AD-A064 872 GENERAL RESEARCH CORP SANTA BARBARA CALIF F/G 9/2
JOVIAL AUTOMATED VERIFICATION SYSTEM.(U)
DEC 78 C GANNON F30602-77-C-0115
UNCLASSIFIED RADC-TR-78-247 NL

1 OF 2
AD
A 064 872

LEVEL II

RADC-TR-78-247
Final Technical Report
November 1978

JOVIAL AUTOMATED VERIFICATION SYSTEM

General Research Corporation

C. Gannon

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

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

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-78-247 has been reviewed and is approved for publication.

APPROVED: Frank S. LaMonica

FRANK S. LAMONICA
Project Engineer

APPROVED: Wendall C. Bauman

WENDALL C. BAUMAN, COL, 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 (ISIE) 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-78-247	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): JOVIAL AUTOMATED VERIFICATION SYSTEM.	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report. Jun 1977 - Jul 1978.
	6. PERFORMING ORG. REPORT NUMBER: N/A
7. AUTHOR(s): C. Gannon	8. CONTRACT OR GRANT NUMBER(s): F30602-77-C-0115
9. PERFORMING ORGANIZATION NAME AND ADDRESS: General Research Corporation, P.O. Box 6770, Santa Barbara CA 93111	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 63728F, 25310203
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (ISIE), Griffiss AFB NY 13441	12. REPORT DATE: December 1978
	13. NUMBER OF PAGES: 176
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: Frank S. LaMonica (ISIE)

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

Computer Software	JAVS
Software Testing	JOVIAL J3
Software Verification	Automated Verification System
Software Documentation	

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

JAVS, for JOVIAL Automated Verification System, provides measurements of testing thoroughness, retesting assistance, and automated software documentation for JOVIAL J3 programs.

This report describes the design, implementation and testing of a new JAVS syntax analyzer. Background information regarding all JAVS contracts is provided in this report, as are procedures for installing the complete JAVS (Cont'd)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

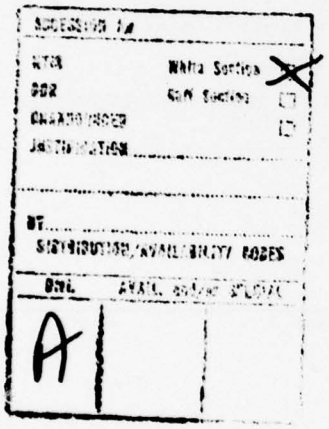
402 754

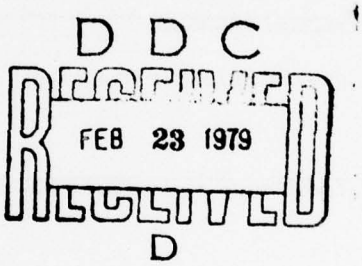
Item 20 (Cont'd)

software package.

Familiarity with the JOVIAL language and with software verification terminology is assumed.

LEVEL II





CONTENTS

Pages 5-39 thru 5-42 are left blank intentionally

ILLUSTRATIONS

Illustrations (Continued)

EVALUATION

The primary purpose of this effort was to develop an efficiency enhanced version of the JOVIAL Automated Verification System (JAVS). The enhancements provided include a re-written syntax analyzer component which has significantly reduced the run-time primary memory requirement, resulting in a system with a higher throughput characteristic. The new system was installed and acceptance tested on the RADC H6180/GCOS Computer System. Following acceptance testing, the system was released to Hq SAC, Offutt AFB, Nebraska.

Frank S LaMonica

FRANK S. LA MONICA
Project Engineer

1 INTRODUCTION

JAVS, for JOVIAL Automated Verification System, provides measurements of testing thoroughness, retesting assistance, and automated software documentation for JOVIAL J3 programs.

This report describes the design, implementation and testing of a new JAVS syntax analyzer. Background information regarding all JAVS contracts is provided in this report, as are procedures for installing the complete JAVS software package.

Familiarity with the JOVIAL language and with software verification terminology is assumed.

The primary purpose of this contract was to design and implement a new JOVIAL J3 syntax analyzer for the JOVIAL Automated Verification System (JAVS) with a main memory requirement of fewer than 60,000 words on the HIS 6180. This design allowed a number of former JAVS constraints which dealt with the syntax and semantics of JOVIAL J3 programs to be removed. Several minor enhancements were made to the control path analysis in the course of designing the interface of this new syntax analyzer with the rest of JAVS.

This report contains a brief overview of JAVS capabilities and background information on the evolution of JAVS. Section 2 describes the newly implemented syntax analyzer. Section 3 contains the procedures and results of acceptance testing for the syntax analyzer and other software components which it affected. Section 4 presents recommendations for additional capabilities and modifications.

Appendix A contains instructions for installing the JAVS software at sites other than RADC, Griffiss AFB, New York, and SAC Headquarters, Offutt AFB, Nebraska. Appendices B and C of this report contain the updated pages to the JAVS User's Guide[1] and the JAVS Reference Manual[2]. The changed pages are provided in this report to ensure their distribution to holders of the 1976 edition of the referenced documents (since they will not be reprinted in their entirety by the Government under this contract).

1.1 JAVS CAPABILITIES

JAVS is a software tool to be used during the testing of JOVIAL J3 programs to aid in recognizing unexercised program paths, assist in developing additional test cases so as to improve test execution coverage, and automatically document the program. JAVS' documentation features can also be applied during software development and debugging stages as long as complete control structures are provided.

Program verification is based on analyzing control flow structures, instrumenting the program by inserting software probes to measure testing coverage during execution, and comprehensive reports which pinpoint unexercised paths in the program structure. Retesting guidance is provided identifying program paths leading to untested program areas and by reports describing module and symbol interaction.

As a testing tool, JAVS provides coverage and trace reports showing program behavior during a test. Test performance coverage reports showing statements and/or decision outways (conditional branches) can be obtained on a per-module, per-test-case, and per-test-run basis. These reports allow the user to focus on untested modules, program paths, and statements. Tracing can be performed, at user option, to show module invocations and returns or to show which outway was taken for each conditional operation in the program. In addition, the user can trace "important" events, such as overlay link loading, by invoking one of the JAVS data collection routines.

If the testing target is determined to be a set of modules which received little or no coverage during the test execution, JAVS reports can be obtained to list all invocations (and the statement numbers of the calls) to the modules and to show the modules' interactions with the rest of the system in terms of calling trees and interaction matrices. If the testing target is a segment of code within a module, the user can request a JAVS report showing the statements that lead up to the target. Armed with this "reaching set" report, the user can spot key variables whose values affect the flow through the program paths and locate all instances of the variables in the system-wide cross reference.

Retesting may necessitate code changes in some of the modules in the system to remove dead code or coding errors found during the test analysis. To facilitate determining all modules in the system which could be affected by the code changes, a JAVS report will show the interaction between the selected set of modules and the rest of the system.

JAVS uses a data base to store information about the test program. The availability and management of this information form the basis for a variety of services in addition to the primary task of testing assistance. Computer program documentation, debugging through JAVS computation directives, and reports useful for code optimization are the major side benefits of JAVS.

Computer documentation requirements for the Air Force typically specify flow charts and lists of program variables and constants. In the JAVS development and implementation contracts these requirements were replaced by specifying certain JAVS reports, i.e., self-documentation. It was found that the module listings (enhanced by indentation and identification of decision points), module control flow pictures, module invocation reports (showing formal and actual parameter lists), module interdependence reports, and a cross-reference report for each JAVS component are more meaningful documentation and are generated automatically by JAVS.

Software development can be assisted by using JAVS to document and test the system as it is built. To aid in data flow analysis and checking of array sizes and variable execution values, JAVS offers computation directives. The directives are a special form of JOVIAL comment, recognized by JAVS and expanded into executable code (using the JOVIAL monitor statement) during the instrumentation phase. The user can check logic expressions with an ASSERT directive, check boundaries of selected variables with an EXPECT directive, and turn on and off the standard monitor tracing with TRACE and OFFTRACE directives. Code optimization is aided by the post-test reports, which show the

number of times each statement is executed and the execution time spent in the modules (in milliseconds of central processor time). Modules which are never called and should be removed are listed in another JAVS report.

Testing coverage results indicate what parts of the program were executed. It is up to the user to determine if the program's output is reasonable. One JAVS post-test analysis report lists the execution coverage during the test run in terms of the percentage of decision outways taken. A decision outway (decision-to-decision path, or DD-path) is the set of statements executed as the result of the evaluation of a predicate (conditional operation). A good standard for the testedness of a program is to exercise every decision outway at least once. This level of testing is more rigorous than testing every program statement at least once. However, it should be emphasized that certain combinations of DD-paths may contain errors which are not detected in merely executing each outway one time.

JAVS reads the user's JOVIAL program as data and performs syntax, structural, and instrumentation analyses on the source code. JAVS communicates with the user through a command language and utilizes a data base to store the information about the program. The user is provided with an instrumented file of the selected program modules with which the user supplies test data for execution. The execution results are written to a file from which JAVS' post-test analyzer issues execution tracing and coverage reports.

Six functional processes, in addition to execution with test data, make up the substance of software validation provided by JAVS. The organization of JAVS is defined by these six tasks. To reduce the burden of the user, JAVS exists as an overlay program at RADC with a macro command language supplementing a large, versatile standard command language. Figure 1.1 shows a block diagram of the processes and the four macro commands (BUILD LIBRARY, PROBE, TEST, and DOCUMENT) which drive the processes. The processing steps and their basic functions are listed below:

BASIC, Source Text Analysis: Source text input, lexical analysis, and initial source library creation

STRUCTURAL, Structural Analysis: Structural analysis and execution path identification; library update with structure and path information

INSTRUMENT, Module Instrumentation: Program instrumentation for path coverage analysis and program performance directed by the user; library update with probe test instrumentation

ASSIST, Module Testing Assistance and Segment Analysis: Testing assistance for improved program coverage

DEPENDENCE, Retesting Guidance and Analysis: Retesting requirements analysis for changed modules

TEST EXECUTION: Execution of instrumented code and analysis of directed program performance

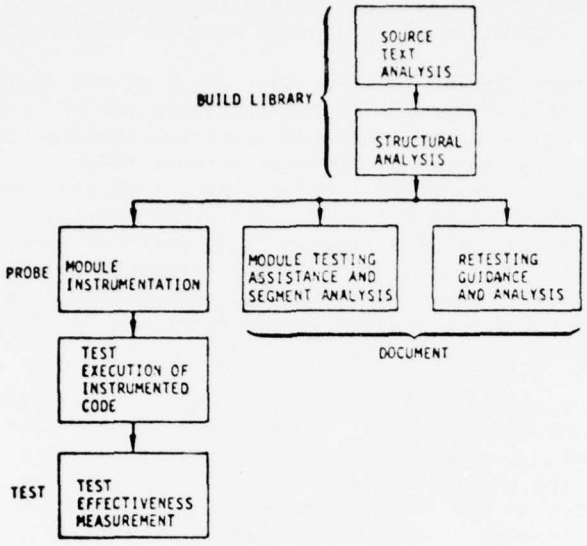


Figure 1.1. JAVS Processing Sequences

ANALYZER, Test Effectiveness Measurement: Detailed analysis of program path coverage; execution traces and summary statistics

The user must provide three major types of input to JAVS: (1) the source code to be tested, (2) a set of commands to direct JAVS processing, and (3) test data for program execution.

1.2 BACKGROUND

JAVS was developed under Air Force contract F30602-73-C-0344 with RADC to engineer workable and practical first-level solutions to the task of automating the measurement of JOVIAL computer program test effectiveness. Other tasks to be included in the test tool were the capabilities of assisting the manual process of test case design and selection and automating certain aspects of software system maintenance. The tool which resulted from that effort in 1973-1975 was a system of six programs which performed the functional processes shown in Fig. 1.1. The common thread between the programs was a database library which contained tables of syntactical, semantic, and structural information describing the user's JOVIAL program. This system required 84,000 words of central memory to execute the largest program (source text analysis) and 56,000 words to execute the smallest. As part of the JAVS acceptance test, all of the JAVS software had to be processed by JAVS itself and demonstrate overall statement execution coverage of 85%.

The syntax analyzer portion of the source text analysis program was developed by GRC's subcontractor, System Development Corporation (SDC), as an extensive modification to the existing GEN1 phase of the SAM-D ED JOVIAL compiler for the UNIVAC 1108. Although this syntax analyzer was fast, it had the following disadvantages, many of which made it inconsistent with JOVIAL J3 constructs:

1. It required 84,000 words to load and execute in the JAVS environment (which is the same for all JAVS functional processors) and, as designed, could not be overlayed.

2. Improper generation of pointers dealing with labels defined in one module and called from another module (within the rules of JOVIAL J3). This constraint was primarily due to the "one-pass" analysis performed on the JOVIAL source.

3. Only one COMPOOL could be analyzed in a single execution of the analyzer.

4. Special characters, such as "<", ":", etc. were considered illegal and blanked out, thereby affecting consistency of the instrumented code with the original source code (which could result in differing execution output) and completeness in the JAVS documentation reports.

5. A JOVIAL CLOSE subprogram (external CLOSE) could not be analyzed without redefining it as a main program.

6. A single prime was ignored; thus 'LOC and other primitives were incorrectly processed without the prime.

7. A comment could not be terminated by a dollar sign.

8. Labels could not be appended to BEGIN statements (they were discarded).

9. Certain keywords, such as PROC, in comments caused erroneous parsing of the text.

10. Nested DEFINE statements would not be expanded properly.

11. The same status variables used in more than one status list would produce numerous unnecessary warnings.

Subsequent effort was directed at JAVS under Air Force Contract F30602-76-C-0233 with RADC. The major tasks were to convert the six programs into a single overlay program, add a macro command language, stress and evaluate JAVS' performance by implementing a systematic software test on a large, complex program not written for Automated Verification System (AVS) testing in mind, correct any JAVS errors found during the effort, and improve the existing testing methodology.[3]

The overlay configuration reduced the main memory requirement only by 3,000 words to 81,000 for complete JAVS analysis and 54,000 for all analyses except syntax analysis. The main advantage in the overlay structure was the ability to incorporate a simple command language using four command keywords: BUILD LIBRARY, DOCUMENT, PROBE, and TEST to perform most of JAVS' activities.

The JAVS "stress and evaluate" activity provided much-needed experience in attempting to determine the value of a path-testing AVS. The fact that JAVS underwent systematic functional and path coverage testing and subsequently was used extensively to analyze "real world" unstructured JOVIAL source code permitted evaluation of the utility of program path testing using functional data.

The outcome of this evaluation was that of the over 28,000 statements (366 modules) comprising the JAVS source, ten errors were found after its delivery. These errors fell into three major categories: structural, design, and logic. Of the structural errors (there were three), one was detected during the coverage self-test, but the code was not corrected before delivery; another structural error manifested itself in the output (i.e., the output was incorrect), but the output was not detected as erroneous; the third was simply one of the untested paths. More thorough path testing would have uncovered this infinite loop error.

The remaining seven errors probably would not have been detected by using JAVS' path-testing capability. Judicious use of JAVS computation directives (especially ASSERT and EXPECT) may have detected the three logic errors. The design errors were primarily in the syntax analyzer, and reconfirmed the widely held belief that extra emphasis should always be placed on the design phase of software development and that functional test data should be designed from the specifications concurrently with the design of the software.

One of the documents delivered under contract F30602-76-C-0233 was the Methodology Report.[4] The section of that document entitled "Application of Systematic Testing Methodology" addresses the role of an AVS (JAVS in particular) in applying the formally defined general testing methodology and provides practical techniques for particular situations. The JAVS evaluation experience provided additional insight into the design of the new syntax analyzer, not only in terms of better understanding of the language specification, but also in terms of designing AVS-testable software. The description and results of acceptance testing for the new syntax analyzer are given in Sec. 3.

1.3 OBJECTIVES OF THE CURRENT CONTRACT

The above background information sets the stage for describing the objectives of the current contract. As previously stated, the primary objective was to design a new syntax analyzer which had a smaller appetite for central memory. Other important considerations, some of which had a bearing on the design, were:

- Eliminate many of the JAVS-imposed restrictions on JOVIAL source
- Report possible structural infinite loops and dead code

- Store comments differently so as to improve certain JAVS reports
- Include loop control variables in the symbol cross reference
- Remove the incorrect generation of control cards ($ in column 1 on Honeywell equipment) in the instrumented source code.
- Allow comments to be imbedded anywhere within statements or between statements
- Permit direct analysis of the executable source text (i.e., without the user's insertion of JAVSTEXT header statements between START-TERM sequences, although this practice is still recommended)
- Construct the syntax analyzer to be easily modified and maintained

1.4 JAVS TECHNICAL REPORTS

The following list of documents describe the current software for JAVS, its utilization and recommended testing methodology.

- JAVS Technical Report: Vol. 1, User's Guide. This report is an introduction to using JAVS in the testing process. Its primary purpose is to acquaint the user with the innate potential of JAVS to aid in the program testing process so that an efficient approach to program verification can be undertaken. Only the basic principles by which JAVS provides this assistance are discussed. These give the user a level of understanding necessary to see the utility of the system. The material on JAVS processing in the report is presented in the order normally followed by the beginning JAVS user. Adequate testing can be achieved using JAVS macro commands and the job streams presented in this guide. The Appendices include a summary of all JAVS commands and a description of JAVS operation at RADC with both sample command sets and sample job control statements. (General Research Corporation CR-1-722, November 1976; available as RADC-TR-77-126, Vol. I; updated as General Research Corporation CR-1-722/1, June 1978. Updated pages available in Appendix B of this report.)
- JAVS Technical Report: Vol. 2, Reference Manual. This report describes in detail JAVS processing and each of the JAVS commands. The Reference Manual is intended to be used along with the User's Guide which contains the machine-dependent information such as job control cards and file allocation. Throughout the Reference Manual, modules from a sample JOVIAL program are used in the examples. Each JAVS command is explained in detail, and a sample of each report produced by JAVS is included with the appropriate command. The report is organized into two major parts: one describing the JAVS system and the other containing the description of each JAVS command in alphabetical order. The Appendices include a complete listing of all error messages directly produced by JAVS processing. (General Research Corporation CR-1-722, November 1976; available as RADC-TR-

77-126, Vol. II; updated as General Research Corporation CR-1-772/1, June 1978. Updated pages available in Appendix C of this report.)

- JAVS Technical Report: Vol. 3, Methodology Report. This report describes the methodology which underlies and is supported by JAVS. The methodology is tailored to be largely independent of implementation and language. The discussion in the text is intended to be intuitive and demonstrative. Some of the methodology is based upon the experience of using JAVS to test a large information management system. A long-term growth path for automated verification systems that supports the methodology is described. (RADC-TR-77-126, Vol. III, April 1977)

- JAVS Computer Program Documentation: Vol. 1, System Design and Implementation. This report contains a description of JAVS software design, the organization and contents of the JAVS data base, and a description of the software for each JAVS component: its function, each of the modules in the component, and the global data structures used by the component. The report is intended primarily as an informal reference for use in JAVS software maintenance as a companion to the Software Analysis reports described below. Included in the appendices are the templates for probe code inserted by instrumentation processing for both structural and directive instrumentation and an alphabetical list of all modules in the system (including system routines) with the formal parameters and data type of each parameter. (GRC, CR-1-782, Vol. I, June 1978)

- JAVS Computer Program Documentation: Vol. 2, Software Analysis. This volume is a collection of computer output produced by JAVS standard processing steps. The source for each component of the JAVS software has been analyzed to produce enhanced source listings of JAVS with indentation and control structure identification, inter-module dependence, all module invocations with formal and actual parameters, module control structure, a cross reference of symbol usage, tree report for each leading module, and report showing size of each component. It is intended to be used with the System Design and Implementation Manual for JAVS software maintenance. The Software Analysis reports, on file at RADC, are an excellent example of the use of JAVS for computer software documentation.

- JAVS Final Report. The final report for the project describes the design, implementation and testing of the JAVS syntax analyzer. Background information regarding all JAVS contracts is provided as well as procedures for installing the complete JAVS software package. This report contains, as appendices, the June 1978 updated pages for the User's Guide and Reference Manual published as RADC-TR-77-126, Vols. I and II, April 1977.

2 JAVS SYNTAX ANALYZER

The function of the syntax analyzer (called JAVS2) is to read the JOVIAL source text stream and generate the lexical and semantic information needed by subsequent phases of JAVS. Each START-TERM sequence is treated as a separate unit and may be either a COMPOOL text or executable text. The sequence of text characters is separated into JOVIAL symbols, the symbols are collected into statements, and the statements are grouped into modules. Symbols are classified according to type, and tables of identifiers essential to structural analysis are built.

The source text presented to JAVS is assumed to be free of syntax errors; i.e., it must have been successfully processed by the JOVIAL compiler without errors. Since the structural properties of executable text are wholly contained within the START-TERM sequence, JAVS2, unlike the JOVIAL compiler, does not require that the text for a referenced COMPOOL be processed together with the executable text.

The new JAVS2 code uses substantially less central memory (29,000 words), creates only that information about symbols needed for structural analysis (with a corresponding savings in auxiliary storage), and fits into the overlay structure of the remainder of the JAVS system. The overall design has this JAVS software component separated into three distinct processes:

1. An initialization process which sets initial data into JAVS2 data structures.

2. A text-recognition process which reads the source text, identifies symbols, statements, and modules, expands text for DEFINES, constructs initial entries in the permanent tables Module Descriptor Block (MDB), Statement Blocks (SB), and Statement Descriptor Blocks (SDB), and constructs temporary tables necessary for the text analysis process which follows.

3. An analysis process which uses the tables prepared by the text-recognition processing to construct the permanent tables Symbol Table Blocks (STB) and Symbol Locator Blocks (SLT) as well as final entries in the MDB, SB, and SDB.

The text-recognition process is concerned primarily with analyzing the text stream for JOVIAL syntactical constructs (i.e., symbols, statements, and modules); the analysis process is concerned with transforming JOVIAL syntactical constructs into the JAVS structural description, a form which defines the basic structural properties of the source text.

The initialization process (1) executes once for a single file of text. The text recognition process (2) and analysis process (3) execute once for each START-TERM sequence in the file of text. With this design, each process can be a secondary overlay in the JAVS system memory layout.

In implementing this design, the JAVS2 code retains characteristics of other JAVS components: it is highly modular and well-structured, makes use of

the JAVS data manager for both permanent and temporary tables, and utilizes the JAVS nucleus support routines for basic services. Syntactic units are to the same specifications as those currently processed by the JOCIT JOVIAL/J3 computer, and the syntax analysis uses a statement recognition algorithm which identifies well-defined statement initiation and statement termination constructs in context.

2.1 EFFECT ON USERS

2.1.1 Resource Requirements

Execution of JAVS with the new syntax analyzer requires 53,000 words of primary memory and 5% less secondary memory for the database library. Processing time and file space estimations are provided in the JAVS User's Guide[1]. JAVS syntax and structural analyses require approximately 50%-70% additional central processor time than was required by the older version. The extra CP time is primarily due to using the JAVS database manager for information storage and to the multi-pass design of the syntax analyzer. One of the reasons for a multi-pass syntax analyzer is to properly handle all references to global labels.

2.1.2 Constraints

The former JAVS constraints listed in Sec. 1.2 have all been removed. In addition, the following structural rules are no longer required: the executable text must be a compound statement (JOVIAL makes this a requirement only for PROCs); a declaration statement with an END (e.g., an ARRAY or TABLE declaration) must not be located immediately preceding a TERM statement.

The following implementation constraints are the current ones which must be observed during source text processing:

1. Each module placed on the same library must have a unique module name for a given JAVSTEXT name. For this purpose, only the first eight characters of any name are used. The first six characters should be unique (a compiler restriction).

2. A PROC must contain at least one executable statement (e.g., RETURN).

3. Statement labels in direct code are not analyzed. A reference to such a label in JOVIAL code is treated as a reference to an external undefined label.

