(!0,KEYWRD+1$) BQ 4H(!   )) $
755                BEGIN ''FOUND A COUNT CLAUSE!'
756                ''IF FIRST COUNT CLAUSE, DECLARE ASSCNT!''
757                   IF NOT DCLASSCNT $
758                      BEGIN
759                         MAKE (MODULE, 20H(ITEM ASSCNT I 24 $ $), 20, 0 ! LENTXT)$
760                         STOPRB (MODULE, DIRTY, LENTXT, LTEXT) $
761                         DCLASSCNT = TRUE $
762                      END
763
764                ''BUILD COUNT INIT STMT!!
765                   MAKE (MODULE, 12H(ASSCNT ! 0 $); 12, 0 ! LENTXT) $
766                   STOPRB (MODULE, DIRTY, LENTXT, LTEXT) $
767
768                ''EXTRACT RANGE AND SAVE IN STACK!!
769                   FROM = SCANSB (KEYWRD+2, LENGTH-1, LPAREN, DELI) $
770                   TO = BALPAR (FROM+1, LENGTH, ULIST) $
771                   MIDL = SCANSB (FROM+1, TO+1, RANGE, DELI)  $
772
773                   ''MINIMUM COUNT!!
774                   BLDLIN (FROM+2, MIDL-1, LLIST, 0, 72 ! INEXT, BUFOUT) $
775                   BYTE ($0,72$) (ASSTXT) ! BYTE ($0,72$) (BUFOUT) $
776                   TMPTXT = JUSTRT (ASSTXT) $
777                   ASSLEN = LASTCH (TMPTXT) $
778                   BYTE ($0,ASSLEN$) (MINCNT ($CSTKPTR$)) ! BYTE ($0,ASSLEN$)
779                      (ASSTXT) $
780                   LENMIN ($CSTKPTR$) = ASSLEN $
781
782                   ''MAXIMUM COUNT!!
783                   BLDLIN (MIDL+2, TO-1, LLIST, 0, 72 ! INEXT, BUFOUT) $
784                   BYTE ($0,72$) (ASSTXT) ! BYTE ($0,72$) (BUFOUT) $
785                   TMPTXT = JUSTRT (ASSTXT) $
786                   ASSLEN = LASTCH (TMPTXT) $
787                   BYTE ($0,ASSLEN$) (MAXCNT ($CSTKPTR$)) = BYTE ($0,ASSLEN$)
788                      (ASSTXT) $
789                   LENMAX ($CSTKPTR$) = ASSLEN $
790                END
791             ORIF 1 $ ''NO COUNT CLAUSE!!
792                LENMIN ($CSTKPTR$) = 0 $
793             END  ''(EITHER)''
794
795             '(GET INDEX VARIABLE NAME - SAVE IN COMMAND STACK)'
796             FROM = SCANSB (3, LENGTH-1, LPAREN, DELI) $
797             TO = BALPAR (FROM+1, LENGTH, ULIST) $
798             BLDLIN (FROM+2, TO-1, LLIST, 0, 72 ! INEXT, BUFOUT) $
799             BYTE ($0,72$) (ASSTXT) = BYTE ($0,72$) (BUFOUT) $
800             TMPTXT = JUSTRT (ASSTXT) $
```

ALTER NO .0...5...0...1...0...5...0...2...0...5...0...3...0...5...0...4...0...5...0...5...0...5...0...6...0...5...0...7...0...5...0...8 ANZ1

```
801               ASSLEN = LASTCH (TMPTXT) $
802               BYTE (SO,ASSLENS) (VARNAME(SCSTKPTRS)) = BYTE (SO,ASSLENS) (ASSTXT) $
803               LENNAME (SCSTKPTRS) = ASSLEN $
804
805               '(START BUILDING INDEX INIT STMT)'
806               TXTPTR = 0 $
807               BYTE (SO,MAXTXTS) (TXTBUF) = BYTE (SO,MAXTXTS) (BLANKS) $
808               ASSTXT = JUSTR (ASSTXT) $
809               ADDTXT (ASSTXT, ASSLEN, TXTPTR = TXTPTR) $
810               ADDTXT (8H( = ), 3, TXTPTR = TXTPTR) $
811
812               '(GET INITIAL VALUE - ADD IT TO STMT)'
813               KEYWRD = SCANSB (3, LENGTH-5, 4H(INIT), IDENT) $
814               IFEITH (KEYWRD NQ -1) AND (LLIST(SO,KEYWRD+1S) EQ 4H(    )) $
815                  BEGIN ''FOUND INIT KEYWRD'$
816                     FROM = SCANSB (KEYWRD+2, LENGTH-1, LPAREN, DELI) $
817                     TO = BALPAR (FROM+1, LENGTH, LLIST) $
818                     BLOLIN (FROM+2, TO-1, LLIST, 0, 72 - LNEXT, BUFOUT) $
819                     BYTE (SO,72S) (ASSTXT) = BYTE (SO,72S) (BUFOUT) $
820                     ASSTXT = JUSTR (ASSTXT) $
821                     ASSLEN = LASTCH (ASSTXT) $
822                     ADDTXT (ASSTXT, ASSLEN, TXTPTR = TXTPTR ) $
823                     ADDTXT (2H( $), 2, TXTPTR = TXTPTR) $
824                     TXTBUF = JUSTR (TXTBUF) $
825                     MAKE (MODULE, TXTBUF, TXTPTR, 0 = LENTXT) $
826                     STOPRB (MODULE, DIRTY, HEATXT, LTEXT) $
827                  END
828               OTHERWISE $
829                  BEGIN ''NO INIT FIELD'I
830                     ERROR (38H(CHAIN ASSERTION ERROR - NO INIT FIELD,)) $
831                     RETURN $
832                  END
833               END ''(EITHER)''
834
835               '(GENERATE LABEL ; SAVE IT IN COMMAND STACK)'
836               PUTLBL ($TMT, 1H(8) = LABEL (SCSTKPTRS)) $
837               LENLABL (SCSTKPTRS) = 6 $
838
839               ''(GENERATE LABEL STMT)''
840               BYTE (SO,MAXTXTS) (TXTBUF) = BYTE (SO,MAXTXTS) (BLANKS) $
841               ASSLEN = LENLABL (SCSTKPTRS) $
842               BYTE (SO,ASSLENS) (TXTBUF) = BYTE (SO,ASSLENS) (LABEL(SCSTKPTRS)) $
843               BYTE (SASSLEN+1S) (TXTBUF) = 1H(,) $
844               TXTBUF = JUSTR (TXTBUF) $
845               MAKE (MODULE, TXTBUF, ASSLEN+1, 0 = LENTXT) $
846               STOPRB (MODULE, DIRTY, LENTXT, LTEXT) $
847
848               '(CHECK FOR AND HANDLE BOUNDS CLAUSE)'
849               KEYWRD = SCANSB (3, LENGTH-5, 4H(BOUN), IDENT) $
850               IFEITH (KEYWRD NQ -1) AND (LLIST(SO,KEYWRD+1S) EQ 4H(DS  )) $
```

```
ALTER NO  ....5....1....5....2....5....3....5....4....5....5....6....5....7....5....8          ANZ1
851                 BEGIN ''FOUND A BOUNDS CLAUSE''
852                 ''ISOLATE LOWER AND UPPER BOUNDS''
853                    FROM = $CANSB (KEYWRD+2, LENGTH-1, LPAREN, DELI) $
854                    TO = BALPAR (FROM+1, LENGTH, LLIST) $
855                    MIDL = $CANSB (FROM+1, TO-1, RANGE, DELI) $
856
857                 ''BUILD APPROPRIATE IF STMT'';
858                    TXTPTR = 0 $
859                    BYTE ($0,MAXTXT$) (TXTBUF) = BYTE ($0,MAXTXT$) (BLANK$) $
860                    ADDTXT (8H(IF NOT (), 8, TXTPTR = TXTPTR) $
861                    BLDLIN (FROM+2, MIDL, LUIST, 0, 72 & (NEXT, BUFOUT) $
862                    BYTE ($0,72$) (ASSTXT) = BYTE ($0,72$) (BUFOUT) $
863                    ASSTXT = JUSTRT (ASSTXT) $
864                    ASSLEN = LASTCH(ASSTXT) $
865                    ADDTXT (ASSTXT, ASSLEN, TXTPTR = TXTPTR) $
866                    ADDTXT (4H( LO ), 4, TXTPTR = TXTPTR) $
867                    ADDTXT (BYTE($0,LENNAME($CSTKPTR$)$) (VARNAME($CSTKPTR$))',
868                       LENNAME($CSTKPTR$), TXTPTR = TXTPTR) $
869                    ADDTXT (4H( LO ), 4, TXTPTR = TXTPTR) $
870                    BLDLIN (MIDL+2, TO-1, LUIST, 0, 72 & (NEXT, BUFOUT) $
871                    BYTE ($0,72$) (ASSTXT) = BYTE ($0,72$) (BUFOUT) $
872                    ASSTXT = JUSTRT (ASSTXT) $
873                    ASSLEN = LASTCH (ASSTXT) $
874                    ADDTXT (ASSTXT, ASSLEN, TXTPTR = TXTPTR) $
875                    ADDTXT (3H() $), 3, TXTPTR = TXTPTR) $
876                    TXTBUF = JUSTRT (TXTBUF) $
877                    MAKE (MODULE, TXTBUF, TXTPTR, 0 = LENTXT) $
878                    STOPRB (MODULE, DIRTV, DENTXT, LTEXT) $
879
880                 ''BUILD BEGIN STMT'';
881                    MAKE (MODULE, 5R(BEGIN)', 5, 0 = LENTXT) $
882                    STOPRB (MODULE, DIRTV, DENTXT, LTEXT) $
883
884                 ''GENERATE STMT TO PRINT ERROR MESSAGE'';
885                    TXTPTR = 0 $
886                    BYTE ($0,MAXTXT$) (TXTBUF) = BYTE ($0,MAXTXT$) (BLANK$) $
887                    ADDTXT (49H(JAVS'ASSERT = 35H(BOUNDS ASSERTION ERROR AT STMT ),
888                       49, TXTPTR = TXTPTR) $
889                    ADDTXT (ITSHOL(STMT), NBYTHD, TXTPTR = TXTPTR) $
890                    ADDTXT (3H() $), 3, TXTPTR = TXTPTR) $
891                    TXTBUF = JUSTRT (TXTBUF) $
892                    MAKE (MODULE, TXTBUF, TXTPTR, 0 = LENTXT) $
893                    STOPRB (MODULE, DIRTV, DENTXT, LTEXT) $
894
895                 ''GENERATE AND SAVE LABEL FOR END OF LOOP'';
896                    PRELBL (STMT, 1H(I) = EQUABEL ($CSTKPTR$) $
897                    LENELBL ($CSTKPTR$) = 6 $
898
899                 ''GENERATE GOTO ELABEL STMT'';
900                    BYTE ($0,MAXTXT$) (TXTBUF) = BYTE ($0,MAXTXT$) (BLANK$) $
```

```
ALTER NO  ....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8  ANZ1

                    TXTPTR = 0 $
                    ADDTXT (5H(GOTO ), 5, TXTPTR, TXTPTR) $
                    ADDTXT (BYTE($0,LENELBL(SCSTKPTRS)$) (ELABEL(SCSTKPTRS)),
                       LENELBL(SCSTKPTRS), TXTPTR, TXTPTR) $
                    ADDTXT (2H( $), 2, TXTPTR, TXTPTR) $
                    TXTBUF = JUSTRT (TXTBUF) $
                    MAKE (MODULE, TXTBUF, TXTPTR, 0, LENTXT) $
                    STOPRB (MODULE, DIRTV, DENTXT, LTEXT) $

                 ''GENERATE END STMT''
                    MAKE (MODULE, 3H(END), $, 0, LENTXT) $
                    STOPRB (MODULE, DIRTV, DENTXT, LTEXT) $