4. The maximum number of nested modules is 150.

5. The maximum number of unique symbols (names and constants) is 4,096.

6. A basic element may not exceed 500 characters (does not include literals).

7. A JOVIAL symbol may not exceed 4,095 characters.

8. A comment, if saved, will be truncated to the maximum JOVIAL symbol length if it exceeds that length.

9. BASIC guarantees that saved comments terminate with a double prime (i.e., a double prime will be generated).

10. Only the first 72 columns of source text line are analyzed.

11. COMPOOLs must have a JAVSTEXT directive stating the PRESET type.

12. A statement name following a TERM (main program only) will not determine the first executable statement.

2.1.3 Error Messages

The former syntax analyzer had a repertoire of approximately a hundred error messages, many of which would never occur if the user's source code was properly compiled. The complete list of error messages emanating from the new syntax analyzer are listed below:

Error Number	Explanation
1	Basic element contains too many characters. Element truncated in saved text. Resubmit with corrected text.
2	Illegal internal text character. System error.*
3	Illegal external text character. System error.*
4	Recursive DEFINE reference. Reference partially expanded. Resubmit with recursive DEFINE definition corrected.
5	JOVIAL symbol too long. Symbol truncated in saved text. Resubmit with corrected text.
6	Too many symbols (names and constants) in text. Fatal error. Resubmit with text partitioned into more START-TERM sequences.
7	Module nesting exceeds limit. Change module nesting structure.
8	Too many ENDS. Resubmit with corrected text.
9	Loop in basic element analysis. System error.*
10	Loop in internal text character analysis. System error.*
11	Loop in JOVIAL element analysis. System error.*
12	Loop in external character analysis. System error.*

*System errors should be reported with output listing card images processed.

2.1.4 Syntax Analysis Commands

Several JAVS syntax analysis commands have been removed. These are:

BASIC, ERRORS = ON/OFF/LIMIT/TRACE.

BASIC, SYMBOLS = ON/OFF/PARTIAL.

BASIC, TEXT = PRESET/COMPUTE/BOTH/JAVSTEXT.

leaving only three commands. These are:

BASIC, CARD IMAGES = ON/OFF.

BASIC, COMMENTS = ON/OFF.

BASIC, DEFINES = ON/OFF.

where the default values are underlined. The JAVS macro command BUILD LIBRARY specifies the default values for syntax analysis.

When the DEFINES option is ON, JAVS expands the DEFINE references (to any level of nesting) but leaves the DEFINE declaration in the text. The former syntax analyzer removed the DEFINE declaration after expansion, making documentation ambiguous. Compilation of the instrumented source text will be unaffected by the presence of the DEFINE and the expansion.

2.1.5 JAVS Output

The computer listing output by the source text analysis process underwent minor change as a result of the new syntax analyzer. The former output is shown in Figs. 2.1 and 2.2. The message:

<module name> (<JAVSTEXT name>) COMPLETED

was written upon completion of each module's analysis, rather than at the end of each START-TERM sequence.

The new syntax analyzer performs an initialization process once, then makes two passes through each START-TERM sequence. One pass reads the source and builds text-recognition tables; the other pass uses the tables and completes the entries. At the end of each START-TERM sequence, the modules in that sequence (called TEXT) are listed. Figures 2.3(a) and (b) show the output: the user's card-image input source with the card count printed at the rightmost column and module and JAVSTEXT identification following the TERM statement.

If the user does not provide the JAVSTEXT identification directive at the beginning of each executable START-TERM, JAVS will assign one using the last JAVSTEXT's name for the module's name (or JAVS0001 if the START-TERM sequence is the first one). If the module is a PROC, its own name will be used. The text's name will be assigned JAVS000i where "i" is the number of the module

```
                    **.JAVSTEXT EXCOMPL PRESET**
                    START$
                         **COMPOOL EXAMPLE CONTAINING PRESET OUTPUT MESSAGES  **
                         COMMON MESAGES
                         BEGIN
                              ITEM MSG1 H 18 P 18H(1JAVS TEST CASE    )$
                              ITEM MSG2 H 18 P 18H(0RESULT LT 4       )$
                              ITEM MSG3 H 18 P 18H(0RESULT EQ 4       )$
                              ITEM MSG4 H 18 P 18H(0RESULT GT 4       )$
                         ITEM MESSAG H 18$
                         END
                     TERM$

EXCOMPL  (EXCOMPL ) COMPLETED
    ********     NO ERRORS WERE FOUND BY JAVS-2  ******



                    **.JAVSTEXT EXPROGM COMPUTE (EXCOMPL)**
                     START$
                         **JOVIAL SIMPLE TEST PROGRAM **
                         DEFINE INTG **I 24 S **$
                         DEFINE HLL ** H 4 **$
                         DEFINE NBYTWD ** 4 **$
                         ITEM ID HLL$
                         ITEM ITER1 INTG$
                         ITEM ITER2 INTG$
                         ITEM ITER1A HLL$
                         ITEM ITER2A HLL$
                         OVERLAY ITER1 = ITER1A$
                         OVERLAY ITER2 = ITER2A$
                         ITEM CARD H 80$
                         FILE READER H 0 R 84 V(OK) V(EOF) TAPE5$
                         FILE PRINTR H 0 R 128 V(OK) V(EOF) TAPE6$
                         MONITOR ID, ITER1A, ITER2A$
                         MESSAG = MSG1$
                         OUTPUT PRINTR MESSAG$
                     GO. INPUT READER CARD$
                         IF READER NO V(EOF)$
                         BEGIN
                              BYTE($0,NBYTWD$) (ID) = BYTE($0,NBYTWD$) (CARD)$
                              BYTE($0,NBYTWD$) (ITER1A)= BYTE($0,NBYTWD$) (CARD)$
                              BYTE($0,NBYTWD$) (ITER2A)= BYTE($19,NBYTWD$) (CARD)$
                              EXMPL)(ITER),ITER2)$
                              CLOSE EXMPL2$      ** MAIN CLOSE **
                              BEGIN
                                   ITER1 = 1$
                                   ITER2 = 1$
                              END **EXMPL2**

EXMPL2   (EXPROGM ) COMPLETED
```

Figure 2.1. Output from Former Syntax Analyzer
(BASIC, CARD IMAGES = ON.)

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

EXMPL2   (EXPROGM ) COMPLETED

EXMPL3   (EXPROGM ) COMPLETED

EXMPL1   (EXPROGM ) COMPLETED

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

Figure 2.2. Output from Former Syntax Analyzer (BASIC, CARD IMAGES = OFF.)

---

which is being processed. Figure 2.3(b) shows an example of the way JAVS assigns the module and JAVSTEXT name, if they are not supplied by the user.

Structural analysis was enhanced to take advantage of the proper global label information now provided by the new syntax analyzer. Figure 2.4 shows the module statement listing for module TSTLBL. During structural analysis, several messages are printed regarding possible structural errors, as shown in Fig. 2.5. The UNDEFINED GOTO messages refer to Statements 12 and 18, since LITTLE and ELABEL are not defined in the entire START-TERM sequence [see Fig. 2.3(a)]. At Statement 19, LITTLE is referenced again. An infinite loop is detected starting at statement 23 for the switch label CLABEL. JAVS can detect only structural infinite loops. Control transfers can be made which JAVS does not detect, such as the result of an invocation, which modify the control flow, thereby making the infinite loop warning superfluous.

Another program structure anomoly is reported in the documentation output generated by the JAVS command:

```
ASSIST, STATEMENT.
```

Figure 2.6 shows four statements in module CLOSEEX (see statement listing in Fig. 2.7) which cannot be executed. The JOVIAL JOCIT compiler does not detect unreachable code. Details of these and all other JAVS reports are given in the JAVS Reference Manual.[2]

```
'', JAVSTEXT LABEL COMPUTE (COMPOL)''                                TEST    2     1
      '' TEST CASE TO TEST LABEL PROBLEMS ''                         TEST    3     2
 START                                                               TEST    4     3
 PROC TSTLBL (IDUMMY)$                                               TEST    5     4
 ITEM IDUMMY I 24 S$                                                 TEST    6     5
 ITEM INDX I 24 S$                                                   TEST    7     6
AAAAAAAAAAA. BEGIN                                                   TEST    8     7
     INDX = -1$                                                      TEST    9     8
 ALABEL.                                                             TEST   10     9
     INDX = INDX+1$                                                  TEST   11    10
     GOTO LABELE$                                                    TEST   12    11
     GOTO LITTLE$                                                    TEST   13    12
     IF INDX EQ 3$                                                   TEST   14    13
     GOTO CLABEL$                                                    TEST   15    14
     GOTO BLABEL$                                                    TEST   16    15
     GOTO SWTCH ($INDX$) $                                           TEST   17    16
     IF INDX EQ 5$                                                   TEST   18    17
     GOTO ELABEL$                                                    TEST   19    18
     SWITCH SWTCH = (ALABEL,BLABEL,LITTLE)$                          TEST   20    19
 BLABEL.                                                             TEST   21    20
     INDX = INDX+1$                                                  TEST   22    21
     CLOSE  LABELE $                                                 TEST   23    22
      BEGIN                                                          TEST   24    23
         GOTO CLABEL$                                                TEST   25    24
     END                                                             TEST   26    25
     GOTO ALABEL$                                                    TEST   27    26
 CLABEL.                                                             TEST   28    27
     INDX = -1$                                                      TEST   29    28
     GOTO CHOICE ($INDX+1$)$                                         TEST   30    29
     SWITCH CHOICE = (DLABEL,BLABEL,CLABEL,DLABEL)$                  TEST   31    30
 DLABEL.                                                             TEST   32    31
     INDX = INDX+1$                                                  TEST   33    32
     GOTO ALABEL$                                                    TEST   34    33
     END                                                             TEST   35    34
     TERM$                                                           TEST   36    35

MODULES DEFINED IN TEXT
   1   TSTLBL  (LABEL   )
   2   LABELE  (LABEL   )
```

Figure 2.3(a). Output from New Syntax Analyzer
(BASIC, CARD IMAGES = ON)

```
START $                                                          TEST  37    1
A=3$.$                                                           TEST  38    2
PROC ZILCH $                                                     TEST  39    3
ARRAY ABCDEF I S $                                               TEST  40    4
BEGIN                                                            TEST  41    5
23.4.5E3A2 END                                                   TEST  42    6
BEGIN                                                            TEST  43    7
I=1$                                                             TEST  44    8
C=2$                                                             TEST  45    9
   ITEM AAAA I 5 1.4$                                            TEST  46   10
   ITEM AAAA I 5 1..4$                                           TEST  47   11
   ITEM AAAA I 5 1...4$                                          TEST  48   12
   ITEM AAAA I 5 1....4$                                         TEST  49   13
   ITEM AAAA I 5 1.....4$                                        TEST  50   14
   ITEM AAAA I 5 194 $                                           TEST  51   15
   ITEM AAAA I 5 .4$                                             TEST  52   16
   ITEM AAAA I 5 ..4$                                            TEST  53   17
   ITEM AAAA I 5 ...4$                                           TEST  54   18
   ITEM AAAA I 5 ....4$                                          TEST  55   19
   ITEM AAAA I 5 .....4$                                         TEST  56   20
       DIRECT                                                    TEST  57   21
       ASSIGN A6 OR A7 AND A8                                    TEST  58   22
A      ASSIGN B                                                  TEST  59   23
C      D  ASSIGN E                                               TEST  60   24
       JOVIAL                                                    TEST  61   25
BBB=AAA$                                                         TEST  62   26
BBB==AAA$                                                        TEST  63   27
END                                                              TEST  64   28
DEFINE BEG '' BEGIN '' $                                         TEST  65   29
BEG                                                              TEST  66   30
END                                                              TEST  67   31
INVOKE $ LABEL.                                                  TEST  68   32
FOR I = AABB(C).DD(E).FF(G)$                                     TEST  69   33
PROC MANYPARAMETERS(IN.OUT.CROSS=MATCH .AGAIN.)$                 TEST  70   34
BEGIN                                                            TEST  71   35
A=3$                                                             TEST  72   36
END                                                              TEST  73   37
TERM$                                                            TEST  74   38

MODULES DEFINED IN TEXT
   3   LABEL    (JAVS0003)
   4   ZILCH    (JAVS0003)
   5   MANYPARA(JAVS0003)
```

Figure 2.3(b). Output from the New Syntax Analyzer (BASIC, CARD IMAGES = ON.)

```
MODULE STATEMENT LISTING

MODULE <TSTLBL  >, JAVSTEXT <LABEL   >, PARENT MODULE <TSTLBL  >

 NO.  LVL          STATEMENT                                   DD-PATHS          CONTROL
---------------------------------------------------------------------------------------
   1 ( 0)          '', JAVSTEXT LABEL COMPUTE ( COMPOL ) ''
   2 ( 0)          '' TEST CASE TO TEST LABEL PROBLEMS ''
   3 ( 0)          START
   4 ( 0)          PROC TSTLBL ( IDUMMY ) $                    (  1)
   5 ( 0)          ITEM IDUMMY I 24 S $
   6 ( 0)          ITEM INDX I 24 S $
   7 ( 0)        AAAAAAAAAAA.                                                  <-----
   8 ( 1)            BEGIN
   9 ( 1)            INDX = - 1 $
  10 ( 1)        ALABEL.                                                       <-----
  11 ( 1)            INDX = INDX + 1 $
  12 ( 1)            GOTO LABELE $                                           INV
  13 ( 1)            GOTO LITTLE $                                                ----->
  14 ( 2)            IF INDX EQ 3 $                           (  2-  3)      IF
  15 ( 1)              GOTO CLABEL $                                              ----->
  16 ( 1)            GOTO BLABEL $                                                ----->
  17 ( 1)            GOTO SWTCH ($ INDX $) $                  (  4-  7)           ----->
  18 ( 2)            IF INDX EQ 5 $                           (  8-  9)      IF
  19 ( 1)              GOTO ELABEL $                                              ----->
  20 ( 1)            SWITCH SWTCH = ( ALABEL , BLABEL , LITTLE ) $
  21 ( 1)        BLABEL.                                                       <-----
  22 ( 1)            INDX = INDX + 1 $
                     GOTO ALABEL $                                              ----->
                 CLABEL.                                                       <-----
                     INDX = - 1 $
  25 ( 1)            GOTO CHOICE ($ INDX + 1 $) $             ( 10- 14)           ----->
  26 ( 1)            SWITCH CHOICE = ( DLABEL , BLABEL , CLABEL , DLABEL ) $
  25 ( 1)        DLABEL.                                                       <-----
                     INDX = INDX + 1 $
  26 ( 1)            GOTO ALABEL $                                               ----->
  27 ( 1)            END
  28 ( 0)          TERM $
---------------------------------------------------------------------------------------
```

Figure 2.4. Module Statement Listing for TSTLBL

```
**** ERROR ****                                                UNDEFINED GOTO.
-------------------------------------------------------------------------------
**** ERROR ****                                                UNDEFINED GOTO.
-------------------------------------------------------------------------------
    STATEMENT DESCRIPTOR BLOCKS UPDATED.
MODULE NAME = TSTLBL  , EXTERNAL LABEL = LITTLE   AT STATEMENT NUMBER =   19.

THE FOLLOWING STATEMENTS CONSTITUTE A POSSIBLE INFINITE LOOP . . .
  23   24   22   23
-------------------------------------------------------------------------------
**** ERROR ****                              POSSIBLE INFINITE LOOP DETECTED.
```

Figure 2.5. Structural Analysis Output for TSTLBL

```
STATEMENT/DDPATH LISTING

MODULE <CLOSEEX >, JAVSTEXT <EXTERNAL>, PARENT MODULE <EXTERNAL>
```

STMT	TYPE	DD-PATHS BEGUN BY STATEMENT	DD-PATHS CONTAINING STATEMENT
1	CLOS	(1)	1B
2	BEGN		1
3	ASMT		1
4	FOR1		1
5	NULL		1 3
6	BEGN		1 3
7	FOR2		1 3 5
8	BEGN		1 3 5
9	FOR3		1 3 4 5
10	BEGN		1 3 4 5
11	IF	(2- 3)	1E 2B 3B 3E 4E 5E
12	TEST		2
13	GOTO		3
14	ASMT	*** POTENTIALLY UNREACHABLE STATEMENT ***	
15	ASMT	*** POTENTIALLY UNREACHABLE STATEMENT ***	
16	END	(4- 5)	2E 4B 5B
17	ASMT		5
18	END		5
19	END	*** POTENTIALLY UNREACHABLE STATEMENT ***	
20	END	*** POTENTIALLY UNREACHABLE STATEMENT ***	

Figure 2.6. Executable Statements Contained on Each Decision-to-Decision Path

```
MODULE STATEMENT LISTING

MODULE <CLOSEEX >, JAVSTEXT <EXTERNAL>, PARENT MODULE <EXTERNAL>

 NO.  LVL            STATEMENT                                    DD-PATHS
-------------------------------------------------------------------------
   1 ( 0)              CLOSE CLOSEEX $                            (   1)
   2 ( 1)                BEGIN
   3 ( 1)                AP = 1 $
   4 ( 1)                FOR J = 2 $
   5 ( 2)          FORLAB.
   6 ( 2)                  BEGIN
   7 ( 2)                  FOR K = 1 , 10 $
   8 ( 3)                    BEGIN
   9 ( 3)                    FOR I = 1 , 1 , 3 $
  10 ( 4)                      BEGIN
  11 ( 4)                      IF REAL LQ 0.0 $             (   2-   3)
  12 ( 5)                        TEST I $
  13 ( 4)                      GOTO FORLAB $
  14 ( 4)                      REAL = I $
  15 ( 4)                      I = 4 $
  16 ( 4)                      END                          (   4-   5)
  17 ( 3)                    REAL =    *ABS ( J ) $
  18 ( 3)                    END
  19 ( 2)                  END
  20 ( 1)                END
-------------------------------------------------------------------------
```

Figure 2.7. Module Statement Listing for CLOSEEX

2.2 DESIGN OF THE SYNTAX ANALYZER

The Syntax Analysis component (known as JAVS-2) reads JOVIAL/J3 source text and creates the MDB, SDB, SB, SLT, and STB tables for each module in the input file. The input file contains one or more START-TERM texts; each of which may be either a COMPOOL text or program text. The type of text is declared in the JAVSTEXT directive used to identify the text; this directive must appear at the beginning of the START-TERM text of a COMPOOL and should be present for executable START-TERM texts as well. Each text is processed as a separate unit and no specific relationships between units (i.e., a program text referencing names declared in a COMPOOL text) are assumed or identified.

JAVS-2 produces three major classes of information:

1. The Module Descriptor Block (MDB)

2. The Statement Block (SB) and Statement Descriptor Block (SDB)

3. The Symbol Locator Table (SLT) and the Symbol Table Block (STB)

The JAVS-2 processor is organized into three distinct processes:

1. An initialization process (FINIT) which sets initial data into JAVS-2 data structures. This process is executed once for the input file.

2. A text recognition process (FONE) which reads the source text, captures DEFINE declarations, expands the text for references to DEFINE, identifies JOVIAL symbols, statements, and modules, creates entries in the permanent tables MDB, SDB, and SB, and constructs other temporary tables necessary for the text analysis process which follows. This process is executed once for each START-TERM text in the input file.

3. An analysis process (FTWO) which uses the tables prepared by the text-recognition process to construct the permanent tables SLT and STB and to complete related entries in the MDB, SDB and SB. This process is executed once for each START-TERM text in the input file.

The initialization process (1) defines values for entries in key items and arrays which describe the JOVIAL/J3 language elements (e.g., character set, primitives, ideograms) and JAVS descriptive elements (e.g., statement types and token types). Since many of these values are interrelated, initialization is better done by executing code rather than by using preset values. This also permits machine dependencies to be isolated within a set of DEFINEs contained in all JAVS-2 compilation units.

The text recognition process (2) is concerned primarily with analyzing the source text stream for elementary JOVIAL syntactical constructs (i.e., JOVIAL symbols, statements, and modules) and recording the syntactic description in the JAVS data base.

The analysis process (3) is concerned with transforming JOVIAL syntactical constructs which pertain to program flow, or control, into the JAVS structural description, a form which defines the basic structural properties of the source text. Since JAVS has no capability for data flow analysis, the resulting data base description contains only control symbols (i.e., modules, labels, and switches) and does not contain other symbols such as items, tables, and arrays.

The modules of JAVS-2 are hierarchically arranged. At the highest level, STEP1 sets BASIC default options, interprets each BASIC command for user-specified options, and invokes JAVS2 to process a source file. JAVS2 processes the options and invokes the lower-level driver FRONTEND. FRONTEND first calls FINIT, then alternately invokes FONE and FTWO for each START-TERM text until an end-of-file is encountered. At the lowest level, FINIT and FONE both invoke modules which clear core-resident work areas (e.g., AZAP, BZAP, etc.), and all three major processors invoke modules (MGET and MPUT) to fetch and store the MDB entry. In its overlay form, the primary link for the component (JAVS-2A) should consist of STEP1, JAVS2, FRONTEND and the low-level modules.

Secondary links are (1) JAVS2-B, other modules executed with FINIT (e.g., INIT, etc.), (2) JAVS2-C, other modules executed with FONE (e.g., SGET, etc.), and (3) JAVS2-D, other modules executed with FTWO (e.g., SYMDEF and SYMREF, etc.). The memory layout in Fig. 2.8 shows each JAVS-2 link along with the remainder of JAVS.

2.2.1 JAVS2-A (Primary Overlay)

In the hierarchy of modules, the top is STEP1 which sets processing options to default values upon first execution, interprets each BASIC command for user-defined options, and invokes JAVS2 for the BASIC execution command. JAVS2 interprets the designated processing options, sets flags for these options and invokes FRONTEND. FRONTEND drives all JAVS-2 processing for an input source text file. After an initializing call to FINIT, FONE and FTWO are called for each START-TERM sequence until an end-of-file is encountered on the input file. The remainder of JAVS2-A consists of "zap" routines which clear the various workspaces and modules to handle table structures (MGET and MPUT).

2.2.2 JAVS-2B Initialization Process Modules (Secondary Overlay)

FINIT drives initialization processing for one input file. It presets initial module values and invokes INIT to initialize values used for internal processing of JOVIAL/J3 language texts. The remainder of JAVS2-B consists of the initialization modules called by INIT.

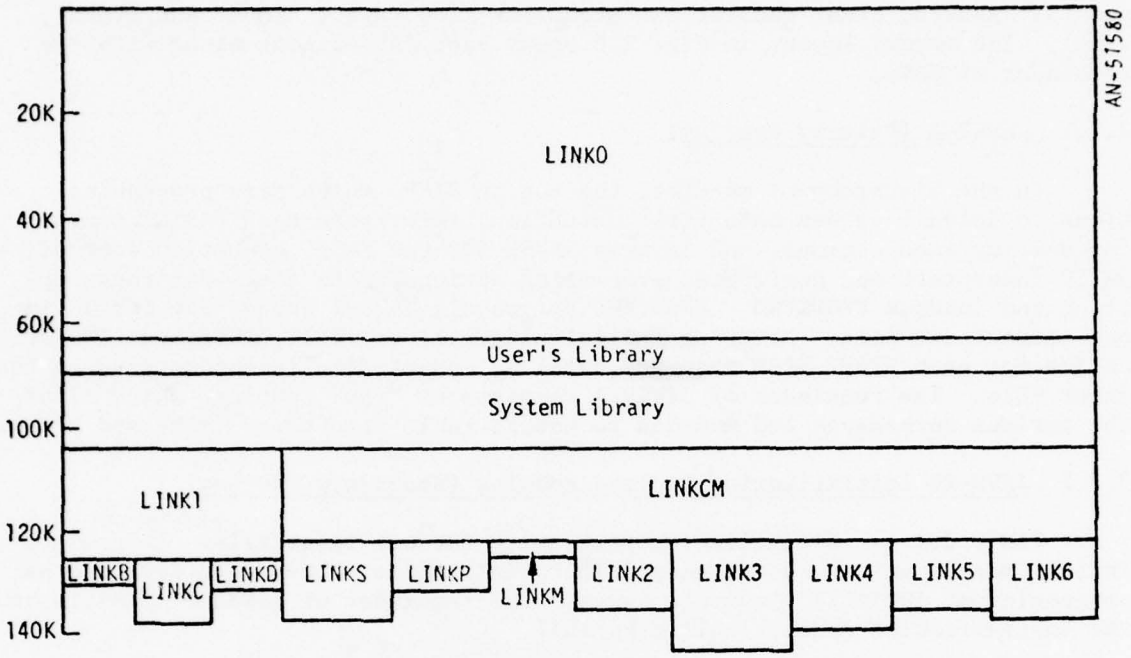
2.2.3 JAVS-2C Text Recognition Process Modules (Secondary Overlay)

FONE drives text recognition processing for one START-TERM text. It begins by invoking STZAP and JGET for the first JOVIAL symbol. If there is not an end-of-file encountered at the first symbol, then it invokes MODZAP to create the first module and LPUSH to put the module on the module nesting stack. Thereafter it invokes SGET for each statement in the START-TERM sequence until the module stack is empty (a condition signifying the end-of-file or a TERM statement).

2.2.4 JAVS-2D Analysis Process Modules (Secondary Overlay)

FTWO drives analysis processing for one START-TERM text. The objective of the analysis process is to construct the SLT and STB entries for control symbols (i.e., module names, statement labels, and switch names) defined or referenced in the text and to place SLT pointers for these symbols in the SB entries. At the conclusion of FONE, the SB contains the hash index for all names and constants; the hash index is used to access the appropriate entry in the temporary spelling table. FTWO replaces the hash index in the SB by the appropriate SLT index for control symbols, by the token descriptor NAME for any other name, or by the token descriptor CNST for a constant.

Specialized algorithms are used to scan the SB for control symbols only. These algorithms permit scanning to terminate as soon as possible in each statement scanned, and only statements of types which are permitted to contain control symbols are scanned. The analysis algorithms permit processing to be further divided into two sub-processes: one for symbol declaration and one



K = 1000 octal

Load Module	CONTENTS
LINK0	Some utility COMMONs, JAVS-0, part of JAVS-1, -10, -11
LINK1	JAVS-2A, JAVS-2 COMMONs
LINKB	JAVS-2B, MAKTAB from JAVS-1
LINKC	JAVS-2C
LINKD	JAVS-2D
LINKCM	Some COMMONs, part of JAVS-1, -10, -11
LINKS	STSTOP from JAVS-1
LINKP	STPROP from JAVS-10
LINKM	JAVS-M
LINK2	JAVS-3
LINK3	JAVS-5
LINK4	JAVS-7
LINK5	JAVS-9
LINK6	JAVS-6

Figure 2.8. JAVS Memory Layout

for symbol reference. All modules for a single START-TERM are first processed for control symbol declaration; the same modules are then processed for control symbol references. Undeclared control symbols are treated as external (to the START-TERM text) undefined symbols which are assumed to be declared in a COMPOOL. This eliminates the need for processing a referenced COMPOOL.

FTWO contains two successive loops, each of which selects the modules in the order encountered in the original source text. The first loop invokes SYMDEF to capture the declared control symbols for a particular module. The second loop invokes SYMREF to resolve references to control symbols and capture any undeclared control symbols referenced by the modules. SYMDEF and SYMREF process all the entries in the SB for the designated module. FTWO controls substitution of the hash index in the SB by the SLT index through a processing flag FLTOKEN which is set FALSE for the first loop and TRUE for the second.

Scope of declaration and reference is handled by module nesting information captured in the MDB during FONE processing. This information is added to the spelling table together with a symbol category when a symbol is declared. If more than one control symbol is declared for a particular spelling entry, a new spelling entry is made and linked to the original entry. When a reference to a control symbol is encountered, the information in the spelling table (which is now a symbol table) is used to resolve the reference according to symbol category, scope of reference, and scope of declaration.

2.2.5 JAVS2 Data Structures

The data structures in JAVS-2 are organized into a number of categories:

- DEFINE declarations which establish dimensions, types, and machine dependencies
- Text buffers and descriptive data for text elements
- Processing flags to control analysis
- Error indicators and loop counts
- Temporary variables
- Tables managed by the Data Management Component
- JAVSTEXT, Module, and statement information
- Symbol dictionary, including a hash table and spelling table

All declarations appear in the JAVS-2 COMPOOL except for local temporary variables. There are no preset values in the COMPOOL; instead, each constant used in processing is declared in a DEFINE or is set during execution of FINIT.

2.2.6 JAVS-2 Text Buffers and Descriptive Data

Source text being processed by JAVS-2 moves through a series of buffers or workspaces in the course of transforming card lines to JAVS statement blocks. An overview of this process is shown in Fig. 2.9.

GTCARD reads the source text a card line at a time from the input file, places it in the input line workspace IS, and sets the number of characters in IL. The input line pointer IP is initialized to the first character in IS and is incremented as each character is drawn from IS by TGET. Whenever the line is exhausted, TGET invokes GTCARD for the next line. If the end-of-file is encountered by GTCARD, IL is set to a large default value (e.g., 81).

TGET always moves the current input character into the external input text workspace TS. If DIRECT code is being processed, each new line is moved directly to the JAVS Statement Block. The type of input text character is set to one of the following: TTEMPTY, TTCHAR, TTENDFILE, or TTENDLINE.

CGET interprets the input character in TS and converts it to an internal JOVIAL sign CS. The type of internal JOVIAL sign is set to one of the following: CTEMPTY, CTBLANK, CTLETTER, CTNUMERAL, CTSPECIAL, or CTOTHER. End-of-line and end-of-file characters are included in the set of permitted values for CS.

CBMOVE adds a character from CS into the basic JOVIAL element workspace BS as directed by BGET. BGET also assigns the type of basic element BT to one of the following: BTEMPTY, BTALPHABETIC, BTALPHANUMERIC, BTASIS, BTIDEOGRAM, BTNUMBER, BTSPACE, BTTEXTBREAK, and BTUNKNOWN. If JOVIAL text is being processed, BGET ignores end-of-line characters; if DIRECT text is being processed, BGET recognizes end-of-line as a basic element.

BJMOVE adds a basic JOVIAL element from BS to the JOVIAL symbol JS. A JOVIAL symbol consists of one or more basic elements. When processing a comment or a string of characters in a textual literal, CJMOVE adds each character to JS from CS, bypassing the use of BS. JGETSYM and JGETCOM control the construction of JS. The type of symbol JT is also set.

Text from JS is added to the JAVS Statement Block workspace by JSMOVE and from there it ultimately goes into the data base. While processing a DEFINE declaration, text from JS is added to the DEFINE descriptor workspace DS by JAMOVE. Symbols of type JTNAME and JTCONSTANT are also entered into the spelling workspace SS by JSAVE.

When expanding a DEFINE reference AGET replaces the referenced DEFINE name in JS by the sequence of symbols previously saved in DS. Nested DEFINE references are permitted, although recursion is not allowed.

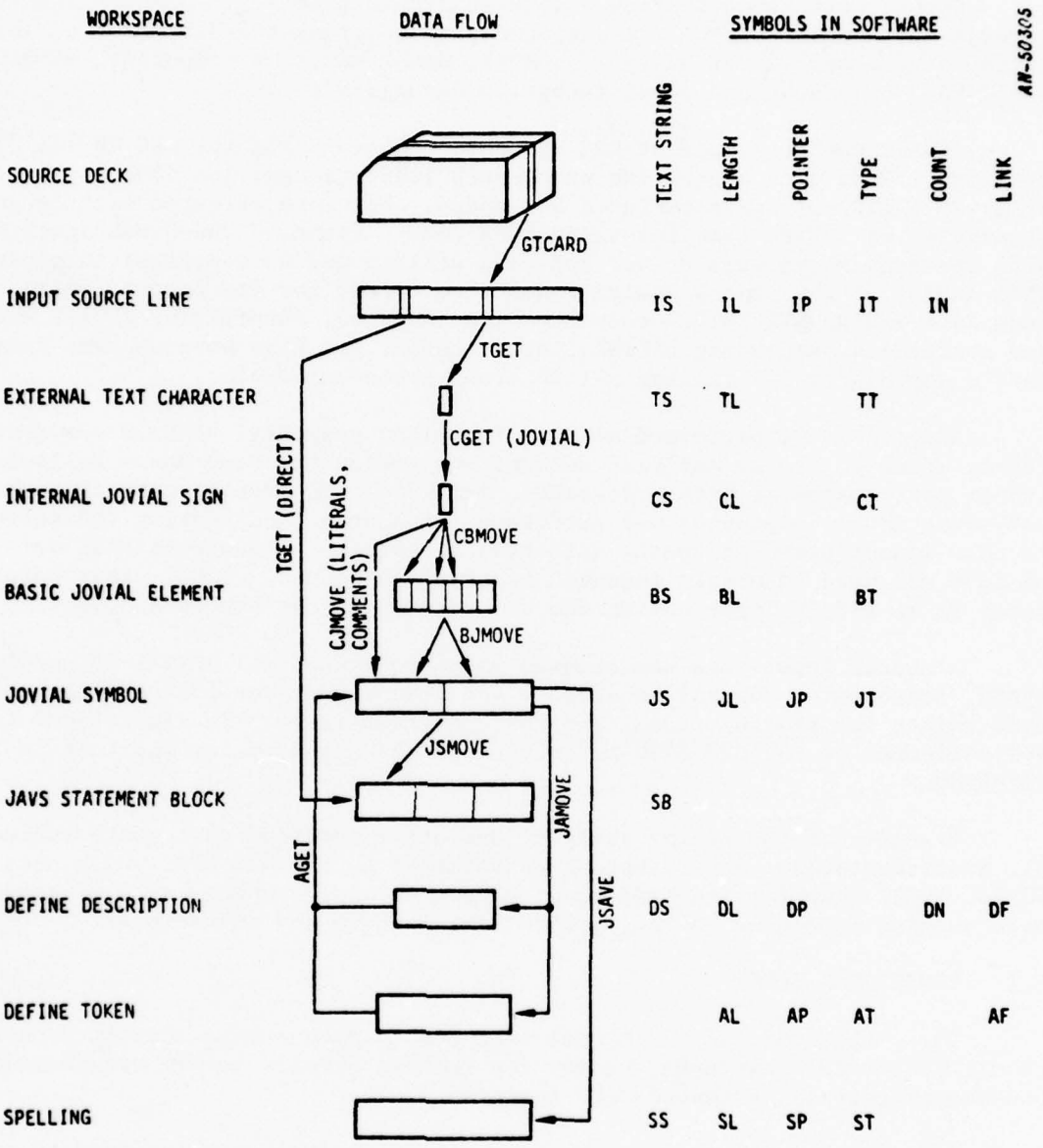


Figure 2.9. Flow of Text Data

3 ACCEPTANCE TESTING

Formal acceptance testing had three distinct phases: a functional test, several operational software tests, and path coverage tests. Prior to installation of the new syntax analyzer and the other modified components at RADC, the JAVS-2 component underwent extensive debugging tests.

JAVS-2 was developed at GRC in Santa Barbara using the CDC JOVIAL J3 compiler. When each of the two processors [text recognition (JAVS2-C) and analysis (JAVS2-D)] were designed and coded, they were executed with large quantities of JOVIAL compiler-validation code. First, JAVS2-C was interfaced with the syntax analysis driver and JAVS utility and data manager components. This subset of the syntax analyzer was then tested for its proper functions, such as breaking START-TERM sequences into modules, identifying JOVIAL symbols and statements, expanding DEFINEs, etc. Output for this process came from JAVS's capability of printing all internal permanent tables.

After JAVS-2C performed all its functions properly, JAVS-2D was designed, coded, added to syntax analyzer subset, and tested the same way. Following proper performance of both processors, interface and testing with the other JAVS functional components was performed. Isolating and testing the software in this manner provided speedy detection of errors. The operational version of JAVS was used to obtain indented module listings and symbol cross references as an aid in implementing the new version of the syntax analyzer.

A source input file was created which contained all JOVIAL J3 module types, statements, control constructs and JAVS directives for usage as the test object for the functional and path coverage tests. The functional test was performed on the CDC 6400 to be used as the standard for the test on the HIS 6180.

Transfer of the syntax analyzer and other modified components (structural, instrumentation, and retesting assistance) to the HIS 6180 which uses the JOVIAL JOCIT compiler required about 30 source line changes, some of which were machine dependencies (e.g., word size in bits and characters).

3.1 FUNCTIONAL TEST

The purpose of the functional test was to demonstrate correct processing of all JOVIAL J3 constructs, verify the reduced primary memory requirement, and demonstrate all enhanced JAVS features.

For a variety of reasons, the new syntax analyzer was designed to perform a few operations differently than the former syntax analyzer. Some of these operations are: (1) store only control symbols in the Symbol Locator Table (SLT), (2) parse labels on BEGIN and END statements as separate statements of type NULL, (3) do not distinguish (as far as statement type is concerned) between an item and a table item, (4) parse comments as a single token, (5) collect all comments which follow a JOVIAL statement (i.e., follow a $) other than a BEGIN or END and store them as part of that statement, (6) store a comment which follows a BEGIN or END as a separate statement of type COMT. In addition to these differences, the new syntax analyzer removed all of the

former restrictions listed in Sec. 1.2. Thus each of these specific design changes were considered as areas for stress testing.

In order to demonstrate functional correctness with a minimal amount of computer resources and printout, these procedures were followed:

1. Develop test object

   - Include at least one of each type of JOVIAL construct with the least amount of duplication of types.

   - Include constructs that would invoke JAVS error processing but not cause a fatal error.

   - Include JAVS computation directives to ensure proper parsing of these constructs

2. Generate JAVS Command Set

   - Print syntax and structural tables following syntax and structural analyses

   - Perform full instrumentation to demonstrate proper expansion of JAVS computation directives as well as structural instrumentation.

   - Include commands which would invoke each JAVS overlay load module.

   - Use a mixture of macro and standard commands.

3. Execute the testing using the job control card setup provided in the JAVS User's Guide[1]

4. Analyze the output

   - Verify all JOVIAL J3 module, statement, and control symbol types

   - Verify correct generation of DD-paths

   - Verify all generated JAVS error messages

   - Verify proper instrumentation

   - Verify correct execution of all enhancements

   - Verify reduced primary core requirements

The test object consisted of approximately 600 JOVIAL source lines. The JAVS report (which concludes all JAVS execution runs) in Fig. 3.1 summarizes some of the characteristics of the test object. The JAVS command sequence, after expansion into standard commands, is shown in Fig. 3.2.

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	DDPS	DS BLKS	PARMS IN	PARMS OUT	DIRECT CODE	COMP TOT	COMP DDP
1	TSTLBL	LABEL	TSTLBL	PROC	EXT	70	28	17	7	210	139	10	11	1	14	86	1	ONO		0	0
2	LABELE	LABEL	TSTLBL	CLSM	GLBL	11	4	1	3	13	8	1	2	0	1	4	0	ONO		0	0
3	LABEL	JAVS0003	LABEL	PROG		17	8	3	2	46	38	5	8	4	3	11	0	ONO		0	0
4	ZILCH	JAVS0003	LABEL	PROC	INT	9	26	4	6	220	113	1	1	0	1	9	0	OYES		0	0
5	MANYPARA	JAVS0003	LABEL	PROC	INT	9	4	1	3	29	21	2	2	0	1	4	3	2NO		0	0
6	JAVS0003	JAVS0006	JAVS0003	PROG		0	3	0	0	310	20	0	0	0	0	0	0	ONO		0	0
7	AAA	AAA	AAA	PROG		0	5	0	0	299	24	0	0	0	0	0	0	ONO		0	0
8	EXCOMPL	EXCOMPL	EXCOMPL	CMPL		0	11	0	0	100	48	0	0	0	0	0	0	ONO		0	0
9	EXPROGM	EXPROGM	EXPROGM	PROG		72	53	23	17	438	332	1	8	2	13	62	0	ONO		0	0
10	EXMPL2	EXPROGM	EXPROGM	CLSM	GLBL	9	5	2	3	23	14	1	1	0	1	5	0	ONO		0	0
11	EXMPL1	EXPROGM	EXPROGM	PROC	INT	126	51	21	10	367	256	5	7	2	19	87	2	ONO		0	0
12	EXMPL3	EXPROGM	EXMPL1	CLSP	LOCL	9	4	1	3	20	12	1	1	0	1	4	0	ONO		0	0
13	TESTALL	TESTALL	TESTALL	PROG		11	55	3	41	360	254	1	31	2	1	5	0	ONO		0	0
14	LISTO	TESTALL	TESTALL	PROC	INT	12	6	2	4	17	11	1	1	0	1	5	0	ONO		0	0
15	LITTLE	TESTALL	TESTALL	CLSM	GLBL	0	4	0	0	14	10	1	1	0	0	0	0	ONO		0	0
16	DUMPCH	TESTALL	TESTALL	PROC	INT	23	14	4	8	108	73	2	4	1	3	13	0	1NO		0	0
17	TSTFOR	TESTALL	TESTALL	PROC	INT	239	122	72	25	823	700	13	18	3	47	222	1	ONO		0	0
18	TSTIFG	TESTALL	TESTALL	PROC	INT	111	52	36	10	290	236	12	13	1	23	125	1	1NO		0	0
19	RANGE	TESTALL	TSTIFG	CLSP	LOCL	21	6	3	3	25	19	1	2	0	3	9	0	ONO		0	0
20	CLOSEA	CLOSEA	CLOSEA	CLSM	EXT	12	16	4	11	98	47	6	11	2	1	14	0	ONO		0	0
21	CC	CLOSEA	CLOSEA	PROC	INT	9	5	1	4	16	12	2	2	0	1	5	0	ONO		0	0
22	DD	CLOSEA	CLOSEA	CLSM	GLBL	9	4	1	3	11	9	1	1	0	1	4	0	ONO		0	0
23	EE	CLOSEA	CLOSEA	PROC	INT	9	4	1	3	11	8	1	2	1	1	4	0	ONO		0	0
24	FF	CLOSEA	EE	CLSP	LOCL	9	4	1	3	11	9	1	1	0	1	4	0	ONO		0	0
25	AA	PROCAA	AA	PROC	EXT	11	7	1	5	29	19	1	2	1	1	5	0	ONO		0	0
26	FF	PROCAA	AA	CLSM	GLBL	9	4	1	3	11	9	1	1	0	1	4	0	ONO		0	0
27	EXTERNAL	EXTERNAL	EXTERNAL	PROG		85	21	11	5	144	103	9	21	2	17	67	0	ONO		0	0
28	PROCEXT	EXTERNAL	EXTERNAL	PROC	INT	78	24	16	2	119	84	8	11	2	11	39	2	2NO		0	0
29	CLOSEEX	EXTERNAL	EXTERNAL	CLSM	GLBL	28	20	11	3	78	65	2	2	0	5	35	0	ONO		0	0
30	CLOZ	CLOZ	CLOZ	CLSM	EXT	11	14	4	4	81	51	8	14	8	1	13	0	ONO		0	0
31	CC	CLOZ	CLOZ	PROC	INT	23	12	5	2	56	39	6	6	2	3	13	1	2NO		0	0
32	FRINK	FRINK	FRINK	CLSM	EXT	10	5	1	4	22	14	2	2	1	1	4	0	ONO		0	0
33	MAINPG	MAINPG	MAINPG	PROG		10	6	3	3	35	24	1	1	1	1	4	0	ONO		0	0
34	EXPROC	EXPROC	EXPROC	PROC	EXT	10	8	2	5	30	19	2	2	0	1	6	0	ONO		0	0
35	IFEITH2	IFEITH2	IFEITH2	PROG		22	13	4	4	59	45	0	0	0	3	14	0	ONO		0	0
				TOTAL		1094	628	260							182						

LIBRARY INFORMATION--RUN OF 0511

NO.	LIBRARY NAME	TYPE ACCESS	DATE CREATED	TIMES ALTERED	LAST ALTERED	TOTAL WORDS	LLINKS USED	LIBRARY MODULES	LIBRARY FRAGMENTS
1	**NOT OPENED**								
2	TEST	NEW	0511	1	0511	140020	467	35	280

Figure 3.1. Module Summary Information

```
CREATE LIBRARY=TEST.
START.
BASIC,COMMENTS= ON.
BASIC.
FOR LIBRARY.
STRUCTURAL.
END FOR.
FOR LIBRARY.
PRINT,MODULE.
PRINT,SDB.
PRINT,SLT.
PRINT,STB.
PRINT,DDP.
PRINT,DDPATHS.
END FOR.
FOR LIBRARY.
INSTRUMENT,MODE=FULL.
INSTRUMENT.
END FOR.
PRINT,JAVSTEXT=TESTALL,INSTRUMENTED=ALL.
PRINT,JAVSTEXT=EXPROGM,INSTRUMENTED=ALL.
PRINT,JAVSTEXT=EXTERNAL,INSTRUMENTED=ALL.
ASSIST,CROSSREF,JAVSTEXT=EXTERNAL.
ASSIST,CROSSREF,LIBRARY.
JAVSTEXT=EXTERNAL.
FOR JAVSTEXT.
ASSIST,STATEMENTS.
END FOR.
DEPENDENCE,GROUP,LIBRARY.
DEPENDENCE,GROUP,AUXLIB.
DEPENDENCE,SUMMARY.
END.
```

Figure 3.2. JAVS Commands for Functional Test

The report in Fig. 3.1 was used to verify recognition of all module types. Statement type recognition was verified by analyzing the "STMT CLASS" and "TYPE CODE" columns of the table printed in Fig. 3.3 for several modules in the test object. Verification of control symbol recognition was performed by analyzing SLT and STB tables for several modules in the test object, such as those in Figs. 3.4(a) and (b).

DD-path generation was demonstrated by printing the DD-path table and definition reports, shown in Figs. 3.5(a) and (b), for several modules.

Reduced core requirement was evident on the computer dayfile for the functional test. Proper instrumentation and implementation of enhancements was demonstrated by analyzing JAVS reports produced by the functional test run. Comparison of execution time and secondary storage requirements between JAVS with the new syntax analyzer and the former one was not possible in this particular functional test, since the test object quickly produced a fatal error when the attempt was made to run with the former version of JAVS.

Two errors stemming from the syntax analyzer were uncovered during the functional test. One error was the improper number of elements being initialized in an array. This would have been detected during debugging, except that debugging took place using the non-overlay version of JAVS on the CDC 6400. The extra elements being initialized were harmlessly resident in unused core. The other error was caused by correct module classification for functions as type "FUNC." The former syntax analyzer classified functions as "PROC." Thus, during structural analysis, modules of type FUNC were not recognized. This was not detected before transfer to RADC because a function example had not been included in the test object. These errors required only five source line changes in the syntax and structural analyzers. The functional test was repeated on the corrected JAVS execute (*H) file.

3.2 OPERATIONAL TESTS

The operational tests took the form of running a small computational program (used by RADC for installation checkout), JAVS self-documentation runs, and analysis of "foreign" JOVIAL code supplied by RADC.

The operational program's analysis helped provide data to compare computer resource differences between the former and new versions of JAVS and verify that the functional performance was the same.

JAVS documentation runs were made for each modified JAVS software component. These components were: the syntax analyzer (JAVS-2), structural analyzer (JAVS-3,-4), instrumentor (JAVS-5), and retesting assistant (JAVS-7). Each documentation run produced the listings specified for computer documentation in the Statement of Work. This and the self-documentation runs generated under the previous contract, together with the System Design and Implementation Manual,[5] make up the complete computer program documentation for JAVS. These runs provided additional data for comparing computer resources between the former and new versions.

STATEMENT DESCRIPTOR LISTING

MODULE <TSTLBL >, JAVSTEXT <LABEL >, PARENT MODULE <TSTLBL >

NO.	STMT CLASS	TYPE CODE	SEQ NUM	WD-PRS	SB INDEX	NO. LABELS	LABEL WD-PRS	INTER-MTH PTR	INTRA-MTH PTR
1	DIPT	0	1	13	1	0	0	0	0
2	COMT	0	2	10	27	0	0	0	0
3	STRT	1	3	2	47	0	0	0	0
4	PROC	100	4	9	51	0	0	0	0
5	ITEM	1	5	8	69	0	0	0	0
6	ITEM	1	6	8	85	0	0	0	0
7	NULL	100	7	4	101	1	4	0	0
8	BEGN	100	8	2	109	0	0	0	0
9	ASMT	100	9	6	113	0	0	0	3
10	ASMT	100	10	11	125	1	3	0	6
11	GTCL	100	11	5	147	0	0	0	0
12	GTEX	100	12	5	157	0	0	28	0
13	IF	200	13	6	167	0	0	15	0
14	GOTO	100	14	5	179	0	0	22	0
15	GOTO	100	15	5	189	0	0	20	0
16	GTSW	100	16	9	199	0	0	19	0
17	IF	200	17	6	217	0	0	20	0
18	GTEX	100	18	5	229	0	0	28	0
19	INSW	100	19	16	239	0	0	0	5
20	ASMT	100	20	11	271	1	3	0	6
21	GOTO	100	25	5	293	0	0	10	0
22	ASMT	100	26	9	303	1	3	0	6
23	GTSW	100	27	11	321	0	0	24	0
24	INSW	100	28	19	343	0	0	0	5
25	ASMT	100	29	11	381	1	3	0	6
26	GOTO	100	30	5	403	0	0	10	0
27	END	100	31	1	413	0	0	0	0
28	TERM	1	32	3	415	0	0	0	0

Figure 3.3. Statement Description

SYMBOL TABLE LISTING

MODULE <TSTLBL >, JAVSTEXT <LABEL >, PARENT MODULE <TSTLBL >

NO.	SYMBOL	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	22
1	TSTLBL	3	GLBL	PROC			0	0					0	0	0	0	0	0	0	0	0	4
2	AAAAAAAA	1	LOCL	LABL			0	0					0	0	0	0	0	0	0	0	0	7
3	ALABEL	1	LOCL	LABL			0	0					0	0	0	0	0	0	0	0	0	10
4	SWTCH	3	LOCL	INSW			3	0					0	0	0	0	0	0	0	0	0	19
5	BLABEL	1	LOCL	LABL			0	0					0	0	0	0	0	0	0	0	0	20
6	CLABEL	1	LOCL	LABL			0	0					0	0	0	0	0	0	0	0	0	22
7	CHOICE	3	LOCL	INSW			4	0					0	0	0	0	0	0	0	0	0	24
8	DLABEL	1	LOCL	LABL			0	0					0	0	0	0	0	0	0	0	0	25
9	LITTLE	3	CMPL	CAT1			0	0					0	0	0	0	0	0	0	0	0	12
10	ELABEL	3	CMPL	CAT1			0	0					0	0	0	0	0	0	0	0	0	18

(a)

SYMBOL LOCATION TABLE LISTING

MODULE <TSTLBL >, JAVSTEXT <LABEL >, PARENT MODULE <TSTLBL >

NO.	SYMBOL	MODULE	JAVSTEXT	STB	FIRST	LAST	NUMBER		
1	TSTLBL	TSTLBL	LABEL	1	4	4	1	5	5
2	AAAAAAAA	TSTLBL	LABEL	2	7	7	1	8	8
3	ALABEL	TSTLBL	LABEL	3	10	26	4	10	10
4	SWTCH	TSTLBL	LABEL	4	16	19	2	16	16
5	BLABEL	TSTLBL	LABEL	5	15	24	4	15	15
6	CLABEL	TSTLBL	LABEL	6	14	24	3	14	14
7	CHOICE	TSTLBL	LABEL	7	23	24	2	19	19
8	DLABEL	TSTLBL	LABEL	8	24	25	3	20	20
9	LABELE	LABELE	LABEL	1	11	11	1	11	11
10	LITTLE	TSTLBL	LABEL	9	12	19	2	12	12
11	ELABEL	TSTLBL	LABEL	10	18	18	1	18	18

(b)

Figure 3.4. Control Symbol Description

DD-path generation was demonstrated by printing the DD-path table and definition reports, shown in Figs. 3.5(a) and (b), for several modules.