                 END
              ORIF 1 $ ''NO BOUNDS CLAUSE''
                 LENELBL(SCSTKPTRS) = 0 $
              END

              ''IF COUNT CLAUSE, BUILD STAT TO INCR COUNT VARIABLE''
              IF LENMIN (SCSTKPTRS) NQ 0 $
                 BEGIN ''COUNT CLAUSE''
                    MAKE (MODULE, 2&H(ASSCNT = ASSCNT + 1 $), 2, 0, LENTXT) $
                    STOPRB (MODULE, DIRTV, DENTXT, LTEXT) $
                 END

              ''ISOLATE NEXT FIELD AND SAVE IT IN COMMAND STACK''
              KEYWRD = SCANSB (3, LENGTH-1, 4H(NEXT), IDER) $
              IFEITH (KEYWRD NQ -1) AND (LLIST(SO:KEYWRD+1$) EQ 4H(    )) $
                 BEGIN ''FOUND NEXT FIELD''
                    FROM = SCANSB (KEYWRD+2, LENGTH-1, LPAREN! DEL!) $
                    TO = BALPAR (FROM+1, LENGTH, LLIST) $
                    BLDLIN (FROM+2, TO-1, LLIST, 0, 72 & $NEXT, BUFOUT) $
                    BYTE ($0,72$) (ASSTXT) = BYTE ($0,72$) (BUFOUT) $
                    THRTXT = JUSTRT (ASSTXT) $
                    ASSLEN = LASTCH (THRTXT) $
                    BYTE ($0,ASSLEN$) (NEXTVAL($CSTKPTRS)) = BYTE ($0,ASSLEN$)
                       (ASSTXT) $
                    LENNEXT (SCSTKPTRS) = ASSLEN $
                 END
              ORIF 1 $ ''NO NEXT FIELD FOUND''
                 BEGIN
                    ERROR (38H(CHAIN ASSERTION ERROR - NO NEXT FIELD.)) $
                    RETURN $
                 END
              END

              ''ISOLATE END FIELD AND SAVE IT IN COMMAND STACK''
              KEYWRD = SCANSB (3, LENGTH-1, 4H(END ), KEYR) $
              IFEITH KEYWRD NQ -1 $
                 BEGIN ''FOUND END FIELD''
```

```
 ALTER NO      ....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8          ANZ1

951                FROM = SCAN9B (KEYWRD$1$ LENGTH=1, LPAREN$ DELI? $
952                TO = BALPAR (FROM+1, LENGTH, LLIST) $
953                BLDLIN (FROM+2, TO-1, LLIST, 0, 72 & INEXT, BUFOUT) $
954                BYTE ($0,72$) (ASSTXT) = BYTE ($0,72$) (BUFOUT) $
955                THRTXT = JUSTRT (ASSTXT) $
956                ASSLEN = LASTCH (TMPTXT) $
957                BYTE ($0,ASSLEN$) (EXIT($CSTKPTRS)) = BYTE ($0,ASSLEN$)
958                   (ASSTXT) $
959                LENEXIT ($CSTKPTRS) = ASSLEN $
960             END
961          OTIF $ $ ''NO END FIELD FOUND$''
962             BEGIN
963                ERROR (37H(CHAIN ASSERTION ERROR - NO END FIELD;)) $
964                RETURN $
965             END
966          END
967       END ''PRBCHAIN''
968
969       PROC PRBENDB (MODULE; STMT) $
970       ''     JAVS$5  --  MODULE INSTRUMENTATION COMPONENT''
971       ''     ''
972       ''PROBE ENDCHAIN AND ENDLOOP DIRECTIVES  --  J.L.RAMSEY, 8/6/78''
973
974       ITEM ASSLEN INTG $
975       ITEM ASSTXT H 72 $
976       ITEM MODULE INTG $
977       ITEM STMT INTG $
978       ITEM TXTPTR INTG $
979
980       BEGIN
981          ''CHECK TO MAKE SURE STACK IS NOT EMPTY''
982          IF CSTKPTR LQ -1 $
983             BEGIN
984                ERROR (34H(ASSERTION COMMAND STACK UNDERFLOW.)) $
985                RETURN $
986             END
987
988          ''GENERATE LOOPING STATEMENTS$''
989             ''ASSIGN NEW VALUE TO INDEX''$
990                TXTPTR = 0 $
991                BYTE ($0,MAXTXT$) (TXTBUF) = BYTE ($0,MAXTXT$) (BLANK$) $
992                BYTE ($0,72$) (ASSTXT) = BYTE ($0,72$) (BLANKS) $
993                ASSLEN = LENNAME ($CSTKPTRS) $
994                BYTE ($0,ASSLEN$) (ASSTXT) = BYTE ($0,ASSLEN$)
995                   (VARNAME ($CSTKPTRS)) $
996                ADDTXT (JUSTRT(ASSTXT), ASSLEN, TXTPTR & TXTPTR) $
997                ADDTXT (3H( = ), 3, TXTPTR = TXTPTR) $
998                BYTE ($0,72$) (ASSTXT) = BYTE ($0,72$) (BLANKS) $
999                ASSLEN = LENNEXT ($CSTKPTRS) $
1000               BYTE ($0,ASSLEN$) (ASSTXT) = BYTE ($0,ASSLEN$)
```

ALTER NO ANZ1

```
1001                    (NEXTVXL ($CSTKPTR$)) $
1002                 ADDTXT (JUSTRT(ASSTXT), ASSLEN, TXTPTR & TXTPTR) $
1003                 ADDTXT (2H( $), 2, TXTPTR + TXTPTR) $
1004                 MARE (MODULE, JUSTRT(TXTBUF), TXTPTR, 0 + LENTXT) $
1005                 STOPRB (MODULE, DIRTV, UENTXT, LTEXT) $
1006
1007              ''CHECK FOR EXIT CONDITION''
1008                 TXTPTR # 0 $
1009                 BYTE ($0,MAXTXT$) (TXTBUF) # BYTE ($0;MAXTXT$) (BLANK$) $
1010                 ADDTXT (8H(IF NOT (), 8, TXTPTR + TXTPTR) $
1011                 BYTE ($0,72$) (ASSTXT) # BYTE ($0,72$) (BLANKS) $
1012                 ASSLEN # LENEXIT ($CSTKPTR$) $
1013                 BYTE ($0,ASSLEN$) (ASSTXT) # BYTE ($0;ASSLEN$)
1014                    (EXIT ($CSTKPTR$)) $
1015                 ADDTXT (JUSTRT(ASSTXT), ASSLEN, TXTPTR & TXTPTR) $
1016                 ADDTXT (3H() $), 3, TXTPTR + TXTPTR) $
1017                 MARE (MODULE, JUSTRT(TXTBUF), TXTPTR, 0 + LENTXT) $
1018                 STOPRB (MODULE, DIRTV, UENTXT, LTEXT) $
1019
1020              ''BRANCH TO TOP OF LOOP''
1021                 TXTPTR # 0 $
1022                 BYTE ($0,MAXTXT$) (TXTBUF) # BYTE ($0;MAXTXT$) (BLANK$) $
1023                 ADDTXT (5H(GOTO ), 5, TXTPTR & TXTPTR) $
1024                 BYTE ($0,72$) (ASSTXT) # BYTE ($0,72$) (BLANKS) $
1025                 ASSLEN # LENLABL ($CSTKPTR$) $
1026                 BYTE ($0,ASSLEN$) (ASSTXT) # BYTE ($0;ASSLEN$)
1027                    (LABEL ($CSTKPTR$)) $
1028                 ADDTXT (JUSTRT(ASSTXT), ASSLEN, TXTPTR & TXTPTR) $
1029                 ADDTXT (2H( $), 2, TXTPTR + TXTPTR) $
1030                 MARE (MODULE, JUSTRT(TXTBUF), TXTPTR, 0 + LENTXT) $
1031                 STOPRB (MODULE, DIRTV, UENTXT, LTEXT) $
1032
1033           ''IF BOUNDS CLAUSE, THEN ADD END-OF-LOOP LABEL''
1034           IF LENELBL ($CSTKPTR$) NQ 0 $
1035              BEGIN
1036                 TXTPTR # 0 $
1037                 BYTE ($0,MAXTXT$) (TXTBUF) # BYTE ($0;MAXTXT$) (BLANK$) $
1038                 BYTE ($0,72$) (ASSTXT) # BYTE ($0,72$) (BLANKS) $
1039                 ASSLEN # LENELBL ($CSTKPTR$) $
1040                 BYTE ($0,ASSLEN$) (ASSTXT) # BYTE ($0;ASSLEN$)
1041                    (ELABEL ($CSTKPTR$)) $
1042                 ADDTXT (JUSTRT(ASSTXT), ASSLEN, TXTPTR & TXTPTR) $
1043                 ADDTXT (1H(:), 1, TXTPTR & TXTPTR) $
1044                 MARE (MODULE, JUSTRT(TXTBUF), TXTPTR, 0 + LENTXT) $
1045                 STOPRB (MODULE, DIRTV, UENTXT, LTEXT) $
1046              END
1047
1048           ''IF COUNT CLAUSE, THEN ADD CHECK STMTS''
1049           IF LENMIN ($CSTKPTR$) NQ 0 $
1050              BEGIN
```

```
ALTER NO  ...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8   ANZ1

1051                ''GENERATE CHECK STMT''I
1052                   TXTPTR = 0 $
1053                   BYTE ($0,MAXTXT$) (TXTBUF) = BYTE ($0,MAXTXT$) (BLANK8) $
1054                   ADDTXT (8H(IF NOT (), 8$ TXTPTR = TXTPTR) $
1055                   BYTE ($0,72$) (ASSTXT) = BYTE ($0,72$) (BLANKS) $
1056                   ASSLEN = LENMIN (SCSTKPTR$) $
1057                   BYTE ($0,ASSLEN$) (ASSTXT) = BYTE ($0,ASSLEN$)
1058                      (MINCNT (SCSTKPTR$)) $
1059                   ADDTXT (JUSTRT(ASSTXT), ASSLEN, TXTPTR = TXTPTR) $
1060                   ADDTXT (14H( LQ ASSONT QQ ), 14, TXTPTR = TXTPTR) $
1061                   BYTE ($0,72$) (ASSTXT) = BYTE ($0,72$) (BLANKS) $
1062                   ASSLEN = LENMAX (SCSTKPTR$) $
1063                   BYTE ($0,ASSLEN$) (ASSTXT) = BYTE ($0,ASSLEN$)
1064                      (MAXCNT (SCSTKPTR$)) $
1065                   ADDTXT (JUSTRT(ASSTXT), ASSLEN, TXTPTR = TXTPTR) $
1066                   ADDTXT (3H() $), 3, TXTPTR = TXTPTR) $
1067                   MAKE (MODULE, JUSTRT(TXTBUF), TXTPTR, 0 = LENTXT) $
1068                   STOPRB (MODULE, DIRTV, CENTXT, LTEXT) $
1069
1070                ''GENERATE STMT TO PRINT ERROR MESSAGE''
1071                   TXTPTR = 0 $
1072                   BYTE ($0,MAXTXT$) (TXTBUF) = BYTE ($0,MAXTXT$) (BLANK$) $
1073                   ADDTXT (48H(JAVS'ASSERT = 34H(COUNT ASSERTION ERROR AT STMT ),
1074                      48, TXTPTR = TXTPTR) $
1075                   ADDTXT (ITSHOL(STMT), NBYTWD, TXTPTR = TXTPTR) $
1076                   ADDTXT (3H() $), 3, TXTPTR = TXTPTR) $
1077                   MAKE (MODULE, JUSTRT(TXTBUF), TXTPTR, 0 = LENTXT) $
1078                   STOPRB (MODULE, DIRTV, CENTXT, LTEXT) $
1079                END
1080
1081             ''ADD END STMT''
1082             MAKE (MODULE, 3H(END), 3, 0 = LENTXT) $
1083             STOPRB (MODULE, DIRTV, LENTXT, LTEXT) $
1084
1085             ''DECREMENT COMMAND STACK PTR''
1086             CSTKPTR = CSTKPTR - 1 $
1087
1088          END ''PRBEND0''
1089
1090                PROC PRBDIR (MODULE, STMT) $                                  PRBDIR 3
1091                ''     JAVS-5  --  MODULE INSTRUMENTATION COMPONENT     ''    0729RU 6
1092                ''     ''                                                    TEMP   1
1093                ''RECOGNIZES THE TYPE OF DIRECTIVE AND INVOKES THE           PRBDIR 4
1094                  PARTICULAR ROUTINE TO IMPLEMENT THE DIRECTIVE''            PRBDIR 5
1095                ITEM MODULE INTG $                                            PRBDIR 8
1096                ITEM STMT INTG $                                              PRBDIR 9
1097                BEGIN  ''PRBDIR''                                             PRBDIR10
1098                  ''STORE DIRECTIVE IN INSTRUMENTED TEXT TABLE''              PRBDIR11
1099                IF BEGBT OR 0$                                                0725JB 7
1100                STOLBL (MODULE)$                                              0725JB 8
```