DD-PATH TABLE LISTING

MODULE <TSTLBL >, JAVSTEXT <LABEL >, PARENT MODULE <TSTLBL >

NO.	1ST ST	END ST	EDG COM	PRD IND	TOP LVL	PAR DD	X TST	NO. ST.	DS INDEX	STATEMENTS ON DD-PATH							
1	4	28	7 0	1	0	0	0	8	1	4	7	8	9	10	11	12	28
2	13	23	3 0	1	0	0	0	4	9	13	14	22	23				
3	13	28	7 0	2	0	0	0	8	13	13	15	20	21	10	11	12	28
4	16	28	5 0	1	0	0	0	6	21	16	19	10	11	12	28		
5	16	28	7 0	2	0	0	0	8	27	16	19	20	21	10	11	12	28
6	16	28	2 0	3	0	0	0	3	35	16	19	28					
7	16	17	2 0	4	0	0	0	3	38	16	19	17					
8	17	28	2 0	1	0	0	0	3	41	17	18	28					
9	17	28	6 0	2	0	0	0	7	44	17	20	21	10	11	12	28	
10	23	28	7 0	1	0	0	0	8	51	23	24	25	26	10	11	12	28
11	23	28	7 0	2	0	0	0	8	59	23	24	20	21	10	11	12	28
12	23	23	3 0	3	0	0	0	4	67	23	24	22	23				
13	23	28	7 0	4	0	0	0	8	71	23	24	25	26	10	11	12	28
14	23	28	7 0	5	0	0	0	8	79	23	24	25	26	10	11	12	28

(a)

```
MODULE DD-PATH DEFINITION LISTING

MODULE <TSTLBL >, JAVSTEXT <LABEL >, PARENT MODULE <TSTLBL >

 NO.  LVL STATEMENT                                          DD-PATHS GENERATED
-------------------------------------------------------------------------------
  3 ( 0)  START
  4 ( 0)  PROC TSTLBL ( IDUMMY ) $
                                                    ** DD-PATH   1 IS PROCEDURE ENTRY
 13 ( 1)    IF INDX EQ 3 $
                                                    ** DD-PATH   2 IS TRUE BRANCH
                                                    ** DD-PATH   3 IS FALSE BRANCH
 16 ( 1)    GOTO SWTCH ($ INDX $) $
                                                    ** DD-PATH   4 IS SWITCH OUTWAY   1
                                                    ** DD-PATH   5 IS SWITCH OUTWAY   2
                                                    ** DD-PATH   6 IS SWITCH OUTWAY   3
                                                    ** DD-PATH   7 IS SWITCH OUTWAY   4
 17 ( 1)    IF INDX EQ 5 $
                                                    ** DD-PATH   8 IS TRUE BRANCH
                                                    ** DD-PATH   9 IS FALSE BRANCH
 19 ( 1)    SWITCH SWTCH = ( ALABEL , BLABEL , LITTLE ) $
 23 ( 1)    GOTO CHOICE ($ INDX + 1 $) $
                                                    ** DD-PATH  10 IS SWITCH OUTWAY   1
                                                    ** DD-PATH  11 IS SWITCH OUTWAY   2
                                                    ** DD-PATH  12 IS SWITCH OUTWAY   3
                                                    ** DD-PATH  13 IS SWITCH OUTWAY   4
                                                    ** DD-PATH  14 IS SWITCH OUTWAY   5
 26 ( 1)    SWITCH CHOICE = ( DLABEL , BLABEL , CLABEL , DLABEL ) $
 28 ( 0)  TERM $
```

(b)

Figure 3.5. DD-path Information

Additional operational tests were made on JOVIAL source supplied by RADC. In these tests, the test objects were subsets of programs. The test objects were processed through instrumentation and the instrumented files passed to the JOCIT compiler. At this stage, instrumentation errors became evident. The errors were due to the presence of long tokens (comments, in this case) which continued onto the next line. The problem arose only in IF statements, where the instrumented control keyword changes to an IFEITH. Along this same line of proper extraction of source, an error in building a source line for indented printing was detected. Both errors required a very select set of circumstances to manifest themselves. The new syntax analyzer's characteristic of parsing a comment as a single token was the catalyst.

These errors were corrected, the operational tests rerun with no compiler errors, and the modified JAVS component was redocumented by JAVS.

3.3 PATH COVERAGE TESTS

The path coverage goal was to determine what parts of the syntax analyzer had not yet been exercised by a data set which should represent a full range of JOVIAL syntax. Any control paths not executed were to be analyzed.

The self-documentation output for JAVS-2 was reviewed to help choose the test objects. The syntax analyzer (JAVS-2) consists of 121 modules in 20 START-TERM sequences, including the COMPOOL. Many of these START-TERMs are low-level routines which contain only one to three DD-paths and are invoked many times. The computer and human resources required in analyzing these routines, the inconvenience of adding the JOCIT-required control cards to each instrumented START-TERM sequence, and the knowledge that the higher level JAVS-2C and JAVS-2D subcomponents invoke most of the low-level routines were the justifications for choosing JAVS2-C and JAVS-2D as test objects.

Figure 3.6 shows the overall flow of activities performed to obtain path coverage results for the two subcomponents. The first step in obtaining path coverage results was to build a database library and instrument JAVS2-C. Both test objects were put on the database library for efficiency purposes. Each software subcomponent was then tested separately since JAVS2-C and JAVS2-D are functionally distinct and do not reside in core at the same time.

The 600-line source program used in the functional test was to be the JOVIAL program used as one of the two inputs to the test execution phase. The other input was the JAVS command set:

```
CREATE LIBRARY = TEST.

START.

BASIC.

END.
```

The full source program required too much computer time and AUDIT file space for realistic analysis of JAVS-2C. JAVS-2C performs syntax analysis at the

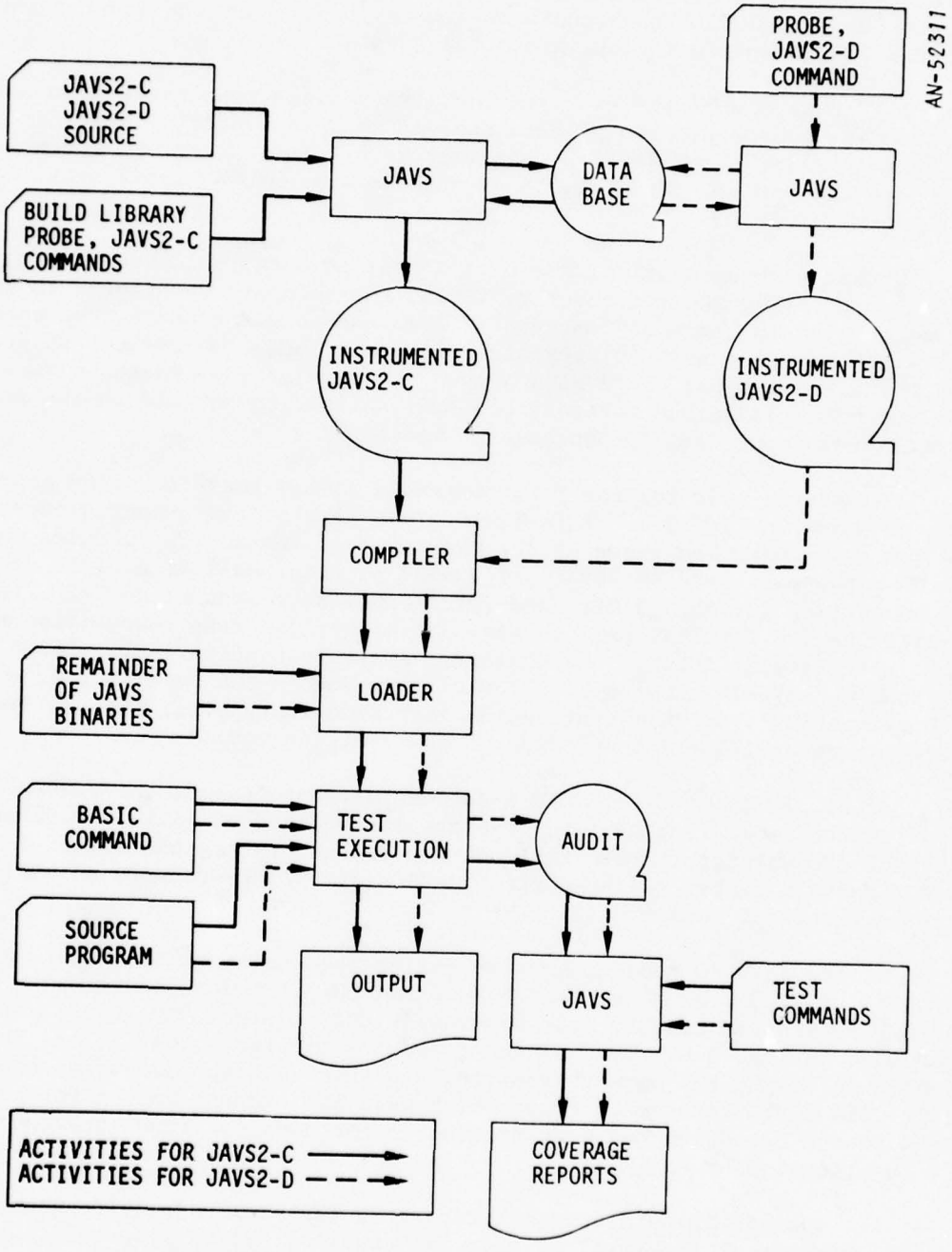


Figure 3.6. Flow Diagram of Path Coverage Tests

character and symbol level and thus executes slowly in its instrumented form. To reduce resource requirements, the source program was cut to 200 lines, leaving out such JOVIAL constructs as item switch, function, input parameter, nested DEFINE, exponentiation, and octal constant.

JAVS analysis of the two execution trace files, generated from using the 200-line source program as data, showed that an overall 69% of the 1,203 DD-paths in JAVS-2C and JAVS2-D were exercised. The overall statement execution coverage was 78% out of 3,055 executable statements (3,622 total source statements).

During instrumentation, the user specifies a testcase boundary via a JAVS command. For the two coverage tests, the boundary was placed in the driver modules for each subcomponent. This caused the execution of each START-TERM sequence to be a separate testcase. Thus, the effects of specific JOVIAL statement types could be analyzed in terms of what DD-paths were or were not hit. Judicious testcase boundary definition can aid in the process of modifying input data for subsequent testing.

The source code for the first testcase (i.e., the START-TERM source input) is shown in Fig. 3.7. This listing was part of the output generated during the test execution phase of the two coverage tests. The modules invoked and DD-paths exercised by JAVS' processing of this small START-TERM sequence are summarized in Figs. 3.8(a) and (b). Since subcomponent JAVS-2C, identified as the J2C JAVSTEXT name in Fig. 3.8(a), is the text recognition processor, it repeatedly invokes the character analysis modules (CGET, CSIGN, CTYPE, and TGET). JAVS-2C handles the flow of text data shown in Fig. 2.9, starting with extraction from the input source line into characters, through symbol and statement recognition and building of text-related tables.

After all input START-TERM sequences were processed, the cumulative summary results showed 73% DD-path coverage of the invoked JAVS-2C modules. Only JGETFACT (containing 15 DD-paths) was not invoked during the test. JGETFACT adds the scale factor to the current JOVIAL symbol; there were no scale factors in the input source data.

Of the JAVS-2D modules invoked during processing of the first START-TERM sequence, Fig. 3.8(b) shows that 64% of the DD-paths were exercised. The JAVS-2D cumulative results from processing the complete 200-line input source are shown in Fig. 3.9. The uninvoked modules (10 for JAVS-2D), are reported by JAVS following the summary results. In this test the uninvoked modules dealt with JOVIAL statement types which were not present in the test coverage input source but which had been present in the complete input source used for the functional test.

The summary DD-path coverage report is intended to provide a concise overall picture of the level of exercise attained by the program's execution of test data. It is usually the first path coverage report analyzed by the tester and focusses his attention on modules that were not invoked at all or those with poor path coverage.

```
'', JAVSTEXT LABEL COMPUTE (COMPOL)''                                TEST    2     1
      '' TEST CASE TO TEST LABEL PROBLEMS ''                         TEST    3     2
 START                                                               TEST    4     3
 PROC TSTLBL (IDUMMY)$                                               TEST    5     4
 ITEM IDUMMY I 24 S$                                                 TEST    6     5
 ITEM INDX I 24 S$                                                   TEST    7     6
AAAAAAAAAAA. BEGIN                                                   TEST    8     7
     INDX = -1$                                                      TEST    9     8
 ALABEL.                                                             TEST   10     9
     INDX = INDX+1$                                                  TEST   11    10
     GOTO LABELE$                                                    TEST   12    11
     GOTO LITTLE$                                                    TEST   13    12
     IF INDX EQ 3$                                                   TEST   14    13
     GOTO CLABEL$                                                    TEST   15    14
     GOTO BLABEL$                                                    TEST   16    15
     GOTO SWTCH ($INDX$) $                                           TEST   17    16
     IF INDX EQ 5$                                                   TEST   18    17
     GOTO ELABEL$                                                    TEST   19    18
     SWITCH SWTCH = (ALABEL,BLABEL,LITTLE)$                          TEST   20    19
 BLABEL.                                                             TEST   21    20
     INDX = INDX+1$                                                  TEST   22    21
     CLOSE  LABELE $                                                 TEST   23    22
      BEGIN                                                          TEST   24    23
         GOTO CLABEL$                                                TEST   25    24
     END                                                             TEST   26    25
     GOTO ALABEL$                                                    TEST   27    26
 CLABEL.                                                             TEST   28    27
     INDX = -1$                                                      TEST   29    28
     GOTO CHOICE ($INDX+1$)$                                         TEST   30    29
     SWITCH CHOICE = (DLABEL,BLABEL,CLABEL,DLABEL)$                  TEST   31    30
 DLABEL.                                                             TEST   32    31
     INDX = INDX+1$                                                  TEST   33    32
     GOTO ALABEL$                                                    TEST   34    33
     END                                                             TEST   35    34
     TERM$                                                           TEST   36    35

MODULES DEFINED IN TEXT
   1   TSTLBL  (LABEL   )
   2   LABELE  (LABEL   )
```

Figure 3.7. Source Code for Testcase 1

JAVS REPORT FOR DD-PATH SUMMARY. 7 CASE(S)
ALL SPECIFIED MODULES ENCOUNTERED ON THE AUDIT FILE 06/29/78

TEST 1 1 06/29/78			SUMMARY -- THIS TEST			CUMULATIVE SUMMARY			
MODULE NAME	JAVSTEXT NAME	NUMBER OF DD-PATHS	NUMBER OF INVOCATIONS	DD-PATHS TRAVERSED	PER CENT COVERAGE	NUMBER OF TESTS	NUMBER OF INVOCATIONS	DD-PATHS TRAVERSED	PER CENT COVERAGE
FOWE	J2C	7	1	6	85	1	2	6	85
AGET	J2C	3	0	0	0	1	0	0	0
BOPT	J2C	77	258	46	59	1	258	46	59
BJMOVE	J2C	11	147	7	63	1	147	7	63
CBMOVE	J2C	4	523	2	50	1	523	2	50
CGET	J2C	17	2671	12	70	1	2671	12	70
CJMOVE	J2C	7	36	2	28	1	36	2	28
CPEEK	J2C	1	0	0	0	1	0	0	0
CPOST	J2C	1	0	0	0	1	0	0	0
CSIGN	J2C	8	2493	4	50	1	2493	4	50
CTYPE	J2C	1	2493	1	100	1	2493	1	100
DNEXT	J2C	3	0	0	0	1	0	0	0
LPOP	J2C	4	0	0	0	1	0	0	0
LPUSH	J2C	3	0	0	0	1	0	0	0
QSAVE	J2C	3	0	0	0	1	0	0	0
ERR	J2C	14	35	3	21	1	35	3	21
JAMOVE	J2C	3	0	0	0	1	0	0	0
JFIND	J2C	19	109	14	73	1	109	14	73
JGET	J2C	19	201	10	52	1	201	10	52
JGETCOM	J2C	26	1	10	38	1	1	10	38
JGETERP	J2C	32	10	3	9	1	10	3	9
JGETLIT	J2C	8	0	0	0	1	0	0	0
JGETSTM	J2C	104	147	48	46	1	147	48	46
JSAVE	J2C	14	53	10	71	1	53	10	71
JSTMBOL	J2C	20	201	9	45	1	201	9	45
JSMOVE	J2C	62	147	50	80	1	147	50	80
LINOUT	J2C	5	39	4	80	1	39	4	80
LPOP	J2C	5	1	3	60	1	1	3	60
LPUSH	J2C	4	2	2	50	1	2	2	50
MBRG	J2C	3	148	3	100	1	148	3	100
MEND	J2C	1	1	1	100	1	1	1	100
MEXHAUST	J2C	5	1	3	60	1	1	3	60
MNAME	J2C	3	3	2	66	1	3	2	66
MREDUCE	J2C	3	0	0	0	1	0	0	0
MSTAME	J2C	11	32	9	81	1	32	9	81
MTESTBEG	J2C	23	32	12	52	1	32	12	52
MTESTEND	J2C	15	32	11	73	1	32	11	73
SEND	J2C	75	32	30	40	1	32	30	40
SGET	J2C	8	30	4	50	1	30	4	50
SGTIDIR	J2C	15	0	0	0	1	0	0	0
SGETJOV	J2C	91	30	42	46	1	30	42	46
STESTEND	J2C	23	26	17	73	1	26	17	73
TGET	J2C	30	2493	13	43	1	2493	13	43
***** ALL *****		791		393	49			393	49

Figure 3.8(a). Path Coverage Summary Report for JAVS-2C (Testcase 1)

JAVS REPORT FOR DD-PATH SUMMARY. 7 CASE(S)

ALL SPECIFIED MODULES ENCOUNTERED ON THE AUDIT FILE 07/11/78

CASE 1 07/11/78			SUMMARY -- THIS TEST			CUMULATIVE SUMMARY			
MODULE NAME	JAVSTEXT NAME	NUMBER OF DD-PATHS	NUMBER OF INVOCATIONS	DD-PATHS TRAVERSED	PER CENT COVERAGE	NUMBER OF TESTS	NUMBER OF INVOCATIONS	DD-PATHS TRAVERSED	PER CENT COVERAGE
FTWO	J2D	7	1	6	85	1	1	6	85
LINKSLT	J2D	5	11	3	60	1	11	3	60
MAKE	J2D	31	11	20	64	1	11	20	64
MODOPEN	J2D	6	4	3	50	1	4	3	50
NEXTTOK	J2D	21	216	15	71	1	216	15	71
SETCAT	J2D	5	37	4	80	1	37	4	80
SETSEQ	J2D	7	37	2	28	1	37	2	28
SLTZAP	J2D	3	11	3	100	1	11	3	100
STAGET	J2D	1	64	1	100	1	64	1	100
STAFIN	J2D	5	32	2	40	1	32	2	40
STAPUT	J2D	3	105	3	100	1	105	3	100
STBZAP	J2D	3	11	3	100	1	11	3	100
TEXIGET	J2D	1	43	1	100	1	43	1	100
TEXTPUT	J2D	1	25	1	100	1	25	1	100
TOKDEFAU	J2D	5	186	5	100	1	186	5	100
SYMDEF	J2D	5	2	4	80	1	2	4	80
STACEF	J2D	15	32	13	86	1	32	13	86
DEFCLOSE	J2D	7	1	4	57	1	1	4	57
DEFINSW	J2D	7	2	4	57	1	2	4	57
DEFITEM	J2D	13	2	4	30	1	2	4	30
DEFLABEL	J2D	7	5	5	71	1	5	5	71
SYMREF	J2D	5	2	4	80	1	2	4	80
LOCSLT	J2D	10	24	10	100	1	24	10	100
MATCHSP	J2D	13	26	10	76	1	26	10	76
REFCLOSE	J2D	5	1	3	60	1	1	3	60
REFEXP	J2D	13	8	10	76	1	8	10	76
REFGOTO	J2D	7	10	5	71	1	10	5	71
REFINDEX	J2D	21	2	14	66	1	2	14	66
REFINSW	J2D	21	2	12	57	1	2	12	57
REFINV	J2D	28	0	0	0	1	0	0	0
REFLABEL	J2D	9	5	7	77	1	5	7	77
REFNULL	J2D	3	32	3	100	1	32	3	100
SETREF	J2D	9	26	7	77	1	26	7	77
STAREF	J2D	22	32	18	81	1	32	18	81
***** ALL *****		324		209	64			209	64

Figure 3.8(b). Path Coverage Summary Report for JAVS-2D (Testcase 1)

CASE 7 07/11/78

MODULE NAME	JAVSTEXT NAME	NUMBER OF DD-PATHS	SUMMARY -- THIS TEST: NUMBER OF INVOCATIONS	DD-PATHS TRAVERSED	PER CENT COVERAGE	CUMULATIVE SUMMARY: NUMBER OF TESTS	NUMBER OF INVOCATIONS	DD-PATHS TRAVERSED	PER CENT COVERAGE
FTWO	J2D	7	1	4	57	6	6	6	85
LINKSLT	J2D	5	0	0	0	6	27	3	60
MAKE	J2D	31	0	0	0	6	27	25	80
MODOPEN	J2D	6	2	3	50	6	24	6	100
NEXTTOK	J2D	21	6	12	57	6	1354	17	80
SETCAT	J2D	5	0	0	0	6	78	4	80
SETSEQ	J2D	7	0	0	0	6	78	7	100
SLIZAP	J2D	3	0	0	0	6	27	3	100
STAGET	J2D	1	4	1	100	6	398	1	100
STAPIN	J2D	5	2	2	40	6	198	4	80
STAPUT	J2D	3	6	2	66	6	655	3	100
STBZAP	J2D	3	0	0	0	6	27	3	100
TEXTGET	J2D	1	2	1	100	6	248	1	100
TEXTPUT	J2D	1	0	0	0	6	131	1	100
TOKDEFAU	J2D	5	6	3	60	6	1226	5	100
SYMDEF	J2D	5	1	4	80	6	12	4	80
STADEF	J2D	15	2	8	53	6	198	13	86
DEFCLOSE	J2D	7	0	0	0	6	3	4	57
DEFINSW	J2D	7	0	0	0	6	3	4	57
DEFITEM	J2D	13	0	0	0	6	30	4	30
DEFLABEL	J2D	7	0	0	0	6	9	5	71
SYMREF	J2D	5	1	4	80	6	12	4	80
LOCSLI	J2D	10	0	0	0	6	48	10	100
MATCHSP	J2D	13	0	0	0	6	51	11	84
REFCLOSE	J2D	5	0	0	0	6	3	3	60
REFEXP	J2D	13	0	0	0	6	57	11	84
REFGOTO	J2D	7	0	0	0	6	14	5	71
REFINDEX	J2D	21	0	0	0	6	3	14	66
REFINSW	J2D	21	0	0	0	6	3	12	57
REFINV	J2D	28	0	0	0	6	2	16	57
REFLABEL	J2D	9	0	0	0	6	9	7	77
REFNULL	J2D	3	2	3	100	6	198	3	100
SETREF	J2D	9	0	0	0	6	51	7	77
STAREF	J2D	22	2	9	40	6	198	19	86
***** ALL *****		324		56	17			245	75

Figure 3.9. Path Coverage Summary Report for JAVS-2D (Testcase 7)

Subsequent to the summary report, the tester may want to review the single module reports which show the number of executions of each DD-path and each statement (separate reports) accumulated from the entire test input data, along with the report showing DD-paths not executed during each separate test case for all invoked modules. Additional execution performance information can be obtained by requesting module invocation and/or DD-path tracing reports, a report showing execution time spent in each module during the test, and reports showing the DD-path execution count by module and testcase.