```
ALTER NO  ....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8   ANZ1
1101                STOSB (MODULE, STMT) $                                          PRBDIR12
1102              HTEMP # BLNKS                                                     0709CG43
1103              BYTB(S0:NBYTWDS)(HTEMP) = BYTB(S0:NBYTWDS)((LLIST(S0/1S))$        0709CG44
1104                 IFEITH HTEMP EQ 4H(ASSE)$                                      0709CG47
1105                   PBRAST (MODULE, STMT) $                                      PRBDIR14
1106                 ORIF HTEMP EQ 4H(TRAC)$                                        0709CG48
1107                   PBBTDR (MODULE, STMT) $                                      PRBDIR16
1108                 ORIF HTEMP EQ 4H(OFFT)$                                        0709CG49
1109                   PBROFT (MODULE, STMT) $                                      PRBDIR18
1110                 ORIF HTEMP EQ 4H(EXPE)$                                        0709CG50
1111                   PBBXPT (MODULE, STMT) $                                      PRBDIR20
1112                 ORIF HTEMP EQ 4H(JAVS)$                                        J80724 8
1113                   RETURN$                                                      J80724 9
1114                 ORIF HTEMP EQ 4H(CHAI) $
1115                     PRBCHAIN (MODULE, STMT) $
1116                 ORIF HTEMP EQ 4H(ENDC) $
1117                     PRBENDC (MODULE, STMT) $
1118                 ORIF HTEMP EQ 4H(LOOP) $
1119                     PRBLOOP (MODULE, STMT) $
1120                 ORIF HTEMP EQ 4H(ENDL) $
1121                     PRBENDC (MODULE, STMT) $
1122                 ORIF HTEMP EQ 4H(VALU) $
1123                     PRBVALUE (MODULE, STMT) $
1124                 ORIF 1 $                                                       PRBDIR21
1125                     ERROR (26H(DIRECTIVE NOT RECOGNIZED;) ) $                  PRBDIR23
1126                 END                                                            PRBDIR26
1127               END   ''PRBDIR''                                                 PRBDIR27
1128
1129            PROC PRBLOOP (MODULE, STMT) $
1130            ''    JAVS-5  --  MODULE INSTRUMENTATION COMPONENT''
1131            ''    ''
1132            ''PRBBE LOOP DIRECTIVE  --  B.U. RAMEY, 2/16/78''
1133
1134            DEFINE DELI ''4H(DELI)'' $
1135            DEFINE IDENT ''4H(IDEN)'' $
1136            DEFINE LPARN ''4H(  )'' $
1137            DEFINE RANGE ''4H(  ,  )'' $
1138            DEFINE MAXCSTK ''10'' $ ''SIZE OF ASSERTION COMMAND STACK''
1139            ITEM ASSLEN INTG $
1140            ITEM ASSTXT H 72 $
1141            ITEM FROM INTG $
1142            ITEM KEYWRD INTG $
1143            ITEM MIDL INTG $
1144            ITEM MODULE INTG $
1145            ITEM STMT INTG $
1146            ITEM TO INTG $
1147            ITEM TXTPTR INTG $
1148            ITEM VARLEN INTG $
1149            ITEM VARTXT H 72 $
1150
```

```
ALTER NO  ....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8  ANZ1
1151          BEGIN
1152             '!SET UP MONITOR FOR ASSERTION VIOLATION OUTPUT!!
1153             PROBOL $
1154
1155             '!INCREMENT AND CHECK COMMAND STACK PTR!!
1156             CSTKPTR = CSTKPTR + 1 $
1157             IF CSTKPTR GQ MAXCSTK $
1158                BEGIN
1159                   ERROR (33H(ASSERTION COMMAND STACK OVERFLOW!)) $
1160                   RETURN $
1161                END
1162
1163             '!CHECK FOR AND HANDLE EMPTY CLAUSE!!
1164             KEYWRD = SCANSB (3, LENGTH-5, 4H(EMPT), IDER!) $
1165             IF (KEYWRD NQ -1) AND (LLIST(NO,KEYWRD+1S) BQ 4H(!   !) $
1166                BEGIN '!FOUND AN EMPTY CLAUSE!!
1167                   '!!ISOLATE BOOLEAN EXPRESSION!!
1168                      FROM = SCANSB (KEYWRD$2$ LENGTH+1, LPAREN, DELI) $
1169                      TO = BALPAR (FROM+1, LENGTH, LLIST) $
1170                      BLDLIN (FROM+2, TO-1, LLIST, 0, 72 = INEXT, BUFOUT) $
1171                      BYTE ($0,72$) (ASSTXT) = BYTE ($0,72$) (BUFOUT) $
1172                      ASSTXT = JUSTRT (ASSTXT) $
1173
1174                   '!BUILD IF NOT STMT!!
1175                      TXTPTR = 0 $
1176                      BYTE ($0,MAXTXT$) (TXTBUF) = BYTE ($0,MAXTX$) (BLANKS) $
1177                      ADDTXT (3H(IF NOT (), 8$ TXTPTR = TXTPTR) $
1178                      ASSLEN = LASTCH (ASSTXT) $
1179                      ADDTXT (ASSTXT, ASSLEN, TXTPTR = TXTPTR) $
1180                      ADDTXT (3H() $), 3, TXTPTR = TXTPTR) $
1181                      TXTBUF = JUSTRT (TXTBUF) $
1182                      MARE (MODULE, TXTBUF, TXTPTR, 0 = LENTXT) $
1183                      STOPRB (MODULE, DIRTY, LENTXT, LTEXT) $
1184                END
1185
1186             '!GENERATE BEGIN STMT!!
1187             MAKE (MODULE, 5H(BEGIN), 5, 0 = LENTXT) $
1188             STOPRB (MODULE, DIRTY, LENTXT$ LTEXT) $
1189
1190             '!ISOLATE INDEX VARIABLE AND SAVE IT ON STACK!!
1191             FROM = SCANSB (3, LENGTH-1, LPAREN, DELI) $
1192             TO = BALPAR (FROM+1, LENGTH, LLIST) $
1193             BLDLIN (FROM+2, TO-1, LLIST, 0, 72 = INEXT, BUFOUT) $
1194             BYTE ($0;72$) (VARTXT) = BYTE ($0;72$) (BUFOUT) $
1195             ASSTXT = JUSTRT (VARTXT) $
1196             VARLEN = LASTCH (ASSTXT) $
1197             BYTE ($0;VARLEN$) (VARNAME (SOSTKPTR$)) = BYTE ($0,VARLEN$) (VARTXT)$
1198             LENNAME (SCSTKPTR$) = VARLEN $
1199
1200             '!SAVE NEXT VALUE ON STACK!!
```

ALTER NO ....5....1....5....2....5....3....5....4....5....5....5....6....5....7....5....8 ANZ1

```
1201                T$TPTR = 0 $
1202                BYTE ($0,MAXTXTS) (TXTBUF) = BYTE ($0,MAXTXTS) (BLANKS) $
1203                VARTXT = JUSTRT (VARTXT) $
1204                ADDTXT (VARTXT, VARLEN, TXTPTR = TXTPTR) $
1205                ADDTXT ('H( + 1), 4, TXTPTR = TXTPTR) $
1206                BYTE ($0,TXTPTRS) (NEXTVAL ($CSTKPTRS)) = BYTE ($1,TXTPTRS) (TXTBUF)$
1207                LENNEXT ($CSTKPTRS) = TXTPTR $
1208
1209                ''ISOLATE INITIAL AND FINAL VALUES''
1210                FROM = SCANSB (TO, LENGTH-1, UPAREN, DELI) $
1211                TO = BALPAR (FROM+1, LENGTH, CLEST) $
1212                MIDL = SCANSB (FROM+1, TO-1, RANGE, DELI) $
1213
1214                ''GENERATE INITIALIZATION STMT FOR INDEX VARIABLE''
1215                T$TPTR = 0 $
1216                BYTE ($0,MAXTXTS) (TXTBUF) = BYTE ($0,MAXTXTS) (BLANKS) $
1217                ADDTXT (VARTXT, VARLEN, TXTPTR = TXTPTR) $
1218                ADDTXT ('H( = ), 3, TXTPTR = TXTPTR) $
1219                BLDLIN (FROM+2, MIDL, LLIST, 0, 72 - LNEXT, BUFOUT) $
1220                BYTE ($0,72$) (ASSTXT) = BYTE ($0,72$) (BUFOUT) $
1221                ASSTXT = JUSTRT (ASSTXT) $
1222                ASSLEN = LASTCH (ASSTXT) $
1223                ADDTXT (ASSTXT, ASSLEN, TXTPTR = TXTPTR) $
1224                ADDTXT ('H( $), 2, TXTPTR = TXTPTR) $
1225                T$TBUF = JUSTRT (TXTBUF) $
1226                MAKE (MODULE, TXTBUF, TXTPTR, 0 , LENTXT) $
1227                STOPRB (MODULE, DIRTY, LENTXT, TEXT) $
1228
1229                ''SAVE EXIT CONDITION ON STACK''
1230                T$TPTR = 0 $
1231                BYTE ($0,MAXTXTS) (TXTBUF) = BYTE ($0,MAXTXTS) (BLANKS) $
1232                ADDTXT (VARTXT, VARLEN, TXTPTR = TXTPTR) $
1233                ADDTXT ('H( 00 ), 4, TXTPTR = TXTPTR) $
1234                BLDLIN (MIDL+2, TO-1, LLIST, 0, 72 - LNEXT, BUFOUT) $
1235                BYTE ($0,72$) (ASSTXT) = BYTE ($0,72$) (BUFOUT) $
1236                ASSTXT = JUSTRT (ASSTXT) $
1237                ASSLEN = LASTCH (ASSTXT) $
1238                ADDTXT (ASSTXT, ASSLEN, T$TPTR = TXTPTR) $
1239                BYTE ($0,TXTPTRS) (EXIT ($CSTKPTRS)) = BYTE ($1,TXTPTRS) (TXTBUF) $
1240                LENEXIT ($CSTKPTRS) = TXTPTR $
1241
1242                ''GENERATE LABEL - SAVE IT ON COMMAND STACK''
1243                PBLBL ($TMT, 1H'0) = LABEL ($CSTKPTRS)) $
1244                LENLABL ($CSTKPTRS) = 6 $
1245
1246                ''GENERATE LABEL STMT''
1247                BYTE ($0,MAXTXTS) (TXTBUF) = BYTE ($0,MAXTXTS) (BLANKS) $
1248                ASSLEN = LENLABL ($CSTKPTRS) $
1249                BYTE ($0,ASSLENS) (TXTBUF) = BYTE ($0,ASSLENS) (LABEL($CSTKPTRS)) $
1250                BYTE ($A$SLEN,1$) (TXTBUF) = 1H(.) $
```

ALTER NO ANZ1