To demonstrate the usage of JAVS path coverage reports, consider the following analysis. The DD-path summary for JAVS-2D in Fig. 3.9 showed that module MAKE achieved 80% path coverage and has the most DD-paths in the subcomponent. The functional description for MAKE states that it creates the symbol table and symbol locator table entries (control symbols only) of designated symbol types. A partial listing of the cumulative DD-path coverage results is shown in Fig. 3.10. The cumulative execution counts are given in the far right column. In the partial listing, five DD-paths were never exercised. This report includes the first statement of each DD-path and shows the importance of including an informative comment at decision points. Retesting can be particularly aided when comments refer to characteristics of the input data.

The DD-path coverage listing should be analyzed along with a functional description of the module, a listing of the input data used (identified by testcase boundaries), the program's execution output, and the JAVS "not hit" and statement coverage reports.

For the JAVS-2D test, the "not hit" report in Fig. 3.11 shows which DD-paths in module MAKE were not executed by the testcases. Paths 4, 10, 12, 17, 20 and 27 were never hit. Each of these paths should be analyzed to determine if they can be executed and whether additional test data should be derived.

DD-path 4 depends upon the current module in the input data being a COMPOOL. A COMPOOL was provided, but it did not contain any control symbols (such as a PROC declaration). Thus the addition of a PROC declaration to the COMPOOL should execute Path 4.

Proper analysis of DD-path 10 requires review of the statement coverage (or other JAVS module) listing in Fig. 3.12 and the functional description document. This review shows that DD-path 10 cannot be executed since the conditions leading up to path 10 are (1) the symbol is global (DD-path 2 is true), (2) the module is not a COMPOOL (DD-paths 5 and 6 are true), and (3) the symbol is being referenced, not declared (DD-path 9 is true).

DD-path 12 is also an example of a DD-path which cannot be hit. This one, however, can be analyzed merely by looking at Paths 2, 3, and 11, 12, at statements 16 and 32, respectively. If FLLOCAL equals 0, then DD-path 2 is true, and neither paths 11 or 12 will be executed. If FLLOCAL is not 0, then DD-paths 3 and 11 will be executed. In neither case will Path 12 be hit.

The last two untested paths in Fig. 3.10 deal with control symbols in formal parameter lists. The two general types of JOVIAL parameters allowed

```
MODULE DD-PATH COVERAGE LISTING

MODULE <MAKE    >, JAVSTEXT <J2D     >, PARENT MODULE <FTWO    >

 NO.  LVL          STATEMENT                                    DD-PATHS GENERATED                          COVERAGE
---------------------------------------------------------------------------------------------------------------------
   1 ( 0)          PROC MAKE ( STYPE ) $ ''MAKE A SYMBOL OF TYPE SPECIFIED''
                                                                ** DD-PATH    1 IS PROCEDURE ENTRY            27

  16 ( 1)            IFEITH FLLOCAL EQ 0 $
                                                                ** DD-PATH    2 IS TRUE BRANCH                15
                                                                ** DD-PATH    3 IS FALSE BRANCH               12

  18 ( 2)              IFEITH MDBS ($ 2 $) EQ 4H(CMPL) $
                                                                ** DD-PATH    4 IS TRUE BRANCH                 0 *
                                                                ** DD-PATH    5 IS FALSE BRANCH               15

  22 ( 2)              ORIF 1 $
                                                                ** DD-PATH    6 IS TRUE BRANCH                15

  24 ( 3)                IFEITH FLTOK EQ 0 $ ''DECLARATIONS''
                                                                ** DD-PATH    7 IS TRUE BRANCH                 8
                                                                ** DD-PATH    8 IS FALSE BRANCH                7

  26 ( 3)                ORIF FLTOK EQ 1 $ ''REFERENCES''
                                                                ** DD-PATH    9 IS TRUE BRANCH                 7
                                                                ** DD-PATH   10 IS FALSE BRANCH                0 *

  28 ( 3)                END
  30 ( 2)              END
  32 ( 1)            ORIF FLLOCAL NQ 0 $
                                                                ** DD-PATH   11 IS TRUE BRANCH                12
                                                                ** DD-PATH   12 IS FALSE BRANCH               0 *

  36 ( 1)            END
  37 ( 1)            IF MDBS ($ 2 $) NQ 4H(CMPL) AND MDBI ($ 1 $) EQ 0 $
                                                                ** DD-PATH   13 IS TRUE BRANCH                 9
                                                                ** DD-PATH   14 IS FALSE BRANCH               18

  46 ( 1)            IFEITH FLPAR EQ 0 $ ''NOT FORMAL PARAMETER''
                                                                ** DD-PATH   15 IS TRUE BRANCH                26
                                                                ** DD-PATH   16 IS FALSE BRANCH               1

  50 ( 1)            ORIF FLPAR EQ 1 $ ''INPUT PARAMETER''
                                                                ** DD-PATH   17 IS TRUE BRANCH                 0 *
                                                                ** DD-PATH   18 IS FALSE BRANCH               1

  54 ( 1)            ORIF FLPAR EQ 2 $ ''OUTPUT PARAMETER''
                                                                ** DD-PATH   19 IS TRUE BRANCH                1
                                                                ** DD-PATH   20 IS FALSE BRANCH               0 *
```

Figure 3.10. Partial Listing of DD-path Coverage Report for Module MAKE

JAVS REPORT FOR DD-PATHS NOT EXECUTED. 7 CASE(S) 07/11/78

MODULE NAME	JAVSTEXT NAME	TEST IDENTIFICATION		PATHS NOT HIT	LIST OF DECISION-TO-DECISION PATHS NOT EXECUTED
FTWO	J2D	CASE 1	07/11/78	1	3
		CASE 2	07/11/78	1	3
		CASE 3	07/11/78	3	3 4 6
		CASE 4	07/11/78	3	3 4 6
		CASE 5	07/11/78	1	3
		CASE 6	07/11/78	3	3 4 6
		CASE 7	07/11/78	ALL	
		TOTAL NOT HIT		1	3
LINKSLT	J2D	CASE 1	07/11/78	2	2 4
		CASE 2	07/11/78	2	2 4
		CASE 3	07/11/78	ALL	
		CASE 4	07/11/78	ALL	
		CASE 5	07/11/78	2	2 4
		CASE 6	07/11/78	ALL	
		CASE 7	07/11/78	ALL	
		TOTAL NOT HIT		2	2 4
MAKE	J2D	CASE 1	07/11/78	11	4 10 12 16 17 18 19 20 21 27 30
		CASE 2	07/11/78	7	4 10 12 17 20 27 30
		CASE 3	07/11/78	ALL	
		CASE 4	07/11/78	ALL	
		CASE 5	07/11/78	10	4 10 12 16 17 18 19 20 21 27
		CASE 6	07/11/78	ALL	
		CASE 7	07/11/78	ALL	
		TOTAL NOT HIT		6	4 10 12 17 20 27
MODOPEN	J2D	CASE 1	07/11/78	3	3 4 6
		CASE 2	07/11/78	2	3 6
		CASE 3	07/11/78	3	3 4 6
		CASE 4	07/11/78	3	2 4 5
		CASE 5	07/11/78	2	3 6
		CASE 6	07/11/78	3	3 4 6
		CASE 7	07/11/78	ALL	
NEXTTOK	J2D	CASE 1	07/11/78	6	6 8 9 16 17 18
		CASE 2	07/11/78	6	6 8 9 16 17 18
		CASE 3	07/11/78	7	9 13 15 16 17 18 21
		CASE 4	07/11/78	6	6 8 9 16 17 18
		CASE 5	07/11/78	6	6 8 9 16 17 18
		CASE 6	07/11/78	9	6 8 9 13 15 16 17 18 21
		CASE 7	07/11/78	ALL	
		TOTAL NOT HIT		4	9 16 17 18

Figure 3.11. Partial Listing of DD-paths not Executed in JAVS-2D Modules

```
NO.  LVL  STATEMENT                                                                  DD-PATHS        CONTROL
 1 ( 0)   PROC MAKE ( STYPE ) $ ''MAKE A SYMBOL OF TYPE SPECIFIED''                  (  1)      27
 2 ( 0)   ITEM STYPE H 4 $ ''TYPE OF SYMBOL''                                                27
 3 ( 1)     BEGIN                                                                            27      IBV
 4 ( 1)     SLIZAP $                                                                         27      IBV
 5 ( 1)     STRZAP $                                                                         27
 6 ( 1)     BYTE ($ 0 , 4 $) ( SLTS ($ 0 $) ) = BYTE ($ 0 , 8 $) ( SS ) $                    27
 7 ( 1)     BYTE ($ 0 , 4 $) ( SLTS ($ 1 $) ) = BYTE ($ 4 , 8 $) ( SS ) $                    27
 8 ( 1)     SLTS ($ 2 $) = TOBS ($ 30 $) $                                                   27
 9 ( 1)     SLTS ($ 3 $) = MOBS ($ 31 $) $                                                   27
10 ( 1)     SLTS ($ 4 $) = MOBS ($ 32 $) $                                                   27
11 ( 1)     SLTS ($ 5 $) = MOBS ($ 33 $) $                                                   27
12 ( 1)     MOBI ($ 14 $) = MOBI ($ 14 $) + 1 $                                              27
13 ( 1)     MOBI ($ 17 $) = MOBI ($ 17 $) + 1 $                                              27
14 ( 1)     SLTI ($ 6 $) = MOBI ($ 14 $) $                                                   27
15 ( 1)     STBI ($ 0 $) = NUMSTOF + 1 $                                     (  2-  3)       27      IFEI
16 ( 1)     IFEITH FLLOCAL EQ 0 $                                                            15
17 ( 2)       BEGIN                                                          (  4-  5)       15      IFEI
18 ( 2)       IFEITH MOBS ($ 2 $) EQ 4H(CMPL) $                                              0 *
19 ( 3)         BEGIN                                                                        0 *
20 ( 3)         STBS ($ 1 $) = 4H(CMPL) $                                                    0 *
21 ( 3)         END                                                            (  6)         15      OBIF
22 ( 2)       ORIF 1 $                                                                       15
23 ( 3)         BEGIN                                                        (  7-  8)       15      IFEI
24 ( 3)         IFEITH FLTOK EQ 0 $ ''DECLARATIONS''                                         8
25 ( 4)           STBS ($ 1 $) = 4H(GLBL) $                                  (  9- 10)       7       OBIF
26 ( 3)         ORIF FLTOK EQ 1 $ ''REFERENCES''                                             7
27 ( 4)           STBS ($ 1 $) = 4H(CMPL) $                                                  15
28 ( 3)         END                                                                          15
29 ( 3)         END                                                                          15
30 ( 2)       END                                                                            15
31 ( 2)       END                                                            ( 11- 12)       12      OBIF
32 ( 1)     ORIF FLLOCAL NQ 0 $                                                            12
33 ( 2)       BEGIN                                                                        12
34 ( 2)       STBS ($ 1 $) = 4H(LOCL) $                                                    12
35 ( 2)       END                                                                          27
36 ( 1)     END                                                              ( 13- 14)       27      IF
37 ( 1)     IF MOBS ($ 2 $) NQ 4H(CMPL) AND MOBI ($ 1 $) EQ 0 $                            9
38 ( 2)       BEGIN                                                                        9
39 ( 2)       MOBI ($ 1 $) = MOBI ($ 17 $) $ ''ASSUME FIRST SLT IS MODULE'S''              9
40 ( 2)       END                                                                          27
41 ( 1)     STBS ($ 2 $) = STYPE $                                                         27
42 ( 1)     STBI ($ 21 $) = NUMCUST $                                                      27
43 ( 1)     STBS ($ 22 $) = MOBS ($ 30 $) $                                                27
44 ( 1)     STBS ($ 23 $) = MOBS ($ 31 $) $                                                27
45 ( 1)     STBI ($ 29 $) = MOBI ($ 17 $) $                                                27
46 ( 1)     IFEITH FLPAR EQ 0 $ ''NOT FORMAL PARAMETER''                     ( 15- 16)       27      IFEI
47 ( 2)       BEGIN                                                                        26
48 ( 2)       STBS ($ 28 $) = 4H(    ) $                                                   26
49 ( 2)       END                                                                          26
50 ( 1)     ORIF FLPAR EQ 1 $ ''INPUT PARAMETER''                            ( 17- 18)       1       OBIF
51 ( 2)       BEGIN                                                                        0 *
52 ( 2)       STBS ($ 28 $) = 4H(INPR) $                                                   0 *
53 ( 2)       END                                                                          0 *
54 ( 1)     ORIF FLPAR EQ 2 $ ''OUTPUT PARAMETER''                           ( 19- 20)       1       OBIF
55 ( 2)       BEGIN                                                                        1
56 ( 2)       STBS ($ 28 $) = 4H(OTPR) $                                                   1
57 ( 2)       END                                                                          1
58 ( 1)     END                                                                            27
```

Figure 3.12. Partial Listing of Statement Coverage for Module MAKE

are input or output. In the source input text, there was an occurrence of a label output parameter (the only allowable control symbol output parameter). To exercise DD-path 17, the source text must include a CLOSE name as an input parameter.

The analysis described in the above paragraphs is the type of procedure undertaken to evaluate the thoroughness of a program's execution. Sometimes additional JAVS documentation reports, such as the module invocation, reaching set or symbol cross reference, are needed to understand the conditions required to execute specific paths.

An integral part of program path testing is the knowledge of the proper and complete function of the program. As was seen in the step-by-step analysis of paths in module MAKE, some paths cannot logically or functionally be executed. In addition, it is often the case that certain combinations of paths should be tested, rather than merely striving for the minimal set of combinations covering a set of DD-paths. These considerations should be made with respect to the module or program's proper function.

4 FUTURE EFFORT

JAVS is currently a powerful software tool for providing static and dynamic analyses for JOVIAL J3 programs that would be impractical or impossible to achieve manually. Its design and implementation allow extensions in syntax recognition and functional capability. The modularity and clear definition of processors allow interchange of software components when new techniques prove themselves better (as demonstrated by the implementation of the new syntax analyzer).

We feel that the capabilities JAVS now has offer a valuable asset in the testing, maintenance and design of high quality JOVIAL software. Effort should be expended in further disseminating the tool and training JOVIAL programmers in its usage. An important part of improving the acceptance of JAVS as a useful JOVIAL software tool would be the development of a document on how to design high quality software with JAVS' assistance. A workshop which covers software development, testing and maintenance should be conducted to stimulate current and new users of JAVS to take advantage of the tool's capabilities.

Several static and dynamic testing techniques can be added to JAVS' current features which would increase its automation and provide more stringent measures of software testing. Static data flow analysis, available in GRC's Fortran Automated Verification System (FAVS),[6] can locate use-before-set errors (uninitialized variables), a common source of program errors which can be difficult to find manually in large programs. Physical units consistency checking can be very useful in mathematical programs using engineering units. This static analysis feature is available in the SQLAB tool[7] and can be efficiently included in JAVS along with set/use checking, since both analyses require much of the same software. Automatic generation of certain types of assertions and coverage measurement of program functions as specified through comments are two dynamic testing techniques which can be incorporated into JAVS.

The above-described added capabilities for JAVS would continue the pursuit of increasing testing thoroughness. Without ignoring that goal, we feel that computer resources should be kept to a minimum. JAVS now operates in 53,000 words of primary core storage, which is not excessive. We have found that processing time can be greatly reduced in FAVS without any loss of capability. Since the two systems (FAVS and JAVS) have similar designs, we are confident that JAVS' processing time can also be reduced.

4.1 JAVS WORKSHOP

A two- or three-day workshop for JOVIAL programmers which integrates the concepts of good software design and testing in the JAVS environment would be very beneficial in increasing the utilization of the tool. A workshop would also provide the opportunity to coordinate and formalize quality programming practices and their effect on AVS-supported testing while getting immediate feedback from workshop participants.

The collection of JAVS documents listed in Sec. 4.1 include guidelines for using JAVS (User's Guide) and a demonstrative testing methodology which is supported by JAVS (Methodology Report). Nevertheless, what seems to be lacking is a discussion of how JAVS contributes and should be used in the full range of software life-cycle activities.

4.2 STATIC DATA FLOW ANALYSIS

Most JOVIAL J3 compilers perform rigorous checks on the consistency between declared and actual usage of symbols and parameters. Other control flow anomolies, not detected by the compiler but reported by JAVS, are structural infinite loops and unreachable code. One common source of errors which is currently undetected by either compiler or JAVS is the usage of uninitialized variables. These "use-before-set" errors can be very difficult to detect manually in programs with complex control structures.

A small function containing the uninitialized variable SUM is shown in Fig. 4.1 along with the static analysis in Fig. 4.2. The type of static analysis recommended for JAVS is the set/use checking demonstrated in the lower half of Fig. 4.2.

Static analysis is already available in FAVS, which was developed by GRC under RADC sponsorship. Most of the required software for JAVS set/use analysis could be supplied by the FAVS-translated-FORTRAN source code. Two approaches can be taken to incorporate the enhancement into JAVS: the syntax analyzer can make a third pass to supply the required symbol information or the current symbol cross-reference table information can be adapted. In either case, the static set/use checking can be performed in a separate overlay load module, thereby not increasing the current core storage requirement.

4.3 PHYSICAL UNITS CHECKING

Requiring that local and global variables be specified in terms of the physical units they represent (if any) allows comprehensive checking of the consistency of units. This type of checking is particularly relevant to technical software where many physical properties are represented and there are many possibilities of confusion over units. Units can be checked on a multi-module basis if each module contains a description of the units for each physical variable to which it refers. The form of the description for JAVS would be:

".UNITS (<variable list-1> = <units-expression-1>,

<variable list-2> = units-expression-2>, ...)"

The units directive would be placed in the source code by the user. JAVS would, if directed by user commands, perform the following analysis during the static set/use checking. An inconsistency in units is indicated if unlike units are added, subtracted, or compared. The physical-units analysis compares the right and left side of assignment statements, the right and left side of relational operations, and actual and formal parameters. For convenience in stating UNITS assertions, all constants are assumed to be unitless,

```
      SUBROUTINE SETUSE
C
      RADIUS = DIAMTR / 2
      AREA = PI * RADIUS ** 2
      PRINT 1, ( RADIUS, AREA )
1     FORMAT ( 2 (F6.2) )
      RETURN
      END
```

Figure 4.1. Listing of Subroutine SETUSE

STATIC ANALYSIS CONT... SUBROUTINE SETUSE

NAME	CLASS	MODE	1ST STMT	TOTAL USES	LAST STMT	ASSERTED USE	ACTUAL USE	PHYSICAL UNITS
RADIUS	LOCAL	REAL	3	3	5			
DIAMTR	LOCAL	REAL	3	1	3			
- VARIABLE DIAMTR						SET/USE ERROR USED BEFORE BEING ASSIGNED A VALUE		
AREA	LOCAL	REAL	4	2	5			
PI	LOCAL	REAL	4	1	4			
- VARIABLE PI						SET/USE ERROR USED BEFORE BEING ASSIGNED A VALUE		

SYMBOL ANALYSIS SUMMARY	ERRORS	WARNINGS
SET/USE CHECKING	2	0

Figure 4.2. Static Analysis of Subroutine SETUSE

except for zero, which will match any units expression. A variable is declared unitless by stating that its units expression is the constant 1, as in UNITS (PI = 1). Figure 4.3 shows a small program subroutine containing specified units. An inconsistency was detected in which area was computed as:

FEET * FEET = INCHES * INCHES

Note that the user's units "FEET**2" was changed to "FEET * FEET" in the units error message. Units checking is capable of simplifying terms for cancellation of units.

This output was generated by SQLAB,[7] a GRC-developed software tool which provides software quality measurements, dynamic and static testing, and verification through symbolic execution for PASCAL and FORTRAN. The techniques and software used for SQLAB's units checking can be implemented in JAVS.

4.4 AUTOMATIC ASSERTION GENERATION

Two types of dynamic assertion statements can increase the thoroughness of testing and quality level of the software. Both types of assertion could be generated automatically by JAVS with a small amount of modification. The assertions are array bounds checking and compound predicate analysis. The situation is best described by example.

Figure 4.4 is a module listing which contains several JOVIAL statements which lend themselves to analysis via assertions. In Statements 3, 10, 12, 14, 16, and 17 are array or table references with computed subscripts. If any of the subscripts are out of the boundaries declared in this module's COMPOOL, the adjacent core locations will be erroneously over-written. JAVS could automatically generate a JAVS computation directive:

".EXPECT, <name> = <low value>, <high value> "

at each array or table reference. JAVS would pick up the low and high values from the declaration statement and instrumentation would expand the assertion into executable code.

The effect of the assertion is that during the program's execution, any occurrence of an out-of-bounds subscript would produce a line of output giving the subscript's name and computed value. JAVS could also provide the module name and statement number of array or table overflow.

Statements 20 and 24 in Fig. 4.4 contain numeric subscripts. These would be less prone to error than computed subscripts, but it might be wise to automatically generate the EXPECT assertions for numeric subscripts as well.

For thorough testing, JOVIAL statements which contain compound predicates such as those in Statement 3, 10, 14, and 24 in Fig. 4.4 should be executed once for each disjunctive term. For example,

IFEITH ZT EQ PD ($PPBEGINS$) OR ZT EQ PD ($PPIFEITH$) $

```
STATIC ANALYSIS                          SUBROUTINE CIRCLE (RADIUS, AREA )

 STMT     IDENT.LINE  SOURCE...
-------------------------------------------------------------------------------
    5                      AREA = PI * RADIUS**2
                    -------------------------------------------------------------
                    -                          UNITS ERROR                      -
                    -      =     OPERATION WITH INCONSISTENT UNITS               -
                    -      ( FEET * FEET )                                      -
                    -      ( INCHES * INCHES )                                  -
                    -------------------------------------------------------------
-------------------------------------------------------------------------------
                     STATEMENT ANALYSIS SUMMARY        ERRORS  WARNINGS
                     --------------------------        ------  --------
                     GRAPH CHECKING                         0         0
                     CALL CHECKING                          0         0
                     UNITS CONSISTENCY                      1         0
                     MODE CHECKING                          0         0

  NAME       SCOPE          TYPE        MODE        USE       OTHER INFORMATION...
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
                     SYMBOL ANALYSIS SUMMARY           ERRORS  WARNINGS
                     -----------------------           ------  --------
                     INPUT/OUTPUT CHECKING                  0         0
                     SET/USE CHECKING                       0         0

        THE FOLLOWING NONLOCAL VARIABLES ARE SET...
        AREA
```

The above report resulted from this routine:

```
STATEMENT LISTING                        SUBROUTINE CIRCLE (RADIUS, AREA )

 STMT     IDENT.LINE  SOURCE...
-------------------------------------------------------------------------------
    1                      SUBROUTINE CIRCLE (RADIUS, AREA )
    2                      DATA PI / 3.1416 /
    3                      UNITS ( RADIUS=INCHES, AREA=FEET**2, PI=1 )
    4                      INPUT (/R/ RADIUS )
    5                      AREA = PI * RADIUS**2
    6                      OUTPUT (/R/ AREA )
    7                      RETURN
    8                      END
-------------------------------------------------------------------------------
```

Figure 4.3. Physical Units Checking

```
MODULE STATEMENT LISTING

MODULE <MTESTEND>, JOVSTEXT <J2C     >, PARENT MODULE <FONE    >

 NO   LVL            STATEMENT                                                    DD-PATHS          CONTROL
---------------------------------------------------------------------------------------------------------
  1 ( 0)          PROC MTESTEND $ ''TEST MODULE END''                            (  1)
  2 ( 1)            BEGIN
  3 ( 1)            IFEITH ZT EQ PD ($ PRTERM $) OR ZT EQ STENDFILE $            (  2-  3)        IFEI
  4 ( 2)              BEGIN
  5 ( 2)              MEXHAUST $                                                                   INV
  6 ( 2)              LZAP $                                                                       INV
  7 ( 2)              END
  8 ( 1)            ORIF FLCOMPOOL EQ 0 $                                        (  4-  5)        ORIF
  9 ( 2)              BEGIN
 10 ( 2)              IFEITH ZT EQ PD ($ PRBEGIN $) OR ZT EQ PD ($ PPIFEITH $) $ (  6-  7)        IFEI
 11 ( 3)                BEGIN
 12 ( 3)                LBEG ($ LSTK $) = LBEG ($ LSTK $) + 1 $
 13 ( 3)                END
 14 ( 2)              ORIF ZT EQ PD ($ PPEND $) OR ZT EQ STPRST $                  (  8-  9)        ORIF
 15 ( 3)                BEGIN
 16 ( 3)                LBEG ($ LSTK $) = LBEG ($ LSTK $) - 1 $
 17 ( 3)                IF LBEG ($ LSTK $) LQ 0 AND MODULE NQ MODST AND LSTK GR 0 $ ( 10- 11)        IF
 18 ( 4)                  BEGIN
 19 ( 4)          '' TEST FOR IMBEDDED MODULE CLOSURE''
 20 ( 4)                  IFEITH MDBI ($ 10 $) GR 0 $ '' EXECUTABLE CODE COMPLETE'' ( 12- 13)        IFEI
 21 ( 5)                    BEGIN
 22 ( 5)                    MEND $                                                                   INV
 23 ( 5)                    END
 24 ( 4)                  ORIF MDBS ($ 2 $) EQ 4H(CLSM) OR MDBS ($ 2 $) EQ 4H(CLSP) $ ( 14- 15)        ORIF
          (''BEGIN-END BALANCED IN A CLOSE'')
 25 ( 5)                    BEGIN
 26 ( 5)                    MEND $                                                                   INV
 27 ( 5)                    END
 28 ( 4)                  END
 29 ( 4)                  END
 30 ( 3)                END
 31 ( 2)              END
 32 ( 2)              END
 33 ( 1)            END
 34 ( 1)            END
```

Figure 4.4. Module Listing for Automatic Assertion Generation Example

should be exercised for both

ZT = PD ($PPBEGIN$) and

ZT = PD ($PPIFEITH$)

JAVS could automatically generate three assertions for the above IFEITH statement:

".ASSERT, ZT EQ PD ($PPBEGIN$) AND ZT NQ PD ($PPIFEITH$)"

".ASSERT, ZT NQ PD ($PPBEGIN$) AND ZT EQ PD ($PPIFEITH$)"

".ASSERT, ZT EQ PD ($PPBEGIN$) AND ZT EQ PD ($PPIFEITH$)"

In this example, the third assertion would never be true, but that may not be the case in general. The currently implemented computation directive instrumentation would translate the assertions into executable code. During execution, the message:

JAVS' ASSERT = ZT EQ PD ($PPBEGIN$) AND

ZT NQ PD ($PPIFEITH$) AT

STMT. xxxx IS TRUE

would be printed. The current JAVS' ASSERT message is printed when the condition is not true, but for compound predicates, it would be more efficient to print the message for the true condition.

For both array/table and compound predicate automatic assertion generation, the user would request the capability via a command, thus being able to turn the option on or off and being able to select one or more modules as targets. Current instrumentation would require modest modification for implementation of these assertions.

4.5 COVERAGE OF PROGRAM FUNCTIONS

Execution coverage reports currently available in JAVS measure DD-path and statement executions. Also available are the module invocation and DD-path trace reports. A combination of these two current features in terms of descriptive functions would be very useful.

Figures 4.5 and 4.6 are execution coverage reports for statements and DD-paths, respectively, for one module in the new JAVS syntax analyzer. Note how useful it is to include descriptive functional comments at each decision. It may be functionally important in a program that a specific sequence of DD-paths be executed. JAVS DD-path coverage analysis may report that each DD-path of interest was executed, but the order of the paths may be crucial to the correctness of the program.

```
MODULE STATEMENT LISTING

MODULE <LOCSLT  >, JAVSTEXT <J2D     >, PARENT MODULE <PTWO    >

 NO.  LVL          STATEMENT                                                DD-PATHS          CONTROL
---------------------------------------------------------------------------------------------------------
   1 ( 0)          PROC LOCSLT $ ''LOCATE SLT IN MODULE AND SUPPLY IF NEEDED''      (  1)      46
   2 ( 0)          ITEM LOCSLT I 24 S $                                                        .
   3 ( 1)            BEGIN                                                          46         .
   4 ( 1)            LOCSLT = 0 $                                                   46         .
   5 ( 1)            IFEITH MODULE EQ SMOD $ ''MODULE IS MODULE OF DECLARATION''    (  2-  3)  46      IFEI
   6 ( 2)              BEGIN                                                        37         .
   7 ( 2)              LOCSLT = SSLT $                                              37         .
   8 ( 2)              END                                                          37         .
   9 ( 1)            ORIF 1 $ ''SEARCH MODULE FOR COPY OF SLT''                     (  4)       7     ORIF
  10 ( 2)              BEGIN                                                        7          .
  11 ( 2)              II = MDBI ($ 17 $) $                                         7          .
  12 ( 2)              JJ = 1 $                                                     7          .
  13 ( 2)       L1.                                                               (  5-  6)  33      IFEI<-----
                       IFEITH JJ LQ II $ ''NOT LAST IN MODULE''                                  .
  14 ( 3)                BEGIN                                                    31         .
  15 ( 3)                KK = IGTWRD ( MODULE , TABSLT , JJ , 11 ) $              31         .
  16 ( 3)                IFEITH KK EQ SLTI ($ 11 $) $ ''FOUND''                   (  7-  8)  31      IFEI
  17 ( 4)                  BEGIN                                                  5          .
  18 ( 4)                  LOCSLT = JJ $                                          5          .
  19 ( 4)                  END                                                    5          .
  20 ( 3)                ORIF 1 $ ''EXAMINE NEXT SLT IN MODULE''                  (  9)      26     ORIF
  21 ( 4)                  BEGIN                                                  26         .
  22 ( 4)                  JJ = JJ + 1 $                                          26         .
  23 ( 4)                  GOTO L1 $                                              26         .---->
  24 ( 4)                  END                                                    0  *       .
  25 ( 3)                END                                                      5          .
  26 ( 3)                END                                                      5          .
  27 ( 2)              ORIF 1 $ ''NOT FOUND, SO COPY SLT''                        (  10)     2      ORIF
  28 ( 3)                BEGIN                                                    2          .
  29 ( 3)                MDBI ($ 17 $) = MDBI ($ 17 $) + 1 $                      2          .
  30 ( 3)                LOCSLT = MDBI ($ 17 $) $                                 2          .
  31 ( 3)                SLTI ($ 7 $) = 0 $                                       2          .
  32 ( 3)                SLTI ($ 8 $) = 0 $                                       2          .
  33 ( 3)                SLTI ($ 9 $) = 0 $                                       2          .
  34 ( 3)                PUTBLK ( MODULE , TABSLT , LOCSLT , SLTI ) $             2          IFV
  35 ( 3)                END                                                      2          .
  36 ( 2)              END                                                        7          .
  37 ( 2)              END                                                        7          .
  38 ( 1)            END                                                          44         .
  39 ( 1)            END                                                          0  *       .
---------------------------------------------------------------------------------------------------------

EXECUTABLE STATEMENTS       38
STATEMENTS EXECUTED         36
  PER CENT EXECUTED         94
```

Figure 4.5. Execution Statement Coverage Report

```
MODULE DD-PATH COVERAGE LISTING

MODULE <LOCSLT  >, JAVSTEXT <J2D     >, PARENT MODULE <FTWO    >

 NO.  LVL          STATEMENT                                    DD-PATHS GENERATED                          COVERAGE
---------------------------------------------------------------------------------------------------------------------
   1 ( 0)          PROC LOCSLT $ ''LOCATE SLT IN MODULE AND SUPPLY IF NEEDED''
                                                                ** DD-PATH   1 IS PROCEDURE ENTRY              44
                   . . . .
   5 ( 1)            IFEITH MODULE EQ SMOD $ ''MODULE IS MODULE OF DECLARATION''
                                                                ** DD-PATH   2 IS TRUE BRANCH                  37
                                                                ** DD-PATH   3 IS FALSE BRANCH                  7
                   . . . .
   9 ( 1)            ORIF 1 $ ''SEARCH MODULE FOR COPY OF SLT''
                                                                ** DD-PATH   4 IS TRUE BRANCH                   7
                     . . . .
  13 ( 2)      L1.
                     IFEITH JJ LQ II $ ''NOT LAST IN MODULE''
                                                                ** DD-PATH   5 IS TRUE BRANCH                  31
                                                                ** DD-PATH   6 IS FALSE BRANCH                 2
                       . . . .
  16 ( 3)              IFEITH KK EQ SLTI ($ 11 $) $ ''FOUND''
                                                                ** DD-PATH   7 IS TRUE BRANCH                  5
                                                                ** DD-PATH   8 IS FALSE BRANCH                 26
                       . . . .
  20 ( 3)              ORIF 1 $ ''EXAMINE NEXT SLT IN MODULE''
                                                                ** DD-PATH   9 IS TRUE BRANCH                  26
                       . . . .
  25 ( 3)              END
                     . . . .
  27 ( 2)            ORIF 1 $ ''NOT FOUND, SO COPY SLT''
                                                                ** DD-PATH  10 IS TRUE BRANCH                  2
                     . . . .
  36 ( 2)            END
                   . . . .
  38 ( 1)          END
---------------------------------------------------------------------------------------------------------------------

TOTAL DDPATHS          10
DDPATHS EXECUTED       10
PERCENT EXECUTED      100
```

Figure 4.6. Execution DD-path Coverage Report

Therefore, if the user can assert that:

1. Locate SLT in module and supply if needed
2. Search module for copy of SLT
3. Found
4. Return

is the specific flow of interest for the module in the figures, then JAVS should translate the supplied functional comments into DD-path numbers and compare the sequence against the actual DD-path trace that occurred during execution. A sample DD-path trace report for a different program is shown in Fig. 4.7. Currently, it is left to the user to find the appropriate sequences of paths. JAVS could be modified to report the existence or omission of the specified functional sequence. The only burden on the user would be to supply concise, functional comments at all DD-paths to be analyzed.

4.6 REDUCTION OF PROCESSING TIME

GRC is currently investigating the possibility of substantially reducing the computer processing time of FAVS. This experience convinces us that JAVS' processing time can also be reduced. Substantial reductions can probably be made in the syntax and structural analyses and in the algorithms used to generate documentation reports.

```
                    TEST-CASE                    SAMPLE   2    2209  08/04/75

        STMT   19     IF READEP NO V(EOF) $                    *DD-PATH   2    TRUE BRANCH

** INVOKED MODULE IS EXMPL1  (EXPROGM )
        STMT    1     PROC EXMPL1 ( LIMIT1 , LIMIT2 ) $        *DD-PATH   1    PROCEDURE ENTRY
        STMT    8     IF LIMIT1 GO 100 $                       *DD-PATH   3    FALSE BRANCH
        STMT   21     IFEITH J LO 3 $                          *DD-PATH   4    TRUE BRANCH
        STMT   24     GOTO PICK (S INDAS - 1 S) $              *DD-PATH   7    SWITCH OUTWAY    1
        STMT   32     ORIF RESULT EQ 4 $                       *DD-PATH  15    FALSE BRANCH
        STMT   30 LABEL1.                                      *DD-PATH  13    FALSE BRANCH
                      IFEITH RESULT LS 4 $
        STMT   34     ORIF 1 $                                 *DD-PATH  16    TRUE BRANCH
        STMT   18     FOR J = 1 , 1 , LIMIT2 $                 *DD-PATH  18    ESCAPE FOR LOOP
        STMT   17     FOR I = 1 , 1 , LIMIT1 $                 *DD-PATH  20    ESCAPE FOR LOOP
** RETURN FROM MODULE EXMPL1  (EXPROGM )

        STMT   26     IF ITER1 GO 100 $                        *DD-PATH   5    FALSE BRANCH
```

Figure 4.7. Excerpt of DD-path Execution Trace

APPENDIX A

INSTALLATION INSTRUCTIONS

This appendix describes procedures for installing JAVS on the HIS 6180 computer under GCOS and for building the JAVS absolute file from source text for non-HIS installations.

A.1 HIS-INSTALLATIONS

The following files should be copied from magnetic tape:

1. HSTAR/JAVSHS - random H* file containing linked JAVS program
2. JAVSXQ - BCD sequential select stream for using JAVS H*file
3. PRPOOL - BCD source for data collection COMPOOL
4. JPROBESX - BCD sequential select stream for loading data collection routine
5. MAKBINPX - ASCII sequential job stream for compiling data collection routines (files 6-10)
6. PRCMPL-X - BCD source for data collection COMPOOL
7. PROBE-X - BCD source for data collection
8. PROBI-X - BCD source for data collection
9. PROBM-X - BCD source for data collection
10. PROBD-X - BCD source for data collection

Filenames 1-4 listed above are the ones described in the JAVS User's Guide. Files HSTAR/JAVSHS and JAVSXQ are used during JAVS processing; PRPOOL, JPROBESX and the binary files generated by executing the job stream MAKBINPX are used during Test Execution.

The purpose for installing the JAVS data collection routines from source is to allow compilation of these routines using the installation's standard version of the JOCIT JOVIAL compiler. Once the data collection routines are compiled (by executing the job control stream MAKBINPX), all JAVS users can utilize the object form of the routines, as described in the JAVS User's Guide.

A.2 NON-HIS INSTALLATIONS

For non-HIS facilities, the installation tape consists of BCD source code for the JAVS processor and data collection routines. Figure A.1 shows a top-down overview of the JAVS software system. Source code for COMPOOLs and executable text is compiled into binaries for the JAVS components and data collection group. The binary files are arranged into overlay links, as shown in Fig. A.2, to build an absolute file. Table A.1 identifies the overlay links by name and the functional keyword known by JAVS.

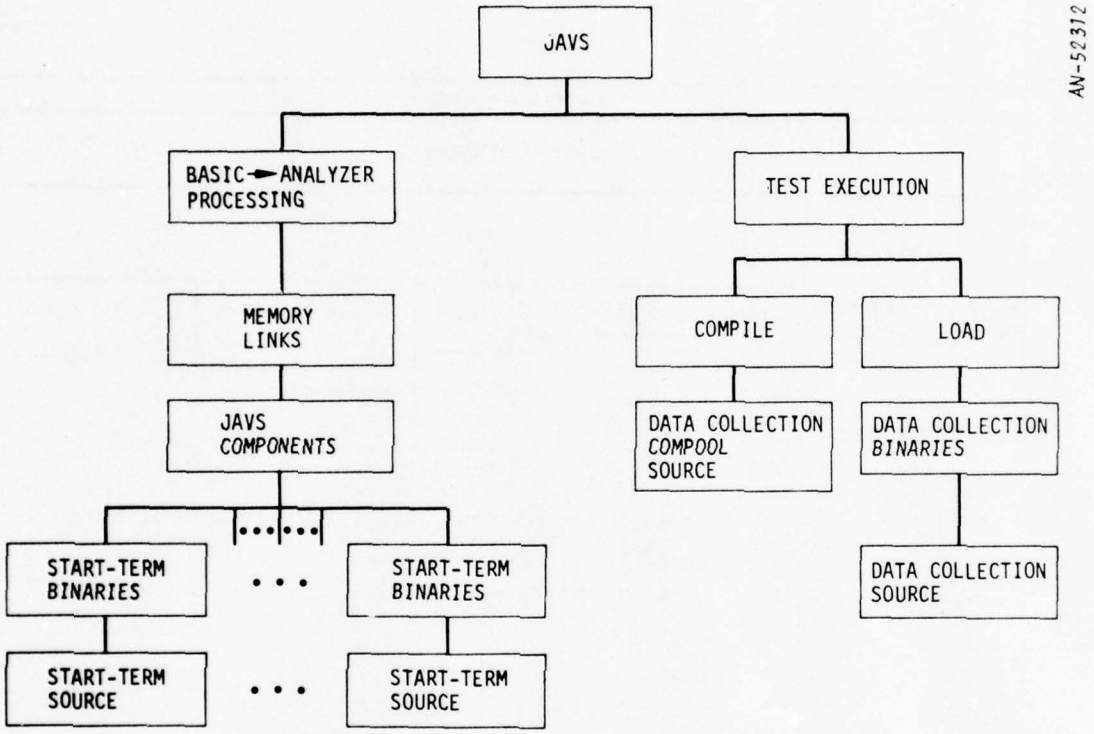
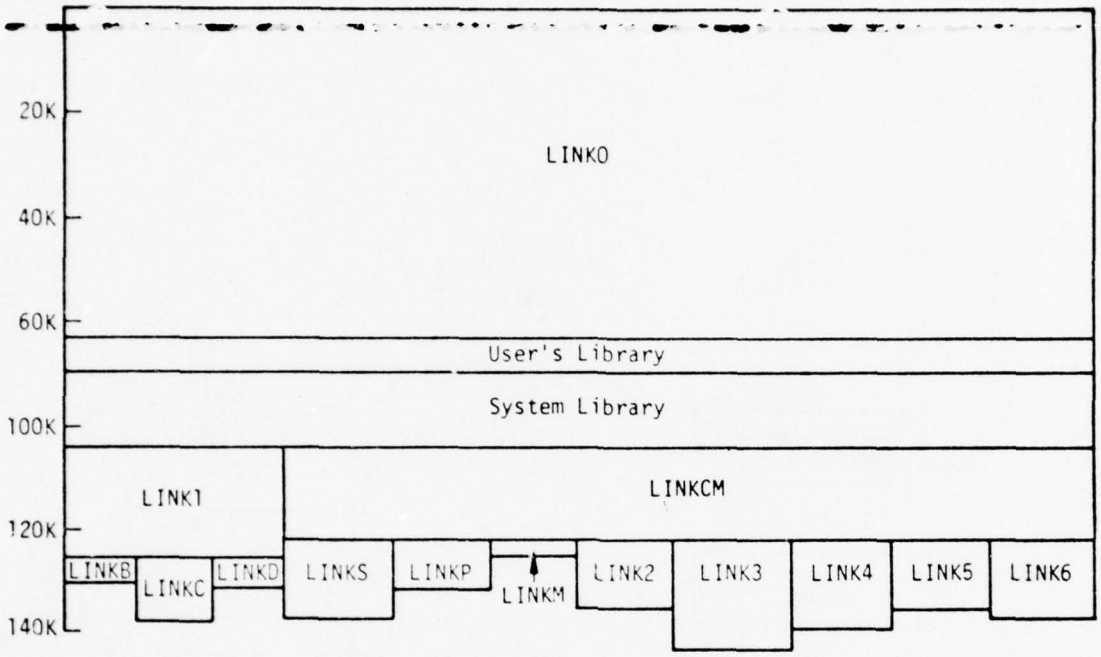


Figure A.1. Overview of JAVS Installation



K = 1000 octal

Load Module	CONTENTS
LINK0	Some utility COMMONs, JAVS-0, part of JAVS-1, -10, -11
LINK1	JAVS-2A, JAVS-2 COMMONs
LINKB	JAVS-2B, MAKTAB from JAVS-1
LINKC	JAVS-2C
LINKD	JAVS-2D
LINKCM	Some COMMONs, part of JAVS-1, -10, -11
LINKS	STSTOP from JAVS-1
LINKP	STPROP from JAVS-10
LINKM	JAVS-M
LINK2	JAVS-3
LINK3	JAVS-5
LINK4	JAVS-7
LINK5	JAVS-9
LINK6	JAVS-6

Figure A.2. JAVS Memory Layout on HIS 6180

TABLE A.1

JAVS COMPONENTS

IDENTIFIER	NAME	FUNCTIONAL KEYWORD
JAVS-0	Command & Control	
JAVS-1	Storage Manager	
JAVS-2	Primary Module Analysis	BASIC
JAVS-3	Secondary Module Analysis	STRUCTURAL
JAVS-4	Structural Analysis	STRUCTURAL
JAVS-5	Instrumentation	INSTRUMENT
JAVS-6	Data Collection and Reduction	ANALYZER
JAVS-7	Testcase Assistance	ASSIST
JAVS-8	Segment Analysis	ASSIST
JAVS-9	Program Modification Analysis	DEPENDENCE
JAVS-10	Data Base Services	
JAVS-11	Support Subroutines	
JAVS-M	Macro Command Processor	

The source tape contains the COMPOOLs and executable source code for all JAVS components including the data collection routines. The tape can be made to have an end-of-file mark following each START-TERM sequence or with a single end-of-file mark terminating the entire source (preferable for installations with CDC UPDATE or similar maintenance program). In either case, single- or multi-file source tape, the arrangement of source code is as follows:

TABLE A.2

SOURCE CODE ARRANGEMENT

Description	JAVS Component
COMPOOL	JAVS-0
3 START-TERMs	JAVS-0
1 START-TERM	JAVS-M (Uses JAVS-0 COMPOOL)
COMPOOL	JAVS-1
49 START-TERMs	JAVS-1
COMPOOL	JAVS-2
19 START-TERMs	JAVS-2
COMPOOL	JAVS-3, -4
1 START-TERM	JAVS-3, -4
COMPOOL	JAVS-5
1 START-TERM	JAVS-5
COMPOOL	JAVS-6
1 START-TERM	JAVS-6
COMPOOL	JAVS-7, -8
1 START-TERM	JAVS-7, -8
COMPOOL	JAVS-9
1 START-TERM	JAVS-9
COMPOOL	JAVS-10, -11
58 START-TERMs	JAVS-10, -11
COMPOOL	JAVS-10 Installation Dependent
4 START-TERMs	JAVS-10
1 FORTRAN routine	JAVS-10
COMPOOL	Data collection
4 START-TERMs	Data collection

Modules containing machine dependencies, in addition to those noted in Table A.2, are the JAVS-2 COMPOOL and ISCLAS, ITSDHL and ITSHOL in the JAVS-10 component. The machine dependencies in the JAVS-2 COMPOOL deal with the number of bits per character and the number of characters per word. Each declaration is commented. The JAVS-10 modules ISCLAS, ITSDHL and ITSHOL depend on the internal character representation. The JAVS-10 COMPOOL, START-TERMs (LSETUP, LBREAD, LBWRIT, LEND) and FORTRAN module listed in Table A.2 are random I/O routines.