```
1251              TXTBUF = JUSTR (TXTBUF) $
1252              MAKE (MODULE, TXTBUF, ASSLEN+1, 0 = LENTXT) $
1253              STOPRB (MODULE, DIRTV, LENTXT, LTEXT) $
1254
1255              !!SET OPTIONS AS NOT USED!!
1256              LENELBL (SCSTKPTRS) = 0 $
1257              LENMIN (SCSTKPTRS) = 0 $
1258
1259           END !!PRBLOOP!!
1260
1261           PROC PRBVALUE (MODULE, STMT) $
1262           !!     VAVS85  -- MODULE INSTRUMENTATION!!
1263           !!     !!
1264           !!PROBE ASSERT VALUE/NOT VALUE DIRECTIVES!!
1265
1266           DEFINE COMMA !!4H(,   )!! $
1267           DEFINE DELI !!4H(DEL()!! $
1268           DEFINE IDENT !!4H(IDEN)!! $
1269           DEFINE OPER !!4H(OPER)!! $
1270           DEFINE RANGE !!4H(,,, )!! $
1271           DEFINE LPAREN !!4H((   )!! $
1272
1273           ITEM ASSLEN INTG $
1274           ITEM ASSTXT H 72 $
1275           ITEM FROM INTG $
1276           ITEM MIDL INTG $
1277           ITEM MODULE INTG $
1278           ITEM RPAREN INTG $
1279           ITEM STMT INTG $
1280           ITEM TO INTG $
1281           ITEM TXTPTR INTG $
1282           ITEM VARLEN INTG $
1283           ITEM VARTXT H 72 $
1284
1285           BEGIN
1286              !!SET UP MONITOR FOR ASSERTION VIOLATION OUTPUT!!
1287              PRBOOL $
1288
1289              !!BUILD TEXT STRING CONTAINING VARIABLE NAME!!
1290              FROM = SCANSB (3, LENGTH-1, LPAREN, DELI) $
1291              TO = BALPAR (FROM+1, LENGTH, CLIST) $
1292              BLDLIN (FROM+2, TO-1, LLIST, 0, 72 & !NEXT, BUFOUT) $
1293              BYTE (S0,72S) (VARTXT) = BYTE (S0,72S) (BUFOUT) $
1294              VARTXT = JUSTR (VARTXT) $
1295              VARLEN = LASTCH (VARTXT) $
1296
1297              !!START BUILDING IF STMT!!
1298              TXTPTR = 0 $
1299              BYTE (S0,MAXTXTS) (TXTBUF) = BYTE (S0,MAXTXTS) (BLANKS) $
1300              ADDTXT (3H(IF ), 3, TXTPTR & TXTPTR) $
```

```
ALTER NO  ....5....1....5....2....3....4....5....6....7....8  ANZ1
1301
1302            !!CHECK FOR 'NOT' BETWEEN VARIABLE AND VALUES!
1303            NEGATE SENSE OF DIRECTIVE IN IF STMT.!!
1304            IF LLIST ($0,TOS) NQ 4H(NOT ) $
1305               ADDTXT (4H(NOT ), 4, TXTPTR = TXTPTR) $
1306
1307            !!INSERT LPAREN TO SURROUND TOTAL EXPRESSION!!
1308            ADDTXT (2H((), 1, TXTPTR = TXTPTR) $
1309
1310            !!FIND PARENS SURROUNDING VALUES!!
1311            FROM = SCANSB (TO, LENGTH-1, LPAREN, DELI) $ !!INDEX VALUE!!
1312            RPAREN = BALPAR (FROM+1, LENGTH, LLIST) - 1 $ !!INDEX VALUE!!
1313
1314            !!LOOP FOR EACH VALUE OR RANGE IN THE LIST!!
1315         VALULOOP:
1316               ADDTXT (1H((), 1, TXTPTR = TXTPTR) $
1317
1318               !!LOOK FOR COMMA SEPERATING NEXT VALUE!!
1319                  TO = SCANSB (FROM+1, LENGTH-1, COMMA, DELI) $ !!INDEX VALUE!!
1320                  IF (TO EQ -1) OR (TO GR RPAREN) $
1321                     !!LAST VALUE OR RANGE!!
1322                     TO = RPAREN $
1323
1324               !!LOOK FOR RANGE INDICATOR!!
1325                  MIDL = SCANSB (FROM+1, TO-1, RANGE, DELI) $
1326
1327               IF MIDL EQ -1 $
1328                  BEGIN !!SIMPLE VALUE - BUILD CORRECT BOOLEAN EXPR.!!
1329                     PRBCHKL (MODULE, VARLEN, TXTPTR = TXTPTR) $
1330                     ADDTXT (VARTXT, VARLEN, TXTPTR = TXTPTR) $
1331                     BLDLIN (FROM+2, TO, LLIST, 0, 72 - INEXT, BUFOUT) $
1332                     BYTE ($0,72$) (ASSTXT) = BYTE ($0,72$) (BUFOUT) $
1333                     ASSTXT = JUSTR (ASSTXT) $
1334                     ASSLEN = LASTCH (ASSTXT) $
1335                     PRBCHKL (MODULE, ASSLEN+4, TXTPTR = TXTPTR) $
1336                     ADDTXT (4H( EQ ), 4, TXTPTR = TXTPTR) $
1337                     ADDTXT (ASSTXT, ASSLEN, TXTPTR = TXTPTR) $
1338                  END
1339               ORIF $ $
1340                  BEGIN !!RANGE - BUILD CORRECT BOOLEAN EXPR.!!
1341                     !!INSERT TEXT FOR LOW BOUND!!
1342                        BLDLIN (FROM+2, MIDL, LLIST, 0, 72 - INEXT, BUFOUT) $
1343                        BYTE ($0,72$) (ASSTXT) = BYTE ($0,72$) (BUFOUT) $
1344                        ASSTXT = JUSTR (ASSTXT) $
1345                        ASSLEN = LASTCH (ASSTXT) $
1346                        PRBCHKL (MODULE, ASSLEN, TXTPTR = TXTPTR) $
1347                        ADDTXT (ASSTXT, ASSLEN, TXTPTR = TXTPTR) $
1348
1349                     PRBCHKL (MODULE, VARLEN+4, TXTPTR = TXTPTR) $
1350                     ADDTXT (4H( LQ ), 4, TXTPTR = TXTPTR) $
```

```
ALTER NO  ....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8  ANZ1
1351                    ADDTXT (VARTXT, VARLEN, TXTPTR = TXTPTR) $
1352
1353                    ''INSERT TEXT FOR HIGH BOUND''
1354                       BLDLIN (MIDL+2, TO, LLIST, 0, 72 $ INTXT, BUFOUT) $
1355                       BYTE ($0,72$) (ASSTMT) = BYTE ($0,72$) (BUFOUT) $
1356                       ASSTXT = JUSTRT (ASSTXT) $
1357                       ASSLEN = LASTCH (ASSTXT) $
1358                       PRBCHKL (MODULE, ASSLEN+4, TXTPTR = TXTPTR) $
1359                       ADDTXT (4H( LO ), 4, TXTPTR = TXTPTR) $
1360                       ADDTXT (ASSTXT, ASSLEN, TXTPTR = TXTPTR) $
1361                 END
1362              END ''IFEITHER''
1363
1364              PRBCHKL (MODULE, 1, TXTPTR = TXTPTR) $
1365              ADDTXT (1H(), 1, TXTPTR = TXTPTR) $
1366
1367              IF TO NQ RRAREN $
1368                 BEGIN ''MORE VALUES OR RANGES TO DO''
1369                    PRBCHKL (MODULE, 5, TXTPTR = TXTPTR) $
1370                    ''ALSO INCLUDES ROOM FOR NEXT OPEN PARENS.''
1371                    ADDTXT (4H( OR ), 4, TXTPTR = TXTPTR) $
1372                    FROM = TO $
1373                    GOTO VALULOOP $
1374                 END
1375
1376           ''TERMINATE IF STMT''
1377              PRBCHKL (MODULE, 3, TXTPTR = TXTPTR) $
1378              ADDTXT (3H() $), 3, TXTPTR = TXTPTR) $
1379              MAKE (MODULE, JUSTRT(TXTBUF), TXTPTR, 0 * LENTXT) $
1380              STOPRM (MODULE, DIRTV, LENTXT, LTEXT) $
1381
1382           ''ADD STMT TO PRINT ERROR MESSAGE''
1383              TXTPTR = 0 $
1384              BYTE ($0,MAXTXT$) (TXTBUF) = BYTE ($0,MAXTXT$) (BLANK$) $
1385              ADDTXT (48H(JAVS'ASSERT # 34H(VALUE ASSERTION ERROR AT STMT ),
1386                 48, TXTPTR # TXTPTR) $
1387              ADDTXT (ITSHOL(STMT), NBYTWD, TXTPTR = TXTPTR) $
1388              ADDTXT (3H() $), 3, TXTPTR = TXTPTR) $
1389              TXTBUF = JUSTRT (TXTBUF) $
1390              MAKE (MODULE, TXTBUF, TXTPTR, 0 * LENTXT) $
1391              STOPRM (MODULE, DIRTV, LENTXT, LTEXT) $
1392
1393        END ''PRBVALUE''
1394
1395        PROC PRBCHKL (MODULE, CHARS, LENGTH = NEW'LENGTH) $
1396        ''     JAVS45  --  MODULE INSTRUMENTATION COMPONENT''
1397        ''     ''
1398        ''     CHECKS TO MAKE SURE STRING OF LENGTH 'CHARS' CAN BE ADDED TO
1399               TXTBUF WHICH ALREADY CONTAINS A STRING OF LENGTH 'LENGTH'.
1400               IF NOT, THE CURRENT CONTENTS OF TXTBUF ARE WRITTEN OUT
```

```
ALTER NO      0...5...1...5...2...5...3...5...4...5...5...6...5...7...5...8          ANZ1

1401                AND THE BUFFER IS CLEARED?  ''NEW'LENGTH'' IS THE LENGTH OF
1402                THE STRING LEFT IN TXTBUF AT THE END OF THE PROC.''
1403
1404          ITEM CHARS INTO $
1405          ITEM LENGTH INTO $
1406          ITEM MODULE INTO $
1407          ITEM NEW'LENGTH INTO $
1408
1409          BEGIN
1410             NEW'LENGTH = LENGTH $
1411             IF LENGTH>CHARS OR MAXTXT $
1412                BEGIN
1413                   TXTBUF = JUSTR (TXTBUF) $
1414                   MAKE (MODULE, TXTBUF, LENGTH, 0 = LENTXT) $
1415                   STOPRB (MODULE, DIRTY, LENTXT, LTEXT) $
1416                   NEW'LENGTH = 0 $
1417                   BYTE ($0,MAXTXT$) (TXTBUF) = BYTE ($0,MAXTXT$) (BLANKS) $
1418                END
1419
1420          END ''PRBCHRL''
1421
1422          PROC PRBDCL $
1423          ''     JAVS#5  --  MODULE INSTRUMENTATION COMPONENT''
1424          ''     ''
1425          ''SET UP MONITOR FOR OUTPUTTING ASSERTION VIOLATION MESSAGES''
1426          ''     U.L. RAMEY, 4/1/78''
1427
1428          BEGIN
1429             IF NOT DECLARED $
1430             BEGIN
1431                MAKE (MODULE, 23H(ITEM JAVS'ASSERT H 72 B)I 23, 0 = LENTXT)$
1432                STOPRB (MODULE, DIRTY, LENTXT, LTEXT) $
1433                MAKE (MODULE, 21H(MONITOR JAVS'ASSERT B)I 21, 0 = LENTXT) $
1434                STOPRB (MODULE, DIRTY, LENTXT, LTEXT) $
1435                DECLARED = TRUE $
1436             END
1437             END ''PRBDCL''
1438
1439          PROC PRBLBL (STMT, NO = LABEL) $
1440          ''     JAVS#5  --  MODULE INSTRUMENTATION COMPONENT''
1441          ''     ''
1442          ''GENERATE A SIX CHARACTER LABEL USING A STMT NUMBER (STMT) AND
1443          CALLER SPECIFIED 1-DIGIT NUMBER (NO)'' ''
1444          ''     U.L. RAMEY, 4/1/78''
1445
1446          ITEM LABEL H 6 $
1447          ITEM NO H 1 $
1448          ITEM STMT INTG $
1449
1450          BEGIN
```

```
  ALTER NO      ...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8          ANZ1
1451              BYTE (SOT1S) (LABBL) = 1H(J) $
1452              BYTE (S1?4S) (LABEL) = ITSHOL (STMT) $
1453              BYTE (S5?1S) (LABBL) = NO $
1454
1455              !!CHANGE SPACES TO ZEROES IN STMT NO.!!
1456              FOR I = 1, 1, 4 $
1457              BEGIN
1458                 IF BYTE (S1,1S) (LABEL) EQ 1H( ) $
1459                    BYTE (S1,1S) (LABEL) = 1H(0) $
1460              END
1461
1462           END !!PROBE!!
1463
1464                PROC PROBE (MODULE, STMT) $                                          PROBEF  3
1465                !!     JAVS-9  --  MODULE INSTRUMENTATION COMPONENT     !!        0729RU  6
1466                !!     !!                                                          TEMP    1
1467                !! PROBE END OF IFEITH STATEMENT !!                                PROBEF  4
1468                ITEM LENTXT INTG $      !!CURRENT LENGTH OF LTEXT ARRAY!!        PROBEF  7
1469                ITEM MARK INTG $        !!STATEMENT NUMBER OF 1ST ORIF IN NEST!! PROBEF  8
1470                ITEM MODULE INTG $      !!MODULE NUMBER!!                        PROBEF  9
1471                ITEM NTR INTG $   !! POINTER TO NEXT CONTROL STATEMENT IN IFEITH!!PROBEF 10
1472                ITEM STMT INTG $        !!CURRENT STATEMENT NUMBER!!             PROBEF 11
1473                ITEM SB INTG $  !! STATEMENT BLOCK NUMBER !!                     PROBEF 12
1474                ITEM JJ INTG $                                                   PROBEF 13
1475                BEGIN                                                            PROBEF 23
1476                  !!                                                        !!   PROBEF 24
1477                  !!.........CHECK FOR ORIF 1 IN NEST.........!!                 PROBEF 25
1478                  !!                                                        !!   PROBEF 26
1479                  MARK = CASE2(MODULE, STMT) $                                   PROBEF 27
1480                  !!SEARCH IFEITH/ORIF FOR ORIF1!!                               PROBEF 28
1481                  SB = MARK $                                                    PROBEF 29
1482           GET!PTR;                                                              PROBEF 30
1483                  PTR = !GTWRD(MODULE, SBBAUM, SB, 7) $                          PROBEF 31
1484                  !TYPE = !GTWRD(MODULE, SBBAU?, PTR, 8) $                       PROBEF 32
1485                  IF !TYPE EQ 900 $       !!ORIF 1 PRESENT --> DON'T PROBE!!     PROBEF 33
1486                    BEGIN                 !!UNSTACK FALSE BRANCH DO-PATHS!!      PROBEF 34
1487           UNSTACK;                                                              PROBEF 35
1488                      IF LASTIF LQ 0 $                                           PROBEF 36
1489                        BEGIN                                                    PROBEF 37
1490                          ERROR (16H(STACK UNDERFLOW.)) $                        PROBEF 38
1491                          RETURN $                                               PROBEF 39
1492                        END                                                      PROBEF 40
1493                      IF FDDPS(S LASTIF   8) EQ MARK $                           PROBEF 41
1494                        BEGIN                                                    PROBEF 42
1495                          LASTIF = LASTIF + 1 $                                  PROBEF 43
1496                          GOTO STORE!END $                                       PROBEF 44
1497                        END                                                      PROBEF 45
1498                      LASTIF = LASTIF - 5 $                                      PROBEF 46
1499                      GOTO UNSTACK $                                             PROBEF 47
1500                    END                                                          PROBEF 48
```

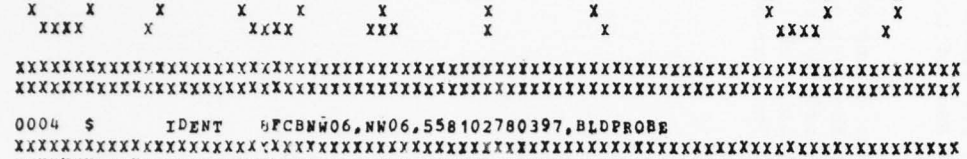
APPENDIX V

This section contains the output listings from an example run of the modified JAVS. The example was designed to illustrate the use of our new assertion constructs and the use of JAVS as a preprocessor for those constructs. The program does nothing and in fact contains only one executable statement. (If there are no executable statements, JAVS ignores the module entirely.) The logical data structure that the assertions are being applied to is a two-dimensional array of records. More specifically, it is a sequential list of linked lists of records. Thus, to access each record, a CHAIN loop is nested within a LOOP loop. Within the inner loop, the VALUE and NOT VALUE assertions are each demonstrated. For illustrative purposes, all the possible optional clauses are used with each of the directives.

The first job activity is the JAVS basic source analysis during which the JAVS library for this particular program is built. In the next activity, JAVS is run again to generate the probed text from the original source and the directives. The third activity is a compool compilation that is necessary for compilation of the instrumented source code. The last activity demonstrates the compilation of the probed text.



```
0004  $       IDENT   BFCBNW06,NW06,558102780397,BLDPROBE
```



```
$$    8751T ENTERED SYST12 AT 19.756 FROM SYSTEM-0 TSS/S   0-20-03

0001  $       SNUMB   8751T
0002  $       COMMENT BFCBNW06         TSS CARDIN
0003  $$      USERID  BFCBNW06$##########
0004  $       IDENT   BFCBNW06,NW06,558102780397,BLDPROBE
0005  $$      USERID  BFCBNW06$####
0006  $$      SELECT  BFCBNW08/STEP1-6
0007*A$       PROGRAM RLHS
0008* $$      PRMFL   H*,R,R,BFCBNW08/MJAVS1-6
0009* $       LIMITS  99,82K,-5K,40000
0010* $       FILE    03,X3R,10R
0011* $       FILE    10,C1R,1L
0012* $       FILE    04,X4R,2L
0013  $$      PRMFL   02,R/W,R,BFCBNW06/LIBRARYJ
0014  $$      PRMFL   09,R,S,BFCBNW06/BCDBLK
0015  $$      SELECT  BFCBNW08/STEP2-6
0016*A$       PROGRAM RLHS
0017* $$      PRMFL   H*,R,R,BFCBNW08/MJAVS2-6
0018* $       FILE    02,X2R,20R
0019* $       FILE    10,C1R,1L
0020* $       LIMITS  60,58K,-5K,20000
0021* $       FILE    03,X3R,10R
0022* $       FILE    04,X4R,2L
0023  $$      PRMFL   01,R,R,BFCBNW06/LIBRARYJ
```

```
0024  $       FILE    04,X4R.2UN
0025  $$      PRMFL   07,W.S.BFCBNW06/BCDBLK
0026  $       LOWLOAD
0027  $       OPTION  FORTRAN
0028  A$      JOVIAL  NOPT.NDECK.NAME/PBPOOL/.POOLOU/PC/
0029  $       FILE    PC,X1S.2L
0030  $$      SELECT  BFCBGRC1/COMPILEB
0031* $$      PRMFL   *,R.R.BFCBGRC1/BACKUP
0032* $       LIMITS  10,54K..20000
0033* $       FILE    *6,C3D.2L
0034* $       FILE    *4,D4D.1L
0035* $       FILE    *7,E5D.1L
0036* $       FILE    *1,F6D.2L
0037* $       FILE    *5,G7D.1L
0038* $       FILE    *2,H8D.3L
0039* $       FILE    *3,I9D.3L
0040  $       LIMITS  1,49K
0041  $$      SELECT  BFCBGRC2/PBPOOL
0042  A$      JOVIAL  NOPT.NDECK.POOLIN/PC/
0043  $       FILE    PC,X1R.2L
0044  $$      SELECT  BFCBGRC1/COMPILEB
0045* $$      PRMFL   *,R.R.BFCBGRC1/BACKUP
0046* $       LIMITS  10,54K..20000
0047* $       FILE    *6,C3D.2L
0048* $       FILE    *4,D4D.1L
0049* $       FILE    *7,E5D.1L
0050* $       FILE    *1,F6D.2L
0051* $       FILE    *5,G7D.1L
0052* $       FILE    *2,H8D.3L
0053* $       FILE    *3,I9D.3L
0054  $       LIMITS  10,49K
0055  $$      PRMFL   S*,R/W.S.BFCBNW06/BCDBLK
0056  $       ENDJOB
  TOTAL CARD COUNT THIS JOB = 000166


* ACCT-01  $CARD #0007* RLHS     05/08/78   SW=000000000000
  * NORMAL TERMINATION      AT 150471 I=4020 SW=000000000000

      START 19.946       LINES   203      PROC 0.0028       I/O   0.001     IU  5   MEMORY   82K
      STOP  19.951       LIMIT 80000      LIMIT 0.9900      LIMIT           CU  5   M*T      1609
      SWAP   0.000
      LAPSE  0.006 FC D   TYPE      BUSY     IP/AT     EP/RT     IB/BC MB/BE     ADDRESS T#/PK#

                   I* R   D191 *      37         0         1        1     1      0-20-03
                   M* R   D191 P    1314         0         0      468   468R     0-20-01
                   03 R   D191 *      52         0         0      120   120R     0-20-04
                   10 R   D191 *      61         0         1       12    12      0-20-05
                   04 R   D191 *      49         0         1       24    24      0-20-02
                   02 R   D191 P    1563         0         0       50    50R     2-17-00
                   09 R   WHAT       53     19.945    19.951       27     3-01-00
                   P*     SYOUT

      BC-06   203 LINES AT STA.  -0


                    CPU       SYSOUT        TAPE        DISK     PRINTER      READER       PUNCH     CONSOLE       FIXED       TOTAL
RATE             225.00         .002      .00031      .00005      .00006      .00065      .00002        0.00        0.00
CHARGE         0000.63      0000.01     0000.00     0000.15     0000.00     0000.00     0000.00     0000.00     0000.00     0001.19
```

```
* ACCT-02  SCARD #0016* BLHS      05/08/78   SW=000000000000
  * NORMAL TERMINATION      AT 104411 I=4020 SW=000000000000

      START 19.952        LINES    252       PROC  0.0039       I/O   0.002     IU  5   MEMORY   58K
      STOP  19.962        LIMIT  20000       LIMIT 0.6000       LIMIT           CU  5   M*T      2089
      SWAP   0.000
      LAPSE  0.010 FC D   TYPE       BUSY     IP/AT     FP/RT     IS/#C MS/#E    ADDRESS T#/PR#

                   I* R   D191 *       34         0         1         1     1    0-20-03
                   H* R   D191 P     1707         0         0       350   350R   0-20-01
                   10 R   D191 *       48         0         1        12    12    0-20-02
                   03 R   D191 *      456         0         0       120   120R   0-20-03
                   09 R   D191 *       59         0         1        24    24    0-20-04
                   01 R   WHAT        342    19.952    19.962         0     0-30-00
                   02 R   WHAT       1757    19.952    19.962        27     3-01-00
                   07 R   D191 P      87         0         4        12    12    0-20-01
                   P*     SYOUT

      PC-06    252 LINES AT STA.  -0


                  CPU        SYSOUT        TAPE         DISK       PRINTER       READER        PUNCH       CONSOLE       FIXED         TOTAL
RATE            225.00          .002       .00031       .00005       .00006       .00065       .00002          0.00        0.00
CHARGE        0000.88       0000.50      0000.00      0000.12      0000.00      0000.00      0000.00      0000.00      0000.00      0001.50

* ACCT-03  SCARD #0028  JOVIAL   05/08/78   SW=000200000000
  * NORMAL TERMINATION      AT 002576 I=5000 SW=000200000000

      START 19.963        LINES    114       PROC  0.0004       I/O   0.001     IU  5   MEMORY   49K
      STOP  19.964        LIMIT  20000       LIMIT 0.0100       LIMIT           CU  5   M*T      337
      SWAP   0.000
      LAPSE  0.002 FC D   TYPE       BUSY     IP/AT     FP/RT     IS/#C MS/#E    ADDRESS T#/PR#