The JAVS System Design and Implementation Manual[5] contains functional descriptions of each JAVS component and module. System routines, such as those providing date and time, which are directly invoked by JAVS are listed in Appendix C and in the module index of that document. The computer listing in the remaining pages of this appendix are included to show the suggested load sequence of the compiled JAVS software. Each binary file name is Bxxxxxx, where xxxxxx is the JAVSTEXT name supplied at the beginning of each START-TERM sequence on the source tape.

```
IDENT   BFCBGRC1,JAVS GRC ,555020030115,HSTAR/JAVSALL
USERID  BFCBGRC1$####
LOWLOAD
OPTION  FORTRAN,NOGO
LIBRARY Z*
SELECT  BFCBGRC2/BPOOL1
OBJECT                                                  C16,35304057BPOOL1 00
DKEND                                                                POOL1 04
SELECT  BFCBGRC2/BCMPL10
OBJECT                                                  C20,23005187BG10CPL00
DKEND                                                                G10CPL04
SELECT  BFCBGRC2/B10/BDSTEP0
OBJECT                                                  J17,90604087DSTEP000
DKEND                                                                DSTEP003
SELECT  BFCBGRC2/B10/BIHVALU
OBJECT                                                  J17,74904087IHVALU00
DKEND                                                                IHVALU06
SELECT  BFCBGRC2/B10/BISCLAS
OBJECT                                                  J17,75304087ISCLAS00
DKEND                                                                ISCLAS06
SELECT  BFCBGRC2/B10/BOUTBUF
OBJECT                                                  J17,87604087OUTBUF00
DKEND                                                                OUTBUF08
SELECT  BFCBGRC2/B1/BACTDAT
OBJECT                                                  J13,834102276ACTDAT00
DKEND                                                                ACTDAT09
SELECT  BFCBGRC2/B1/BACTDMP
OBJECT                                                  J13,839102276ACTDMP00
DKEND                                                                ACTDMP08
SELECT  BFCBGRC2/B1/BACTFRG
OBJECT                                                  J13,845102276ACTFRG00
DKEND                                                                ACTFRG11
SELECT  BFCBGRC2/B1/BACTICH
OBJECT                                                  J13,850102276ACTICH00
DKEND                                                                ACTICH04
SELECT  BFCBGRC2/B1/BACTMOD
OBJECT                                                  J13,857102276ACTMOD00
DKEND                                                                ACTMOD14
SELECT  BFCBGRC2/B1/BACTOLD
OBJECT                                                  J13,866102276ACTOLD00
DKEND                                                                ACTOLD08
SELECT  BFCBGRC2/B1/BCRTFRG
OBJECT                                                  J13,870102276CRTFRG00
DKEND                                                                CRTFRG05
SELECT  BFCBGRC2/B1/BDMPFRG
OBJECT                                                  J13,874102276DMPFRG00
DKEND                                                                DMPFRG11
SELECT  BFCBGRC2/B1/BDMPMOD
OBJECT                                                  J14,128102276DMPMOD00
DKEND                                                                DMPMOD09
SELECT  BFCBGRC2/B1/BGETBLK
OBJECT                                                  J13,884102276GETBLK00
DKEND                                                                GETBLK11
SELECT  BFCBGRC2/B1/BGETFRG
OBJECT                                                  J13,904102276GETFRG00
DKEND                                                                GETFRG14
```

```
SELECT  BFCBGRC2/B1/BGETLBT
OBJECT                                  J14.004102276GETLBT00
DKEND                                           GETLBT24
SELECT  BFCBGRC2/B1/BGETLST
OBJECT                                  J14.041102276GETLST00
DKEND                                           GETLST15
SELECT  BFCBGRC2/B1/BGETMOD
OBJECT                                  J14.064102276GETMOD00
DKEND                                           GETMOD12
SELECT  BFCBGRC2/B1/BGETTAB
OBJECT                                  J14.069102276GETTAB00
DKEND                                           GETTAB08
SELECT  BFCBGRC2/B1/BICHFRG
OBJECT                                  J14.085102276ICHFRG00
DKEND                                           ICHFRG17
SELECT  BFCBGRC2/B1/BIFNDEP
OBJECT                                  J14.095102276IFNDEP00
DKEND                                           IFNDEP16
SELECT  BFCBGRC2/B1/BIGTLIT
OBJECT                                  J14.104102276IGTLIT00
DKEND                                           IGTLIT04
SELECT  BFCBGRC2/B1/BIGTWRD
OBJECT                                  J14.109102276IGTWRD00
DKEND                                           IGTWRD13
SELECT  BFCBGRC2/B1/BIMAKEP
OBJECT                                  J14.114102276IMAKEP00
DKEND                                           IMAKEP04
SELECT  BFCBGRC2/B1/BISRTAB
OBJECT                                  J14.124102276ISRTAB00
DKEND                                           ISRTAB17
SELECT  BFCBGRC2/B1/BLBFREE
OBJECT                                  J14.135102276LBFREE00
DKEND                                           LBFREE06
SELECT  BFCBGRC2/B1/BLBMOVE
OBJECT                                  J14.148102276LBMOVE00
DKEND                                           LBMOVE06
SELECT  BFCBGRC2/B1/BLBTGET
OBJECT                                  J14.154102276LBTGET00
DKEND                                           LBTGET06
SELECT  BFCBGRC2/B1/BLBTPUT
OBJECT                                  J14.161102276LBTPUT00
DKEND                                           LBTPUT06
SELECT  BFCBGRC2/B1/BLBZERO
OBJECT                                  J14.166102276LBZERO00
DKEND                                           LBZERO06
SELECT  BFCBGRC2/B1/BLGTBLK
OBJECT                                  J14.171102276LGTBLK00
DKEND                                           LGTBLK09
SELECT  BFCBGRC2/B1/BLGTWRD
OBJECT                                  J14.176102276LGTWRD00
DKEND                                           LGTWRD08
SELECT  BFCBGRC2/B1/BLPTBLK
OBJECT                                  J14.181102276LPTBLK00
DKEND                                           LPTBLK09
SELECT  BFCBGRC2/B1/BLPTWRD
OBJECT                                  J14.186102276LPTWRD00
DKEND                                           LPTWRD08
SELECT  BFCBGRC2/B1/BMAKFRG
OBJECT                                  J14.193102276MAKFRG00
DKEND                                           MAKFRG10
SELECT  BFCBGRC2/B1/BMAKMOD
OBJECT                                  J14.197102276MAKMOD00
DKEND                                           MAKMOD14
SELECT  BFCBGRC2/B1/BMDBGET
OBJECT                                  J14.207102276MDBGET00
```

```
SELECT  BFCBGRC2/B1/BMDBPUT
OBJECT                                  J14.213102276MDBPUT00
DKEND                                           MDBPUT07
SELECT  BFCBGRC2/B1/BPUTBLK
OBJECT                                  J14.236102276PUTBLK00
DKEND                                           PUTBLK14
SELECT  BFCBGRC2/B1/BPUTLIT
OBJECT                                  J14.241102276PUTLIT00
DKEND                                           PUTLIT04
SELECT  BFCBGRC2/B1/BPUTLST
OBJECT                                  J14.247102276PUTLST00
DKEND                                           PUTLST27
SELECT  BFCBGRC2/B1/BPUTWRD
OBJECT                                  J14.254102276PUTWRD00
DKEND                                           PUTWRD16
SELECT  BFCBGRC2/B1/BUPDEPT
OBJECT                                  J14.261102276UPDEPT00
DKEND                                           UPDEPT16
SELECT  BFCBGRC2/B1/BWRAPUP
OBJECT                                  J14.268102276WRAPUP00
DKEND                                           WRAPUP04
SELECT  BFCBGRC2/B1/BWSGFRG
OBJECT                                  J14.277102276WSGFRG00
DKEND                                           WSGFRG07
SELECT  BFCBGRC2/B1/BWSGWRD
OBJECT                                  J14.281102276WSGWRD00
DKEND                                           WSGWRD07
SELECT  BFCBGRC2/B1/BWSPFRG
OBJECT                                  J14.286102276WSPFRG00
DKEND                                           WSPFRG07
SELECT  BFCBGRC2/B1/BWSPWRD
OBJECT                                  J14.335102276WSPWRD00
DKEND                                           WSPWRD07
SELECT  BFCBGRC2/B10/BMDBDAD
OBJECT                                  J17.672040877MDBDAD00
DKEND                                           MDBDAD05
SELECT  BFCBGRC2/B10/BMDBMOD
OBJECT                                  J17.699040877MDBMOD00
DKEND                                           MDBMOD05
SELECT  BFCBGRC2/B10/BMDBTXT
OBJECT                                  J17.703040877MDBTXT00
DKEND                                           MDBTXT05
SELECT  BFCBGRC2/B10/BERROR
OBJECT                                  J20.245051878ERROR 00
DKEND                                           ERROR 06
SELECT  BFCBGRC2/B10/BFATAL
OBJECT                                  J17.736040877FATAL 00
DKEND                                           FATAL 06
SELECT  BFCBGRC2/B10/BGTCARD
OBJECT                                  J17.740040877GTCARD00
DKEND                                           GTCARD06
SELECT  BFCBGRC2/B10/BITSFRG
OBJECT                                  J17.761040877ITSFRG00
DKEND                                           ITSFRG04
SELECT  BFCBGRC2/B10/BITSHOL
OBJECT                                  J17.766040877ITSHOL00
DKEND                                           ITSHOL07
SELECT  BFCBGRC2/B10/BLENTAB
OBJECT                                  J17.775040877LENTAB00
DKEND                                           LENTAB03
SELECT  BFCBGRC2/B10/BMOVEWD
OBJECT                                  J17.779040877MOVEWD00
DKEND                                           MOVEWD06
SELECT  BFCBGRC2/B10/BNFRGSZ
OBJECT                                  J17.784040877NFRGSZ00
DKEND                                           NFRGSZ03
```

```
SELECT  BFCBGRC2/B10/BNUMMDB
OBJECT                                          J17,845040877NUMMDB00
DKEND                                                       NUMMDB03
SELECT  BFCBGRC2/B10/BNUMSB
OBJECT                                          J17,860040877NUMSB 00
DKEND                                                       NUMSB 03
SELECT  BFCBGRC2/B10/BNUMSDB
OBJECT                                          J17,864040877NUMSDB00
DKEND                                                       NUMSDB03
SELECT  BFCBGRC2/B10/BNUMSLT
OBJECT                                          J17,868040877NUMSLT00
DKEND                                                       NUMSLT03
SELECT  BFCBGRC2/B10/BNUMSTB
OBJECT                                          J17,872040877NUMSTB00
DKEND                                                       NUMSTB03
SELECT  BFCBGRC2/B10/BPRDBG
OBJECT                                          J17,885040877PRDBG 00
DKEND                                                       PRDBG 07
SELECT  BFCBGRC2/B10/BSPRYWD
OBJECT                                          J17,901040877SPRYWD00
DKEND                                                       SPRYWD04
SELECT  BFCBGRC2/B10/BLBCMPL
OBJECT                                          C17,914040877LBCMPL00
DKEND                                                       LBCMPL03
SELECT  BFCBGRC2/B10/BLSETUP
OBJECT                                          J17,918040877LSETUP00
DKEND                                                       LSETUP15
SELECT  BFCBGRC2/B10/BLBREAD
OBJECT                                          J17,923040877LBREAD00
DKEND                                                       LBREAD08
SELECT  BFCBGRC2/B10/BLBWRIT
OBJECT                                          J17,927040877LBWRIT00
DKEND                                                       LBWRIT11
SELECT  BFCBGRC2/B10/BLBIO
OBJECT                                          Y17,928040877LBI00000
DKEND                                                       LBI00006
SELECT  BFCBGRC2/B10/BLEND
OBJECT                                          J17,933040877LEND  00
DKEND                                                       LEND  04
SELECT  BFCBGRC2/B10/BSHDATE
OBJECT                                          J17,869040877SHDATE00
DKEND                                                       SHDATE04
SELECT  BFCBGRC2/BCMPL0L
OBJECT                                          C15,534051678G0CMPL00
DKEND                                                       G0CMPL03
SELECT  BFCBGRC2/BSRCE0L
OBJECT                                          J15,539051678......00
DKEND                                                       ......92
SELECT  BFCBGRC2/BCCRACK
OBJECT                                          J15,542051678CCRACK00
DKEND                                                       CCRACK18
SELECT  BFCBGRC2/BGTLIN
OBJECT                                          J15,544051678GTLIN 00
DKEND                                                       GTLIN 11
LINK    LINK1
ENTRY   STEP1
SELECT  BFCBGRC2/NB2/BJ2A
OBJECT                                          J22,442051878J2A   00
DKEND                                                       J2A   15
SELECT  BFCBGRC2/NB2/BAZAP
OBJECT                                          J22,444051878AZAP  00
DKEND                                                       AZAP  03
SELECT  BFCBGRC2/NB2/BBZAP
OBJECT                                          J22,446051878BZAP  00
```

```
SELECT  BFC3GRC2/NB2/BCZAP
OBJECT                                          J22,44805187 8CZAP  00
DKEND                                                        CZAP  03
SELECT  BFCBGRC2/NB2/BDZAP
OBJECT                                          J22,45005187 8DZAP  00
DKEND                                                        DZAP  05
SELECT  BFCBGRC2/NB2/BEZAP
OBJECT                                          J22,45205187 8EZAP  00
DKEND                                                        EZAP  03
SELECT  BFCBGRC2/NB2/BIZAP
OBJECT                                          J22,45405187 8IZAP  00
DKEND                                                        IZAP  05
SELECT  BFCBGRC2/NB2/BJAVZAP
OBJECT                                          J22,45605187 8JAVZAP00
DKEND                                                        JAVZAP05
SELECT  BFCBGRC2/NB2/BJZAP
OBJECT                                          J22,45805187 8JZAP  00
DKEND                                                        JZAP  05
SELECT  BFCBGRC2/NB2/BLZAP
OBJECT                                          J22,46005187 8LZAP  00
DKEND                                                        LZAP  05
SELECT  BFCBGRC2/NB2/BMODZAP
OBJECT                                          J22,46205187 8MODZAP00
DKEND                                                        MODZAP10
SELECT  BFCBGRC2/NB2/BSTZAP
OBJECT                                          J22,46405187 8STZAP 00
DKEND                                                        STZAP 06
SELECT  BFCBGRC2/NB2/BSZAP
OBJECT                                          J22,46605187 8SZAP  00
DKEND                                                        SZAP  05
SELECT  BFCBGRC2/NB2/BTZAP
OBJECT                                          J22,46805187 8TZAP  00
DKEND                                                        TZAP  03
SELECT  BFCBGRC2/NB2/BMGET
OBJECT                                          J22,47005187 8MGET  00
DKEND                                                        MGET  04
SELECT  BFCBGRC2/NB2/BMPUT
OBJECT                                          J22,47205187 8MPUT  00
DKEND                                                        MPUT  04
LINK    LINKB
ENTRY   FINIT
SELECT  BFCBGRC2/NB2/BJ2B
OBJECT                                          J22,47905187 8J2B   00
DKEND                                                        J2B   85
SELECT  BFCBGRC2/B1/BMAKTAB
OBJECT                                          J14,20210227 6MAKTAB00
DKEND                                                        MAKTAB08
LINK    LINKC,LINKB
ENTRY   FONE
SELECT  BFCBGRC2/NB2/BJ2C
OBJECT                                          J22,49405187 8J2C   00
DKEND                                                        J2C   50
LINK    LINKD,LINKC
ENTRY   FTWO
SELECT  BFCBGRC2/NB2/BJ2D
OBJECT                                          J22,50305187 8J2D   00
DKEND                                                        J2D   89
LINK    LINKCM,LINK1
ENTRY   GETWRD
SELECT  BFCBGRC2/NBPOOL3
OBJECT                                          C16,68605107 8POOL3 00
DKEND                                                        POOL3 07
SELECT  BFCBGRC2/NBPOOL5
OBJECT                                          C17,26704057 8POOL5 00
DKEND                                                        POOL5 08
SELECT  BFCBGRC2/NBPOOL7
```

```
OBJECT                                                  C15,191051678POOL7 00
DKEND                                                                 POOL7 03
SELECT  BFCBGRC2/BPOOL9
OBJECT                                                  C13,453060377POOL9 00
DKEND                                                                 POOL9 03
SELECT  BFCBGRC2/BPOOL6
OBJECT                                                  C19,963101376POOL6 00
DKEND                                                                 POOL6 03
SELECT  BFCBGRC2/B1/BGETWRD
OBJECT                                                  J14,074102276GETWRD00
DKEND                                                                 GETWRD04
SELECT  BFCBGRC2/B1/BITABEP
OBJECT                                                  J14,119102276ITABEP00
DKEND                                                                 ITABEP04
SELECT  BFCBGRC2/B1/BLBMERG
OBJECT                                                  J14,330102476LBMERG00
DKEND                                                                 LBMERG15
SELECT  BFCBGRC2/B1/BMAKTAB
OBJECT                                                  J14,202102276MAKTAB00
DKEND                                                                 MAKTAB08
SELECT  BFCBGRC2/B1/BMKVTAB
OBJECT                                                  J14,220102276MKVTAB00
DKEND                                                                 MKVTAB05
SELECT  BFCBGRC2/B10/BPRDVAL
OBJECT                                                  J18,003040877PRDVAL00
DKEND                                                                 PRDVAL12
SELECT  BFCBGRC2/B10/BPREMTH
OBJECT                                                  J17,982040877PREMTH00
DKEND                                                                 PREMTH07
SELECT  BFCBGRC2/B10/BPRSKEL
OBJECT                                                  J17,998040877PRSKEL00
DKEND                                                                 PRSKEL26
SELECT  BFCBGRC2/B10/BPRTMTH
OBJECT                                                  J20,232051878PRTMTH00
DKEND                                                                 PRTMTH18
SELECT  BFCBGRC2/B10/BPSTMTH
OBJECT                                                  J17,992040877PSTMTH00
DKEND                                                                 PSTMTH22
SELECT  BFCBGRC2/B10/BGETVAR
OBJECT                                                  J17,937040877GETVAR00
DKEND                                                                 GETVAR06
SELECT  BFCBGRC2/B10/BMTHDDP
OBJECT                                                  J17,942040877MTHDDP00
DKEND                                                                 MTHDDP09
SELECT  BFCBGRC2/B10/BSORTAB
OBJECT                                                  J17,947040877SORTAB00
DKEND                                                                 SORTAB08
SELECT  BFCBGRC2/B10/BIGTBIT
OBJECT                                                  J17,953040877IGTBIT00
DKEND                                                                 IGTBIT05
SELECT  BFCBGRC2/B10/BPUTBIT
OBJECT                                                  J17,961040877PUTBIT00
DKEND                                                                 PUTBIT06
SELECT  BFCBGRC2/B10/BISELEM
OBJECT                                                  J17,957040877ISELEM00
DKEND                                                                 ISELEM04
SELECT  BFCBGRC2/B10/BFNDLB
OBJECT                                                  J17,966040877FNDLB 00
DKEND                                                                 FNDLB 08
SELECT  BFCBGRC2/B10/BNXTEX
OBJECT                                                  J17,972040877NXTEX 00
DKEND                                                                 NXTEX 19
SELECT  BFCBGRC2/B10/BSCANSB
OBJECT                                                  J17,977040877SCANSB00
```

```
SELECT  BFCBGRC2/B10/BPRMBUF
OBJECT                                          J17.707040877PRMBUF00
DKEND                                                   PRMBUF07
SELECT  BFCBGRC2/B10/BPRMODL
OBJECT                                          J17.712040877PRMODL00
DKEND                                                   PRMODL16
SELECT  BFCBGRC2/B10/BPRSTMT
OBJECT                                          J17.717040877PRSTMT00
DKEND                                                   PRSTMT14
SELECT  BFCBGRC2/B10/BANALEX
OBJECT                                          J17.910040877ANALEX00
DKEND                                                   ANALEX15
SELECT  BFCBGRC2/B10/BBALPAR
OBJECT                                          J17.722040877BALPAR00
DKEND                                                   BALPAR06
SELECT  BFCBGRC2/B10/BBLDLIN
OBJECT                                          J20.23505187 8BLDLIN00
DKEND                                                   BLDLIN20
SELECT  BFCBGRC2/B10/BIGTSTB
OBJECT                                          J17.745040877IGTSTB00
DKEND                                                   IGTSTB06
SELECT  BFCBGRC2/B10/BITSDHL
OBJECT                                          J17.757040877ITSDHL00
DKEND                                                   ITSDHL05
SELECT  BFCBGRC2/B10/BJDENT
OBJECT                                          J17.771040877JDENT 00
DKEND                                                   JDENT 14
SELECT  BFCBGRC2/B10/BNUMDMT
OBJECT                                          J17.812040877NUMDMT00
DKEND                                                   NUMDMT03
SELECT  BFCBGRC2/B10/BNUMDDP
OBJECT                                          J17.788040877NUMDDP00
DKEND                                                   NUMDDP03
SELECT  BFCBGRC2/B10/BNUMDS
OBJECT                                          J17.825040877NUMDS 00
DKEND                                                   NUMDS 03
SELECT  BFCBGRC2/B10/BNUMEPT
OBJECT                                          J17.829040877NUMEPT00
DKEND                                                   NUMEPT03
SELECT  BFCBGRC2/B10/BNUMFDT
OBJECT                                          J17.833040877NUMFDT00
DKEND                                                   NUMFDT03
SELECT  BFCBGRC2/B10/BNUMMLT
OBJECT                                          J17.849040877NUMMLT00
DKEND                                                   NUMMLT03
SELECT  BFCBGRC2/B10/BNUMODS
OBJECT                                          J17.855040877NUMODS00
DKEND                                                   NUMODS03
SELECT  BFCBGRC2/B10/BNUMPRB
OBJECT                                          J17.840040877NUMPRB00
DKEND                                                   NUMPRB03
SELECT  BFCBGRC2/B10/BPCHBUF
OBJECT                                          J17.861040877PCHBUF00
DKEND                                                   PCHBUF04
SELECT  BFCBGRC2/B10/BSLTSTB
OBJECT                                          J17.893040877SLTSTB00
DKEND                                                   SLTSTB07
SELECT  BFCBGRC2/B10/BSORT
OBJECT                                          J17.898040877SORT  00
DKEND                                                   SORT  08
LINK    LINKS
ENTRY   STSTOP
SELECT  BFCBGRC2/B1/BSTSTOP
OBJECT                                          J16.395040578STSTOP00
DKEND                                                   STSTOP41
LINK    LINKP,LINKS
```

```
ENTRY    STPROP
SELECT   BFCBGRC2/B10/BSTPROP
OBJECT                                          J20,24305187 8STPROP00
DKEND                                                        STPROP44
LINK     LINKM,LINKP
ENTRY    MACROC
SELECT   BFCBGRC2/BMACRO
OBJECT                                          J16,64804057 8MACROC00
DKEND                                                        MACROC59
LINK     LINK2,LINKM
ENTRY    STEP2
SELECT   BFCBGRC2/NBINRY3
OBJECT                                          J16,73605107 8SORCE300
DKEND                                                        SORCE514
LINK     LINK3,LINK2
ENTRY    STEP3
SELECT   BFCBGRC2/NBINRY5
OBJECT                                          J17,29704057 8SORCE500
DKEND                                                        SORCE880
LINK     LINK4,LINK3
ENTRY    STEP4
SELECT   BFCBGRC2/NBINRY7
OBJECT                                          J15,20505167 8SORCE700
DKEND                                                        SORCE937
LINK     LINK5,LINK4
ENTRY    STEP5
SELECT   BFCBGRC2/BINARY9
OBJECT                                          J13,48206037 7SORCE900
DKEND                                                        SORCE086
LINK     LINK6,LINK5
ENTRY    STEP6
SELECT   BFCBGRC2/BSORCE6
OBJECT                                          J19,99710137 6SORCE600
DKEND                                                        SORCE842
EXECUTE
PRMFL    Z*,R,S,BFCBGRC1/JOVLIB
LIMITS   10,55K,-5K
PRMFL    H*,R/W,R,BFCBGRC1/HSIAR/JAVSHS
FILE     10,C1R,L
FILE     01,X1R,R
FILE     02,X2R,R
FILE     03,X3R,R
FILE     04,X4R,L
FILE     07,X7R,L
FILE     08,X8R,L
FILE     09,X9R,L
ENDJOB
```

APPENDIX B

UPDATES TO USER'S GUIDE

Appendix B consists entirely of updated pages to the November 1976 Edition of the JAVS Technical Report: Vol. 1, "User's Guide," available as RADC-TR-77-126, Vol. I. Replacement of the modified pages in this appendix will make the November 1976 guide identical with the JAVS Technical Report: Vol. 1, "User's Guide," General Research Corporation CR-1-722/1, June 1978.

ABSTRACT

The JOVIAL Automated Verification System (JAVS) is a control-path testing tool which analyzes source programs written in the J3 dialect of the JOVIAL language. From the user's viewpoint, JAVS consists of a sequence of processing steps which (1) builds a database containing syntactical and structural information about his JOVIAL source text, (2) prepares documentation reports describing inter- and intra-module characteristics of the source code, (3) measures control-path coverage during program execution, and (4) assists in pinpointing untested source code and preparing additional test data.

The purpose of this document is to introduce the tester to JAVS and to the process of software testing supported by JAVS. The information provided in this guide on JAVS usage is intentionally limited to the beginning user. The appendixes provide the information necessary for operating JAVS at RADC and can be referenced by the sophisticated as well as the beginning user. The information presented on the testing methodology which JAVS supports is applicable to both the beginning and sophisticated user of JAVS.

PREFACE

The purpose of this guide is to introduce the user into the realm of automated testing. Software testing supported by an automated verification system requires knowledge of two inseparable factors: the verification tool and the testing methodology which the tool supports. The information concerning the usage of JAVS is intentionally limited to the beginning user. The only prerequisite information is a knowledge of the JOVIAL language. The casual user should have good success in analyzing the behavior of his programs using the description of JAVS capabilities and a few commands set forth in this guide. All job control and file information is presented in appendixes, along with estimates of processing time and core requirements.

The testing methodology which JAVS supports is described in this guide because of its importance to JAVS users at all levels of expertise. Although there is no single general methodology which applies to all testing situations, there are a number of important issues that even the beginning verification tool user should recognize in order to make the testing experience successful. Section 10 of this guide focuses on software preparation for JAVS-supported testing, testing goals, resources required, and testing strategy.