                   S* R   D191 *       45         0         5         5     5    0-20-03
                   D* R   D191 *       14         0         1         1     1    0-20-03
                   PC S   D191 *       61         0         0        24    24    0-20-02
                   ** R   D191 P     1312         0         0       960   960R   0-20-01
                   *6 R   D191 *       31         0         0        24    24    0-20-03
                   *8 R   D191 *       51         0         0        12    12    0-20-02
                   *7 D   WHAT          0    19.963    19.964        27     3-01-00
                   *1 R   D191 *        0         0         0        24    24    0-20-04
                   *5 R   D191 *        0         0         0        12    12    0-20-06
                   *2 R   D191 *        0         0         0        36    36    0-20-02
                   *3 R   D191 *        0         0         0        36    36    0-20-03
                   P*     SYOUT
                   K*     SYOUT
                   C*     SYOUT

      LIST     114 LINES AT STA.  -0


                  CPU        SYSOUT        TAPE         DISK       PRINTER       READER        PUNCH       CONSOLE       FIXED         TOTAL
RATE            225.00          .002       .00031       .00005       .00006       .00065       .00002          0.00        0.00
CHARGE        0000.09       0000.23      0000.00      0000.08      0000.00      0000.00      0000.00      0000.00      0000.00      0000.40
```

```
* ACTY-04  $CARD #0042  JOVIAL   05/08/78   SW=000200000000
  * NORMAL TERMINATION      AT 003576 I=5000 SW=000200000000

      START 19.965       LINES     86      PROC  0.0005       I/O   0.001    IU  5   MEMORY   49K
      STOP  19.967       LIMIT  20000      LIMIT 0.1000       LIMIT          CU  5   M*T      385
      SWAP   0.000
      LAPSE  0.002  FC D   TYPE      BUSY     IP/AT     FP/RT     IS/#C  MB/#E    ADDRESS T#/PK#

                    FC  R   D191  *     52         0         0        24     24    0-20-02
                    D*  R   D191  *     17         0         1         1      1    0-20-03
                    **  R   D191  P   1675         0         0       960    960R   0-20-01
                    *6  R   D191  *     54         0         0        24     24    0-20-03
                    *E  R   D191  *     21         0         0        12     12    0-20-02
                    *7  D   WHAT        50    19.965    19.967         0   0-30-00
                    *1  R   D191  *     73         0         0        24     24    0-20-04
                    *5  R   D191  *     15         0         0        12     12    0-20-06
                    *2  R   D191  *     84         0         0        36     36    0-20-02
                    *3  R   D191  *      0         0         0        36     36    0-20-03
                    5*  R   D191  P     85         0         5        12     12    0-20-01
                    P*      SYOUT
                    K*      SYOUT
                    C*      SYOUT

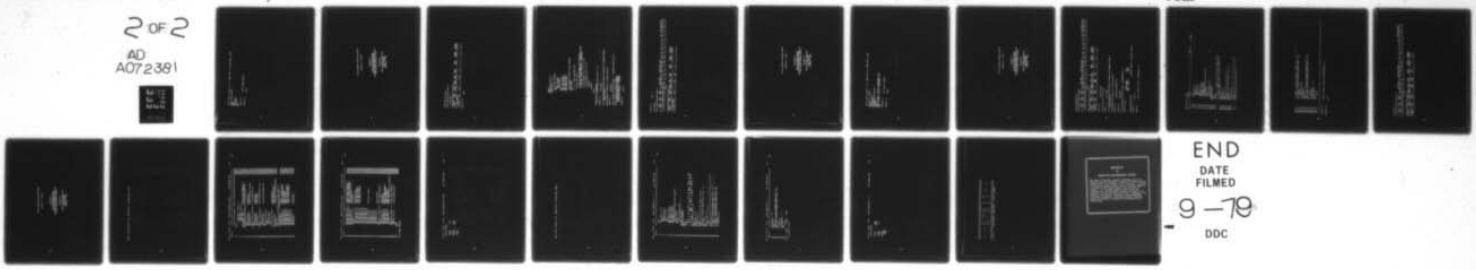
      LIST     86 LINES AT STA.  -0

                    CPU        SYSOUT        TAPE        DISK      PRINTER      READER       PUNCH      CONSOLE      FIXED       TOTAL
RATE             225.00          .002       .00031      .00005      .00006      .00065      .00002         0.00       0.00
CHARGE         0000.11       0000.17      0000.00     0000.10     0000.00     0000.00     0000.00      0000.00    0000.00     0000.38
```

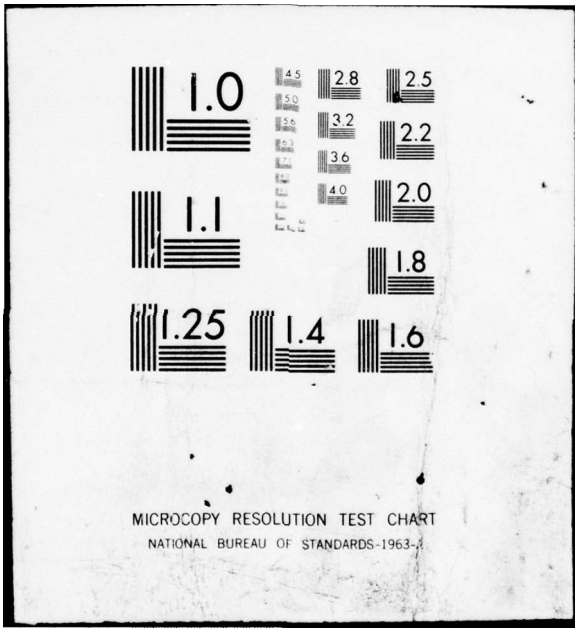
AD-A072 381 NORTHWESTERN UNIV EVANSTON IL DEPT OF ELECTRICAL ENG--ETC F/G 9/2
DYNAMIC MONITORING FOR LINEAR LIST DATA STRUCTURES.(U)
JUN 79 S S YAU, J L RAMEY F30602-76-C-0397

UNCLASSIFIED RADC -TR-79-128 NL

2 OF 2
AD
A072381



END
DATE
FILMED
9-79
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```
 SNUMB = 87517, ACTIVITY # = 01, , REPORT CODE = 06, RECORD COUNT = 000203

CREATE LIBRARY=LIBRARYJ.
START.
BASIC.
FOR LIBRARY.
STRUCTURAL.
END FOR.
END.

COMMAND IS...
                    CREATE LIBRARY=LIBRARYJ.

COMMAND IS...
                    START.
```

JOVIAL AUTOMATED VERIFICATION SYSTEM

VERSION 2.2 06-03-77

SPONSORED BY

AIR FORCE SYSTEMS COMMAND
ROME AIR DEVELOPMENT CENTER
GRIFFISS AIR FORCE BASE, NEW YORK 13441
UNDER CONTRACT F30602-73-C-0348

DEVELOPED BY

GENERAL RESEARCH CORPORATION
P. O. BOX 3587
SANTA BARBARA
CALIFORNIA 93105

```
STARTER INITIALIZATION...

KNOWN MODULE DESCRIPTOR BLOCKS...

     SYSTEM EMPTY

LIBRARY INFORMATION--RUN OF 0508
   LIBRARY      TYPE       DATE    TIMES    LAST     TOTAL       LINKS      LIBRARY     LIBRARY
NO. NAME       ACCESS     CREATED ALTERED  ALTERED  WORDS       USED       MODULES     FRAGMENTS
---------------------------------------------------------------------------------------------
 1  **NOT OPENED**
 2 LIBRARYJ     NEW        0508     1       0508      20          1          0           1
---------------------------------------------------------------------------------------------

COMMAND IS...
                    BASIC.
```

```
START $
DEFINE INTG ''I 24 S'' $
DEFINE NN ''5'' $
DEFINE MAXRECS ''100'' $

   TABLE MATRIX R MAXRECS $
   BEGIN
      ITEM LINK INTG $
      ITEM XX INTG $
      ITEM YY INTG $
   END
   ARRAY PTR NN INTG $
   ARRAY LOCNT NN INTG $
   ARRAY HICNT NN INTG $
   ARRAY LOBND NN INTG $
   ARRAY HIBND NN INTG $

   ''DECLARATIONS FOR ASSERTION INDEX VARIABLES''
   ITEM II INTG $
   ITEM KK INTG $

   II = 0 $  ''NEED AN EXECUTABLE STMT''

   ''.LOOP (KK) (0...NN-1)''

      ''.CHAIN (II) INIT (PTR(0KK0)) NEXT (LINK(0II0))
      END (II NQ -1) INIT (PTR(0KK0) NQ -1 ) COUNT (LOCNT(0KK0)
      ...HICNT(0KK0)) BOUNDS (LOBND(0KK0)...HIBND(0KK0))''

      ''.VALUE (XX(0II0)) (-11...1, 3, 9...10)''

      ''.VALUE (YY(0II0)) NOT (1, 4...7)''

      ''.ENDCHAIN''

   ''.ENDLOOP''

TERM $
```

```
*MAIN* (*MOJAVS*) COMPLETED
   ******** NO ERRORS WERE FOUND BY JAVS-2 ******

COMMAND IS...
                FOR LIBRARY.

THE FOLLOWING MODULES HAVE BEEN QUEUED FOR PROCESSING BY SUBSEQUENT COMMANDS
     *MOMAIN*(*MOJAVS*)

MODULE SELECTED IS <*MOMAIN*> OF JAVSTEXT <*MOJAVS*>

COMMAND IS...
                STRUCTURAL.

JOVIAL AUTOMATED VERIFICATION SYSTEM   *** SECONDARY MODULE ANALYSIS ***

MODULE  *MOMAIN*  OF JAVSTEXT  *MOJAVS*
     MODULE DEPENDENCE TABLE CONSTRUCTED.
     STATEMENT DESCRIPTOR BLOCKS UPDATED.
     DD-PATH TABLE CONTAINS    1 ENTRIES.

COMMAND IS...
                END FOR.
```

```
COMMAND IS...
                END.

JAVS WRAPUP...

KNOWN MODULE DESCRIPTOR BLOCKS...

     MODULE     TEXT     PARENT                PROC PROBE        EXEC FIRST  WORD                          DS   PARMS DIRECT COMP
NO.   NAME      NAME      NAME     TYPE SCOPE LINS STMTS STMTS  EXEC PAIRS  TOKS  SIMS  SLTS  DMTS DDPS BLKS  IN OUT CODE TOT DDP
---------------------------------------------------------------------------------------------------------------------------------
  1 *MOMAIN* *MOJAVS* *MOMAIN* PROG EXT    0    25     1    17   260   203     0    14     0    1    2    0   0   MO    0   0
---------------------------------------------------------------------------------------------------------------------------------
                             TOTAL        0    25     1                                          1

LIBRARY INFORMATION--RUN OF 0508
   LIBRARY     TYPE      DATE    TIMES   LAST     TOTAL       LLINKS      LIBRARY     LIBRARY
NO. NAME       ACCESS    CREATED ALTERED ALTERED  WORDS       USED        MODULES     FRAGMENTS
-----------------------------------------------------------------------------------------------
 1  **NOT OPENED**
 2 LIBRARIJ    NEW       0508    1       0508     4520        16          1           9
-----------------------------------------------------------------------------------------------
```

JOVIAL AUTOMATED VERIFICATION SYSTEM

VERSION 2.2 08-03-77

SPONSORED BY

AIR FORCE SYSTEMS COMMAND
ROME AIR DEVELOPMENT CENTER
GRIFFISS AIR FORCE BASE, NEW YORK 13441
UNDER CONTRACT F30602-73-C-0348

DEVELOPED BY

GENERAL RESEARCH CORPORATION
P. O. BOX 3587
SANTA BARBARA
CALIFORNIA 93105

```
 SNUMB = 87617, ACTIVITY # = 02, , REPORT CODE = 06, RECORD COUNT = 000252

OLD LIBRARY = LIBRARYJ.
START.
JAVSTEXT='NOJAVS'.
FOR JAVSTEXT.
INSTRUMENT,MODE=DIRECTIVES.
INSTRUMENT.
END FOR.
PRINT. JAVSTEXT = 'NOJAVS', INSTRUMENTED = ALL.
PUNCH. JAVSTEXT = 'NOJAVS', INSTRUMENTED = ALL.
END.

COMMAND IS...
                    OLD LIBRARY = LIBRARYJ.

COMMAND IS...
                    START.
```

JOVIAL AUTOMATED VERIFICATION SYSTEM

VERSION 2.2 06-03-77

SPONSORED BY

AIR FORCE SYSTEMS COMMAND
ROME AIR DEVELOPMENT CENTER
GRIFFISS AIR FORCE BASE, NEW YORK 13441
UNDER CONTRACT F30602-73-C-0348

DEVELOPED BY

GENERAL RESEARCH CORPORATION
P. O. BOX 3587
SANTA BARBARA
CALIFORNIA 93105

```
STARTER INITIALIZATION...

KNOWN MODULE DESCRIPTOR BLOCKS...

     MODULE     TEXT     PARENT          PROC PROBE        EXEC FIRST  WORD                                    DS   PARMS DIRECT COMP
NO.   NAME      NAME      NAME    TYPE SCOPE LINS STMTS    STMTS EXEC PAIRS  TOKS  SIMS  SLTS  DMTS  DDPS  BLKS  IN OUT CODE TOT DDP
---------------------------------------------------------------------------------------------------------------------------------
  1 *MOMAIN* *MOJAVS* *MOMAIN* PROG EXT     0    25        1    17   260   203     0    14     0     1     2    0   0 NO    0   0
---------------------------------------------------------------------------------------------------------------------------------
                               TOTAL        0    25        1                                          1

LIBRARY INFORMATION--RUN OF 0508
   LIBRARY     TYPE      DATE    TIMES    LAST     TOTAL       LLINKS      LIBRARY    LIBRARY
NO. NAME       ACCESS    CREATED ALTERED  ALTERED  WORDS       USED        MODULES    FRAGMENTS
-------------------------------------------------------------------------------------------
 1 LIBRARYJ    READ      0508      1      0508     4520         16          1          9
 2 SCRATCH     NEW       0508      1      0508       20          1          0          3
-------------------------------------------------------------------------------------------

COMMAND IS...
                    JAVSTEXT=*MOJAVS*.

JAVSTEXT <*MOJAVS*> IS NOW ACTIVE

COMMAND IS...
                    FOR JAVSTEXT.

THE FOLLOWING MODULES HAVE BEEN QUEUED FOR PROCESSING BY SUBSEQUENT COMMANDS
     *MOMAIN*(*MOJAVS*)

MODULE SELECTED IS <*MOMAIN*> OF JAVSTEXT <*MOJAVS*>

COMMAND IS...
                    INSTRUMENT.MODE=DIRECTIVES.

COMMAND IS...
                    INSTRUMENT.

JOVIAL AUTOMATED VERIFICATION SYSTEM *** INSTRUMENTATION ***

OPTIONS IN EFFECT . . .
                              DD-PATH PROBE =          PROBE
                              MODULE  PROBE =          PROBM
                              TEST    PROBE =          PROBI

INSTRUMENTING DIRECTIVES                          OF MODULE <*MOMAIN*> OF JAVSTEXT <*MOJAVS*>

COMMAND IS...
                    END FOR.

COMMAND IS...
                    PRINT. JAVSTEXT = *MOJAVS*  INSTRUMENTED = ALL.
```

```
JAVBTEXT (*NOJAVS*) LISTING

 NO.  LVL          STATEMENT                                                                    DD-PATHS          CONTROL

   1 ( 0)            START
   2 ( 0)            $
   3 ( 0)            TABLE MATRIX R 100 $
   4 ( 1)              BEGIN
   5 ( 1)              ITEM LINK I 24 S $
   6 ( 1)              ITEM XX I 24 S $
   7 ( 1)              ITEM YY I 24 S $
   8 ( 1)              END
   9 ( 0)            ARRAY PTR S I 24 S $
  10 ( 0)            ARRAY LOCNT S I 24 S $
  11 ( 0)            ARRAY HICNT S I 24 S $
  12 ( 0)            ARRAY LOBND S I 24 S $
  13 ( 0)            ARRAY HIBND S I 24 S $
  14 ( 0)            '' DECLARATIONS FOR ASSERTION INDEX VARIABLES ''
  15 ( 0)            ITEM II I 24 S $
  16 ( 0)            ITEM KK I 24 S $
  17 ( 0)            II = 0 $                                                                   (  1)
  18 ( 0)            '' NEED AN EXECUTABLE STMT ''
  19P( 0)          '', LOOP   (  KK  )  (  0  ... 5  -  1  )  ''
  19P( 0)          ITEM JAVS'ASSERT H 72 $
  19P( 0)          MONITOR JAVS'ASSERT $
  19P( 0)          BEGIN
  19P( 0)          KK = 0 $
  19P( 0)          J00190.
  20P( 0)          '', CHAIN  (  II  )  INIT   (  PTR ($  KK  $)  )  NEXT   (  LINK
                     ($  II  $)  )  END  (  II  EQ  -  1  )  EMPTY  (  PTR ($  KK
                   $)  EQ  -  1  )  COUNT  (  LOCNT  ($  KK  $)  ... HICNT  ($  KK
                   $)  )  BOUNDS  (  LOBND  ($  KK  $)  ... HIBND  ($  KK  $)  )  ''
  20P( 0)          IF NOT (PTR ($ KK $) EQ - 1) $
  20P( 0)          BEGIN
  20P( 0)          ITEM ASSCNT I 24 S $
  20P( 0)          ASSCNT = 0 $
  20P( 0)          II = PTR ($ KK $) $
  20P( 0)          J00200.
  20P( 0)          IF NOT (LOBND ($ KK $) LQ II LQ HIBND ($ KK $)) $
  20P( 0)          BEGIN
  20P( 0)          JAVS'ASSERT = 35H(BOUNDS ASSERTION ERROR AT STMT   20) $
  20P( 0)          GOTO J00201 $
  20P( 0)          END
  20P( 0)          ASSCNT = ASSCNT + 1 $
  21P( 0)          '', VALUE  (  XX  ($  II  $)  )  (  -  11  ... 1  ,  3  ,  9
                   ... 18  )  ''
  21P( 0)          IF NOT ((- 11 LQ XX ($ II $) LQ 1) OR (XX ($ II $) EQ 3) OR ($ LQ XX ($
                   II $) LQ 18)) $
  21P( 0)          JAVS'ASSERT = 34H(VALUE ASSERTION ERROR AT STMT   21) $
```

```
22P( 0)          '', VALUE   (   YY  ($  II  $)  )   NOT (   1   ,   4   ... 7   )   ''
22P( 0)          IF ((YY ($ II $) EQ 1) OR (4 LQ YY ($ II $) LQ 7)) $
22P( 0)          JAVS'ASSERT = 34H(VALUE ASSERTION ERROR AT STMT   22) $
23P( 0)          '', ENDCHAIN    ''
23P( 0)          II = LINK ($ II $) $
23P( 0)          IF NOT (II EQ - 1) $
23P( 0)          GOTO J00200 $
23P( 0)          J00201.
23P( 0)          IF NOT (LOCNT ($ KK $) LQ ASSCNT LQ HICNT ($ KK $)) $
23P( 0)          JAVS'ASSERT = 34H(COUNT ASSERTION ERROR AT STMT   23) $
23P( 0)          END
24P( 0)          '', ENDLOOP ''
24P( 0)          KK = KK + 1 $
24P( 0)          IF NOT (KK GR 5 - 1) $
24P( 0)          GOTO J00190 $
24P( 0)          END
25 ( 0)              TERM $
```

```
COMMAND IS...
                 PUNCH, JAVSTEXT = *NOJAVS*, INSTRUMENTED = ALL,

COMMAND IS...
                 END.
```

JAVS WRAPUP...

KNOWN MODULE DESCRIPTOR BLOCKS...

NO.	MODULE NAME	TEXT NAME	PARENT NAME	TYPE	PROC SCOPE	PROBE LINS	PROBE STMTS	EXEC STMTS	FIRST EXEC	WORD PAIRS	TOKS	SYMS	SLTS	DMTS	DDFS	DS BLKS	PARMS IN	PARMS OUT	DIRECT CODE	COMP TOT	COMP DDP
1	*NOMAIN*	*NOJAVS*	*NOMAIN*	PROG	EXT	57	25	1	17	260	203	0	14	0	1	2	0	0	NO	0	0
				TOTAL		57	25	1							1						

LIBRARY INFORMATION--RUN OF 0508

NO.	LIBRARY NAME	TYPE ACCESS	DATE CREATED	TIMES ALTERED	LAST ALTERED	TOTAL WORDS	LLINKS USED	LIBRARY MODULES	LIBRARY FRAGMENTS
1	LIBRARIJ	READ	0508	1	0508	4520	16	1	9
2	SCRATCH	NEW	0508	1	0508	5520	19	1	11

JOVIAL AUTOMATED VERIFICATION SYSTEM

VERSION 2.2 06-03-77

SPONSORED BY

AIR FORCE SYSTEMS COMMAND
ROME AIR DEVELOPMENT CENTER
GRIFFISS AIR FORCE BASE, NEW YORK 13441
UNDER CONTRACT F30602-73-C-0348

DEVELOPED BY

GENERAL RESEARCH CORPORATION
P. O. BOX 3587
SANTA BARBARA
CALIFORNIA 93105

SNUMB = 87817, ACTIVITY # = 03, REPORT CODE = 74, RECORD COUNT = 000118

```
8751T 03  05-08-78   19.963    COMPOOL COMPILATION OF PROOL          JOCIT VERSION 042275              PAGE      1

    ALTER NO      ****5****1****5****2****5****3****5****4****5****5****5****6****5****7****5****8                ANSC

  1               ''.JAVSTEXT PRCMPL PRESET = COMPOOL FOR PROBE ANALYSIS DURING EXECUTION 0319NB62
  2               ''                                                                      0319NB63
  3               START $                                                                 0319NB64
  4                       DEFINE INTG '' I 24 S '' $                                      DEFINS 2
  5                       DEFINE HLL '' H 4 '' $                                          DEFINS 3
  6                       DEFINE DINTG '' I 48 S '' $                                     DDEFINS2
  7                       DEFINE DHLL '' H 8 '' $                                         DDEFINS3
  8               DEFINE MAXDDP ''2000'' $        ''MAXIMUM DD-PATHS SUMMARIZED ''        0319NB67
  9               DEFINE MAXMOD ''100'' $         ''MAXIMUM MODULES SUMMARIZED ''         0319NB68
 10               PROC PROBM (MODNAM,JAVTXT,NDDPS) $ ''MODULE INVOCATION CALL ''          0319NB69
 11                 BEGIN                                                                 0319NB10
 12                 ITEM MODNAM DHLL $           ''MODULE NAME ''                         0319NB11
 13                 ITEM JAVTXT DHLL $           ''JAVSTEXT NAME ''                       0319NB12
 14                 ITEM NDDPS INTG $            ''NUMBER OF DD-PATHS IN MODULE ''        0319NB13
 15                 END                                                                   0319NB14
 16               PROC PROBE(MODNAM,JAVTXT,DDP) $ ''DD-PATH EXECUTION CALL ''             0319NB15
 17                 BEGIN                                                                 0319NB16
 18                 ITEM MODNAM DHLL $           ''MODULE NAME ''                         0319NB17
 19                 ITEM JAVTXT DHLL $           ''JAVSTEXT NAME ''                       0319NB18
 20                 ITEM DDP INTG $              ''DD-PATH NUMBER ''                      0319NB19
 21                 END                                                                   0319NB20
 22               PROC PROBI (TESNAM,TFLAG) $    ''TEST START/TERMINATION CALL ''         0319NB21
 23                 BEGIN                                                                 0319NB22
 24                   ITEM TESNAM H 8  $                                                  0319NB23
 25                 ITEM TFLAG INTG $            ''TEST START/TERMINATION FLAG ''         0319NB24
 26                 END                                                                   0319NB25
 27                 PROC PROBD(LINE,FLAG) $                                               0319NB26
 28                   BEGIN                                                               0319NB27
 29                   ITEM LINE H 80 $                                                    0319NB28
 30                   ITEM FLAG INTG $                                                    0319NB29
 31                   END                                                                 0319NB30
 32               PROC DATIM(IDATE,ITIME) $      ''SYSTEM DATE CALL ''                    0319NB31
 33                                              ''SYSTEM TIME-OF-DAY CALL ''             0319NB32
 34               PROC PTIME (TIME)$             ''SYSTEM ELAPSED TIME CALL ''            0319NB33
 35               COMMON PROBEC $                ''COMMON BLOCK FOR PROBE PROCEDURES ''   0319NB34
 36                 BEGIN                                                                 0319NB35
 37                  FILE AUDIT B 1 R 8 V(OK) V(X1) V(X2) V(X3) V(EOF) 08 $
 38               ITEM CLOSED INTG P 0 $  ''CLOSE FLAG ON AUDIT FILE ''
 39                 ITEM DATE DHLL $             ''DATE ''                                0319NB37
 40                 ITEM NDDPP INTG P 0 $        '' NUMBER OF DDPATHS PROBED''            0319NB38
 41                 ITEM NMODI INTG P 0 $        ''NUMBER OF MODULES INVOKED''            0319NB39
 42                 ITEM NMODS INTG P 0 $        ''NUMBER OF MODULES''                    0319NB40
 43                 ITEM NTESTS INTG P 0 $       ''NUMBER OF TESTS''                      0319NB41
 44                 ITEM OPENED INTG P 0 $       ''OPEN FLAG ON FILE AUDIT''              0319NB42
 45                 ITEM PROBEN INTG P 0 $       ''NUMBER OF PROBES EXECUTED''            0319NB43
 46                 ITEM TESTN INTG P 0 $        ''NUMBER OF TESTS EXECUTED''             0319NB44
 47                 ITEM TIMEO F P 0.0 $                                                  0319NB45
 48                 ITEM DELTAT F P 0.0 $                                                 0319NB46
 49                 ITEM TOD DHLL $              ''TIME-OF-DAY ''                         0319NB47
 50                 ITEM TIME F $                ''ELAPSED TIME ''                        0319NB48
```

```
ALTER NO      ****5****1****5****2****5****3****5****4****5****5****5****6****5****7****5****8          ANSC

  51              ITEM MXMOD INTG P MAXMOD $   ''MAXIMUM MODULES SUMMARIZED ''          0319WB49
  52              ITEM NMOD INTG P 0 $         ''NUMBER OF MODULES SUMMARIZED ''       0319WB50
  53              ITEM MXDDP INTG P MAXDDP $   ''MAXIMUM DD-PATHS SUMMARIZED ''        0319WB51
  54              ITEM NDDP INTG P 0 $         ''NUMBER OF DD-PATHS SUMMARIZED ''      0319WB52
  55              ITEM PROBEF INTG P 2 $       ''PROBE CAPTURE FLAG ''                 0319WB53
  56              TABLE PRBUFR R 1 S 8 $       ''TEST BUFFER ''                        0319WB54
  57                BEGIN                                                             0319WB55
  58                ITEM RECORD NLL 0 0 $      ''TEST RECORD TYPE ''                   0319WB56
  59                ITEM NAME1 DNLL 1 0 $      ''FIRST NAME IN RECORD ''               0319WB57
  60                ITEM NAME2 DNLL 3 0 $      ''SECOND NAME IN RECORD ''              0319WB58
  61                ITEM NAME3 DNLL 5 0 $      ''THIRD NAME IN RECORD ''               0319WB59
  62                ITEM INTG1 INTG 1 0 $      ''FIRST INTEGER IN RECORD ''            0319WB60
  63                ITEM INTG2 INTG 2 0 $                                              0319WB61
  64                ITEM INTG3 INTG 3 0 $                                              0319WB62
  65                ITEM INTG4 INTG 4 0 $                                              0319WB63
  66                ITEM INTG5 INTG 5 0 $                                              0319WB64
  67                ITEM INTG6 INTG 6 0 $                                              0319WB65
  68                ITEM INTG7 INTG 7 0 $                                              0319WB66
  69                ITEM REAL1 F 1 0 $         ''FIRST REAL IN RECORD ''               0319WB67
  70                ITEM REAL2 F 2 0 $                                                 0319WB68
  71                ITEM REAL3 F 3 0 $                                                 0319WB69
  72                ITEM REAL4 F 4 0 $                                                 0319WB70
  73                ITEM REAL5 F 5 0 $                                                 0319WB71
  74                ITEM REAL6 F 6 0 $                                                 0319WB72
  75                ITEM REAL7 F 7 0 $                                                 0319WB73
  76                ITEM CARD H 40 1 0 $       ''HALF CARD IMAGE''                     0319WB74
  77                END                                                               0319WB75
  78              TABLE MODSUM R MAXMOD S 0 $  ''MODULE SUMMARY WORKSPACE ''           0319WB76
  79                BEGIN                                                             0319WB77
  80                ITEM MOD DNLL 0 0 $        ''MODULE NAME ''                        0319WB78
  81                ITEM TXT DNLL 2 0 $        ''JAVSTEXT NAME ''                      0319WB79
  82                ITEM MBASE INTG 4 0 $      ''BASE IN DD-PATH WORKSPACE ''          0319WB80
  83                ITEM MINV INTG 5 0 $       ''NUMBER OF INVOCATIONS ''              0319WB81
  84                ITEM MRET INTG 6 0 $       ''NUMBER OF RETURNS ''                  0319WB82
  85                ITEM MDDP INTG 7 0 $       ''NUMBER OF DD-PATHS ''                 0319WB83
  86                ITEM MTIM F 8 0 $          ''ELAPSED TIME IN MODULE AND UNDERLINGS''0319WB84
  87                END                                                               0319WB85
  88              TABLE DDPSUM R MAXDDP S 1 $  ''DD-PATH SUMMARY WORKSPACE ''          0319WB86
  89                BEGIN                                                             0319WB87
  90                ITEM DEX INTG 0 0 $        ''DD-PATH EXECUTION COUNT ''            0319WB88
  91                END                                                               0319WB89
  92              END                                                                 0319WB90
  93            TERM $                                                                0319WB91

** NO DIAGNOSTIC MESSAGES**
```

PROGRAM SUMMARY

SIMDEFS ADDR
NONE

COMMONS SIZE
PROBEC 005564

SIMREFS
NONE

SHUMB = 07B1I, ACTIVITY # = 08, , REPORT CODE = 74, RECORD COUNT = 000086

```
8751I 04  05-08-78   19.966     JOVIAL COMPILATION OF ......          JOCIT VERSION 042275              PAGE      1

   ALTER NO      ****5****1****5****2****3****3****5****4****5****5****5****6****5****7****5****8                ANS1

    1                 START
    2                 $
    3                 TABLE MATRIX R 100 $
    4                   BEGIN
    5                   ITEM LINK I 24 S $
    6                   ITEM XX I 24 S $
    7                   ITEM YY I 24 S $
    8                   END
    9                 ARRAY PTR 5 I 24 S $
   10                 ARRAY LOCNT 5 I 24 S $
   11                 ARRAY HICNT 5 I 24 S $
   12                 ARRAY LOBND 5 I 24 S $
   13                 ARRAY HIBND 5 I 24 S $
   14                 '' DECLARATIONS FOR ASSERTION INDEX VARIABLES ''
   15                 ITEM II I 24 S $
   16                 ITEM KK I 24 S $
   17                 II = 0 $
   18                 '' NEED AN EXECUTABLE STMT ''
   19               ''. LOOP    (   KK  )   (   0   ... 5   -   1   )   ''
   20               ITEM JAVS'ASSERT N 72 $
   21               MONITOR JAVS'ASSERT $
   22               BEGIN
   23               KK = 0 $
   24               J00190.
   25               ''. CHAIN   (   II  )   XBIT    (   PTR ($  KK  $)  )   NEXT    (   LINK
   26                   ($  II  $)  )   END (   II  EQ  -   1   )   SNDTY   (   PTR ($  KK
   27               $)  EQ  -   1   )   COUNT   (   LOCNT   ($  KK  $)  ... HICNT   ($  KK
   28               $)  )   BOUNDS  (   LOBND   ($  KK  $)  ... HIBND   ($  KK  $)  )   ''
   29               IF NOT (PTR ($ KK $) EQ - 1) $
   30               BEGIN
   31               ITEM ASSCNT I 24 S $
   32               ASSCNT = 0 $
   33               II = PTR ($ KK $) $
   34               J00200.
   35               IF NOT (LOBND ($ KK $) LQ II LQ HIBND ($ KK $)) $
   36               BEGIN
   37               JAVS'ASSERT = 35H(BOUNDS ASSERTION ERROR AT STMT   20) $
   38               GOTO J00201 $
   39               END
   40               ASSCNT = ASSCNT + 1 $
   41               ''. VALUE   (   XX  ($  II  $)  )   (   -   11  ... 1   ,   3   ,   9
   42               ... 14  )   ''
   43               IF NOT ((- 11 LQ XX ($ II $) LQ 1) OR (XX ($ II $) EQ 3) OR ($ LQ XX ($
   44               II $) LQ 14)) $
   45               JAVS'ASSERT = 34H(VALUE ASSERTION ERROR AT STMT   21) $
   46               ''. VALUE   (   YY  ($  II  $)  )   NOT (   1   ,   4   ... 7   )   ''
   47               IF ((YY ($ II $) EQ 1) OR ($ LQ II ($ II $) LQ 7)) $
   48               JAVS'ASSERT = 34H(VALUE ASSERTION ERROR AT STMT   22) $
   49               ''. ENDCHAIN    ''
   50               II = LINK ($ II $) $
```

```
8751T 04  05-08-78   19.966     JOVIAL COMPILATION OF ......          JOCIT VERSION 042275              PAGE      2

    ALTER NO      ****5****1****5****2****5****3****5****4****5****5****5****6****5****7****5****8                ANZ1

    51            IF NOT (II EQ - 1) $
    52            GOTO J00200 $
    53            J00201.
    54            IF NOT (LOCNT ($ KK $) LQ ABSCNT LQ HICNT ($ KK $)) $
    55            JAVS'ASSERT = J4H(COUNT ASSERTION ERROR AT STMT   23) $
    56            END
    57            ''. ENDLOOP ;)
    58            KK = KK + 1 $
    59            IF NOT (KK GR 5 - 1) $
    60            GOTO J00190 $
    61            END
    62               TERM $
 COMPOOL    ERPOOL       VERSION    050878

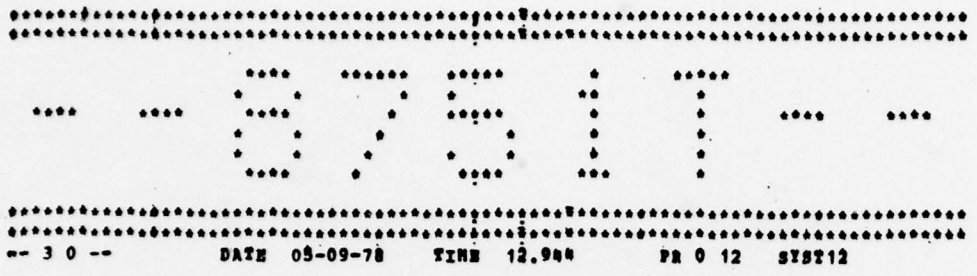
** NO DIAGNOSTIC MESSAGES**
```

PROGRAM SUMMARY

SYMDEFS	ADDR
.....	000560

COMMONS	SIZE
NONE	

SYMREFS
JOVMD.
JOVMOV
.JEXIT


-- 3 0 --
DATE 09-09-78
TIME 12.944
PR 0 12
SYST12

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.