In the series of JAVS reports, this guide should be read first. The information presented should enable the tester to become a new user of JAVS at RADC. Once the user has experienced some of the capabilities that JAVS offers, the JAVS Reference Manual[1] should be used to supply the complete details of JAVS features and command language.

For more comprehensive treatment of software testing methodology not restricted solely to JAVS-supported testing, the reader is referred to the Methodology Report.[2] This report describes experiences with using current Automated Verification Systems, approaches to software quality, and advanced AVS capabilities.

SPECIAL INSTRUCTIONS:

This User's Guide supersedes the JAVS User's Guide, dated November 1976. Change bars in the page margins indicate additions and changes to the 1976 guide; asterisks denote deletions.

LIST OF JAVS REPORTS

- Revised Methodology for Comprehensive Software Testing. This report describes the methodology which underlies and is supported by JAVS. The methodology is tailored to be largely independent of implementation and language. The discussion in the text is intended to be intuitive and demonstrative. Some of the methodology is based upon the experience of using JAVS to test a large information management system. A long-term growth path for automated verification systems that supports the methodology is described.

- JAVS Technical Report: Vol. 1, User's Guide. This report is an introduction to using JAVS in the testing process. Its primary purpose is to acquaint the user with the innate potential of JAVS to aid in the program testing process so that an efficient approach to program verification can be undertaken. Only the basic principles by which JAVS provides this assistance are discussed. These give the user a level of understanding necessary to see the utility of the system. The material on JAVS processing in the report is presented in the order normally followed by the beginning JAVS user. Adequate testing can be achieved using JAVS macro commands and the job streams presented in this guide. The Appendices include a summary of all JAVS commands and a description of JAVS operation at RADC with both sample command sets and sample job control statements.

- JAVS Technical Report: Vol. 2, Reference Manual. This report describes in detail JAVS processing and each of the JAVS commands. The Reference Manual is intended to be used along with the User's Guide which contains the machine-dependent information such as job control cards and file allocation. Throughout the Reference Manual, modules from a sample JOVIAL program are used in the examples. Each JAVS command is explained in detail, and a sample of each report produced by JAVS is included with the appropriate command. The report is organized into two major parts: the first four sections describing the JAVS system and the fifth section containing the description of each JAVS command in alphabetical order.

  The Appendices include a complete listing of all error messages directly produced by JAVS processing.

- JAVS Computer Program Documentation: Vol. 1, System Design and Implementation. This report contains a description of JAVS software design, the organization and contents of the JAVS data base, and a description of the software for each JAVS component: its function, each of the modules in the component, and the global data structures used by the component. The report is intended primarily as an informal reference for use in JAVS software maintenance as a companion to the Software Analysis reports described below. Included in the appendices are the templates for probe code inserted by instrumentation processing for both structural and directive instrumentation and an alphabetical list of all modules in the system (including system routines) with the formal parameters and data type of each parameter.

- JAVS Computer Program Documentation: Vol. 2, Software Analysis. This volume is a collection of computer output produced by JAVS standard processing steps. The source for each component of the JAVS software has been analyzed

to produce enhanced source listings of JAVS with indentation and control structure identification, inter-module depencence, all module invocations with formal and actual parameters, module control structure, a cross reference of symbol useage, tree report for each leading module, and report showing size of each component. It is intended to be used with the System Design and Implementation Manual for JAVS software maintenance. The Software Analysis reports, on file at RADC, are an excellent example of the use of JAVS for computer software documentation.

- JAVS Final Report. The final report for the project describes the design, implementation and testing of the JAVS syntax analyzer. Background information regarding all JAVS contracts is provided as well as procedures for installing the complete JAVS software package. This report contains, as appendices, the June 1978 updated pages for the User's Guide and Reference Manual published as RADC-TR-77-126, Vols. I and II, April 1977.

2 USING JAVS

The process of program verification is best described by example. One purpose of this User's Guide is to present an overview of JAVS capabilities through example programs processed by the JAVS execution steps. It is important to note that while there are six processing steps a given validation effort may require use of only a few of these. The selection of appropriate processes is largely a user decision, based upon his requirement for the information that the various steps provide. As each step is described, through example, the user will gain insight into its utility for his particular needs. In order to develop a basic understanding of the processing sequences to be utilized in the examples, Fig. 2.1 illustrates the potential JAVS processing flows in terms of step interdependencies.

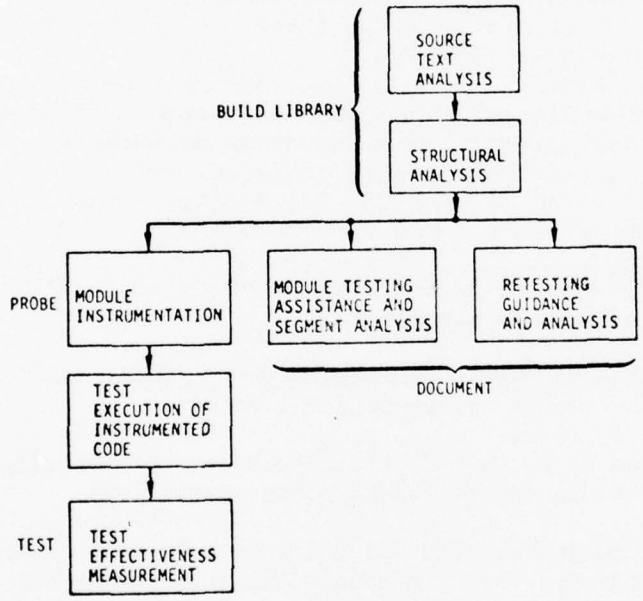


Figure 2.1. JAVS Processing Sequences

The user must provide three major types of input to JAVS: (1) the source code to be tested, (2) a set of commands to direct JAVS processing, and (3) test data for program execution. Section 2.2 describes the preparation of the source code for input to JAVS. Section 2.3 describes the rules for inputting commands.

2.1 TYPICAL PROCESSING SEQUENCE

This guide is organized to lead the user through the following sequence of steps:

1. Build a data base library containing source text and structural analyses (Sec. 3).

2. Document the source text (Sec. 4).

3. Instrument the modules (Sec. 5).

4. Execute the program (Sec. 6).

5. Measure the test's effectiveness (Sec. 7).

6. Retest the program (Sec. 8).

These steps provide the primary assistance needed to generate test cases and measure the extent of program testing coverage as each test case is input to the system.

2.2 PRELIMINARY STEPS

Before the source text to be verified is submitted to JAVS, the user should take certain preliminary steps:

1. The source text should be compiled by a JOVIAL compiler to confirm that it is free of any syntactical errors.

2. The program to be tested should have been previously executed with test data necessary to ensure proper execution.

3. JAVS text identification directives should be inserted in the source if there is more than one START-TERM sequence in the program.

4. JAVS computation directives should be inserted in the source if the performance testing capability is utilized.

Both types of JAVS directives (a special form of JOVIAL comment) are described in detail in the JAVS Reference Manual[1]. The JAVS text identification directive is used to assign a unique name to a JOVIAL START-TERM sequence (program, subprogram, or COMPOOL). If no text directive is assigned to a START-TERM sequence, a text name is assigned as a default. The computation directives are used to make assertions about the behavior of the program and to verify the value of specified variables without altering the logic of the program. These directives can make valuable contributions in debugging and boundary conditions testing. It is suggested that the user acquire some familiarity with JAVS before utilizing the computation directives. The Reference Manual (Sec. 1.5) describes their capability and utilization.

The following sections describe the recommended sequence of step executions to be utilized by the beginning user. Although JAVS is capable of processing very large JOVIAL programs, we recommend that the tester select a modest program (several hundred JOVIAL statements or less) to use in his first experience with JAVS processing.

2.3 COMMAND STRUCTURE

The user directs JAVS processing by a set of commands. There are four "macro" commands which can be used with the JAVS 3.0 overlay program, in addition to a variety of standard commands. Each macro command expands into a set of commonly used standard JAVS commands. While both types of commands can be used together, the user is advised to be aware of the expansion of the macros before combining commands. Table 2.1 shows the relationship between macro commands, standard commands, and the processing tasks. Sections 3-8 describe each task, as well as the appropriate commands to use, and the process of executing the test program is described in Sec. 6.

All commands are input one per card. Blanks are ignored, so the commands are free-form. The card scan ends with a period or with the end of a card. If a command requires more than one card, a comma must appear at the last non-blank character of each card preceeding the continuation card. Up to three continuation cards may be used. Each command consists of a sequence of terms separated by a comma or an equals sign.

TABLE 2.1

RELATIONSHIP BETWEEN COMMANDS AND TASKS

Macro Command Keyword	Standard Command Keyword	Task
BUILD LIBRARY	BASIC	Syntax analysis
	STRUCTURAL	Structural analysis
PROBE	INSTRUMENT	Structural and computation instrumentation
DOCUMENT	ASSIST	Module and inter-module reports
	PRINT	
	DEPENDENCE	
TEST	ANALYZER	Post-test coverage and trace analysis

3 PRIMARY ANALYSIS

Prior to instrumentation, documentation, testing, or retesting, a set of primary analyses must be performed. Syntax analysis is performed on the JOVIAL source program, transforming it into a format appropriate for storage on a random-access data base (library file). Using the information on the data base, structural analysis is performed on the executable modules, updating the tables in the data base library. Structural analysis includes building a directed program graph which is the basis for instrumentation and testing analyses. The subject matter of this section, primary analysis, is shown in the context of the testing process in Fig. 3.1.

3.1 TASKS

Syntax analysis consists of breaking-down each START-TERM sequence of the JOVIAL source text into invokable modules. A data base library is created containing internal tables representative of program text, statement descriptions and symbol classification.

Structural analysis adds to the data base library a description of program structure in terms of decision-to-decision paths. These paths represent a unique and systematic ordering of all decision outways. Figures 3.2 and 3.3 illustrate the concept of DD-paths. A DD-path consists of all the executable statements from a conditional statement to the next conditional statement. Figure 3.2 shows the statement membership for each DD-path in module EXAMPL. This module contains 12 DD-paths. Below each DD-path number (listed across the page) is the order in which the statements are placed on each DD-path. For example, DD-path 2 consists of statements 15, 16, 29, 30, and 31 in that order.

3.2 PRIMARY ANALYSIS INPUT

JAVS requires two input files for syntax analysis: the JOVIAL source program in BCD mode on file READER (09) and the JAVS commands in BCD mode on file COMMAC (05). If the source program contains more than one START-TERM sequence, or if the source text is a COMPOOL or requires a COMPOOL to compile, the user must insert a JAVS text identification directive as the first statement. This statement is described in Sec. 1.4 of the Reference Manual and is shown in Figs. 3.2 and 3.4.

Input for structural analysis are the JAVS commands and the data base library created during the syntax analysis.

3.3 COMMANDS

Primary analysis can be accomplished by the single JAVS command:

```
BUILD LIBRARY [=<name>].
```

This command expands into the set of standard commands on page 3-5.

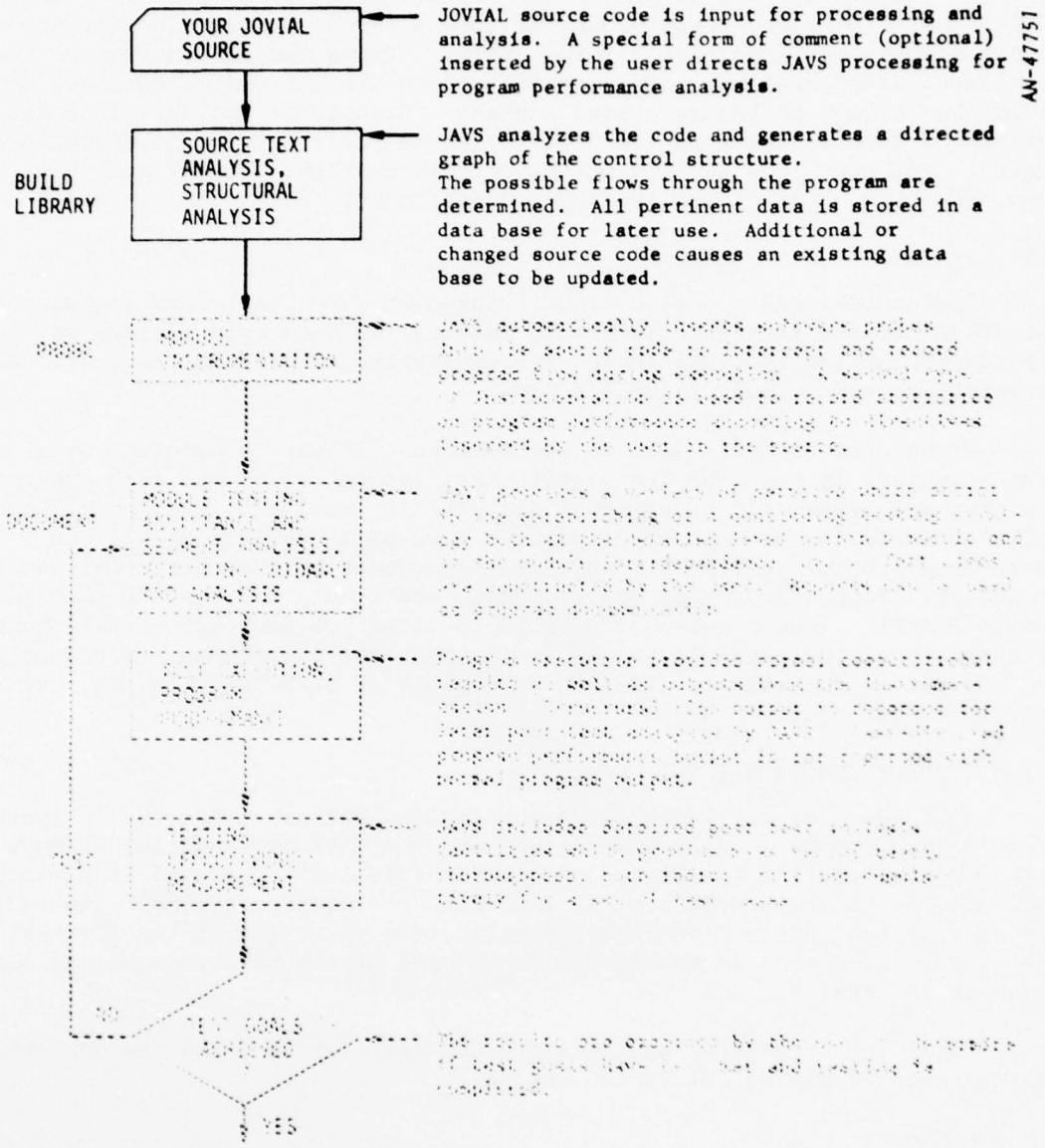


Figure 3.1. The Role of Primary Analysis in the Testing Process

```
".JAVSTEXT EXAMPL COMPUTE (COMPOL)"
  START
  PROC EXAMPL   (AA=BB)$
  ITEM AA F $
  ITEM BB F$
  ARRAY CC 2 2 F$
  BEGIN  BEGIN  1.0 1.0 END
         BEGIN  1.0 1.0 END  END
  BEGIN
  MONITOR BB, CC$
  IFEITH AA LS 0.0$
  BB = -AA$
  ORIF AA EQ 0.0$
  BEGIN
  BB = 0.0$
  FOR I=0,1,1$
  BEGIN
  FOR J=0,1,1$
  CC($I,J$) = 0.0$
  END
  RETURN$
  END
  ORIF 1$
  BB = AA$
  END
  FOR K=0,1,1$
  CC($K,0$) = BB/2.0$
  END
  TERM$

MODULES DEFINED IN TEXT
   1      EXAMPL  (EXAMPL  )
```

Figure 3.4. BASIC Output

```
CREATE LIBARY=<name>.  (default name is TEST)
START.
BASIC.
FOR LIBRARY.
STRUCTURAL.
END FOR.
END.
```

The actions taken by the macro command (or equivalent set of standard commands) is to initialize the JAVS system with a library whose name is <name> (or TEST if none is specified), process syntax analysis (BASIC) removing JOVIAL comment statements, and perform structural analysis for all modules on the newly created library.

There are several BASIC processing options, all described in the Reference Manual. If the user wishes to exercise any of the options, to delete

the JOVIAL comments in the source text from the library, or to perform structural analysis on a subset of the modules, he cannot use the BUILD LIBRARY macro command; instead, the desired sequence of BASIC commands must be supplied. Section 5 of the Reference Manual contains sample command sets for each command description.

3.4 PRIMARY ANALYSIS OUTPUT

The main output is a data base library file containing the source text transformed into invokable modules and tables for other functional processing and reports. Printed output consists of the card image listing of the JOVIAL source code (this can be turned off with a BASIC option) along with JAVS error messages, if any, and a few descriptive lines for each module stating the number of DD-paths generated. If any syntax errors are printed adjacent to the offending source text line, they should be scrutinized. A complete list of JAVS errors is in Appendix B of the Reference Manual. Some errors will require source code changes before further processing, and some errors are syntactical warnings.

Figure 3.4 shows the syntax analysis output and Fig 3.5 shows structural analysis output for module EXAMPL.

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

MODULE  EXAMPL  > OF JAVSTEXT <EXAMPL  >.
     MODULE DEPENDENCE TABLE CONSTRUCTED.
     STATEMENT DESCRIPTOR BLOCKS UPDATED.
     DD-PATH TABLE CONTAINS   12 ENTRIES.
```

Figure 3.5. STRUCTURAL Output

9 COMMAND SUMMARY

The subset of JAVS commands which will enable the first-time user to process his source code are summarized in Table 9.1. The command streams in Table 9.2 show the natural order of processing and a typical selection of commands. Instrumentation, activity 2 in Table 9.2, is shown as a separate activity so that the user can obtain the statement numbers needed by the PROBI commands. Instrumentation can be performed as part of the first activity following the BUILD LIBRARY operation if the user wishes to manually insert (through a text editor) the necessary invocations to the PROBI data collection routine.

The rules for JAVS macro command usage and the default option values are described in Appendix B. For a complete description of all JAVS commands and sample output each command produces, see the Reference Manual.

TABLE 9.1

COMMAND SUMMARY

	TASK	COMMAND	OPTIONS
(1)	Perform syntax and structural analyses create data base library	BUILD LIBRARY	=<library name>.
(2)	Generate reports for documentation	DOCUMENT	,JAVSTEXT = <text name>, MODULE = <name-1>,..., <name-n>.
(3)	Insert test case initiation	PROBE,STARTTEST = <module name>, <text name>, <statement no.>	,<test case name>, <tracing level>.
(4)	Insert test termination	PROBE,STOPTEST = <module name>, <text name>, <statement no.>.	
(5)	Instrumentation	PROBE,JAVSTEXT = <text name>	,MODULE = <name-1>,..., <name-n>.
(6)	Post-test analysis	TEST	,MODULE = <name-1>,..., <name-n>.
(7)	Retesting	ASSIST,REACHING SET, <target>	,<initial>,ITERATIVE, PICTURE

TABLE 9.2

SAMPLE COMMAND SETS

ACTIVITY	COMMANDS
1	BUILD LIBRARY. DOCUMENT.
2	OLD LIBRARY = TEST. START. PROBE,STARTTEST = <module name>, <JAVSTEXT name>,<statement no.>. . . . PROBE,STOPTEST = <module name>, <JAVSTEXT name>,<statement no.>. PROBE,JAVSTEXT = <JAVSTEXT name>.
3	Perform Test Execution
4	OLD LIBRARY = TEST. START. ANALYZER,MODLST. ANALYZER,DDPATHS. TEST.

APPENDIX A

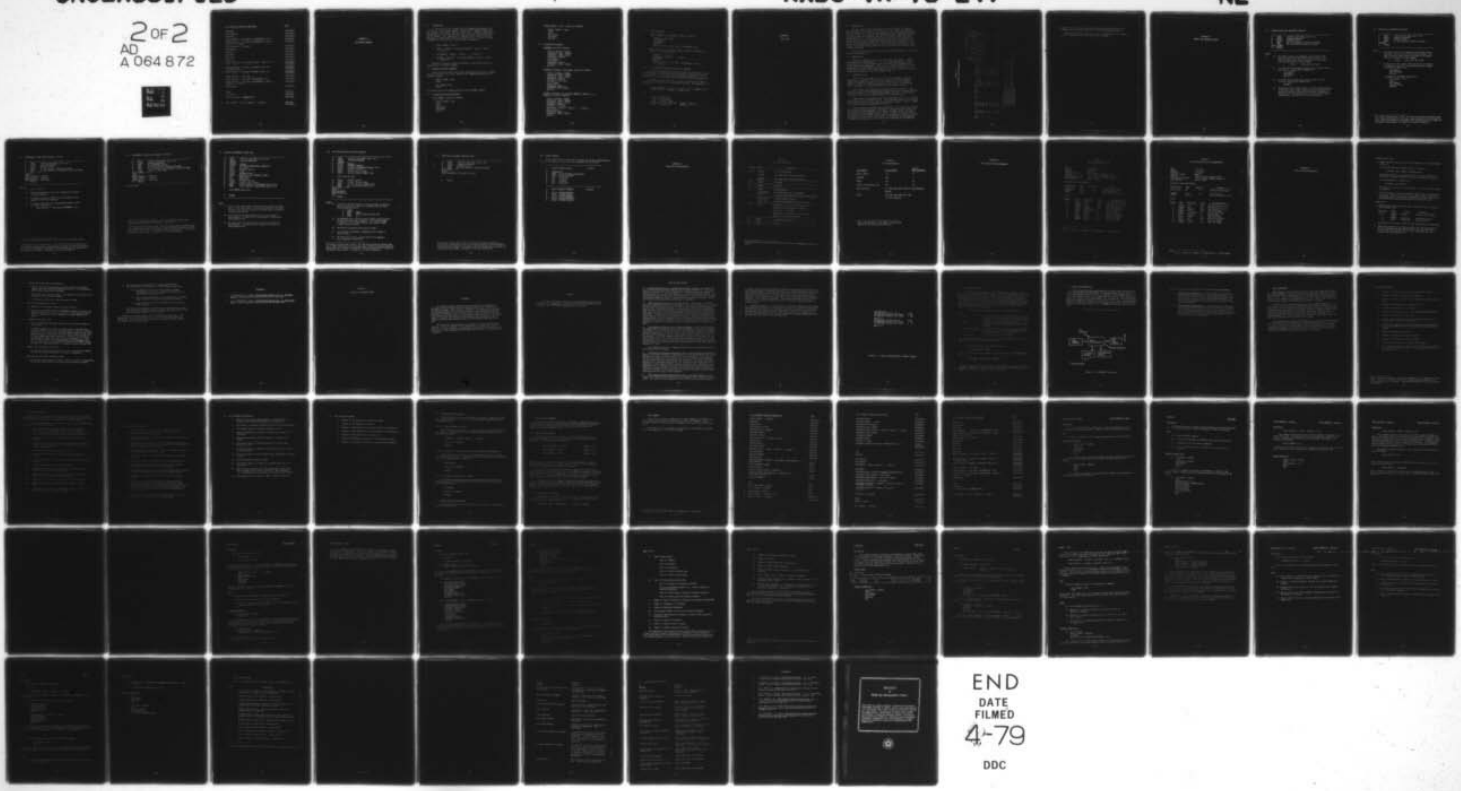
JAVS COMMAND SUMMARY

JAVS COMMANDS (DEFAULTS UNDERLINED)	STEP
ALTER LIBRARY = <libname>.	(Universal)
ANALYZER.	ANALYZER
ANALYZER,ALL.	ANALYZER
ANALYZER,ALL MODULES.	ANALYZER
ANALYZER,CASES = <number>.	ANALYZER
ANALYZER,DDPATHS.	ANALYZER
ANALYZER,DDPTRACE.	ANALYZER
ANALYZER,FACTOR = <percent-increase>.	ANALYZER
ANALYZER,HIT.	ANALYZER
ANALYZER,MODLST.	ANALYZER
ANALYZER,MODTRACE.	ANALYZER
ANALYZER,MODULE = <name-1>,<name-2>,...,<name-n>.	ANALYZER
ANALYZER,NOTHIT.	ANALYZER
ANALYZER,SUMMARY.	ANALYZER
ANALYZER,TIME.	ANALYZER
ASSIST,CROSSREF,JAVSTEXT = <text-name-1>,<text-name-2>,...,<text-name-n>.	ASSIST
ASSIST,CROSSREF,LIBRARY.	ASSIST
ASSIST,PICTURE.	ASSIST
ASSIST,PICTURE{,CONTROL}{,NOSWITCH}.	ASSIST
ASSIST,REACHING SET,<number-to>{,<number-from>}{,PICTURE{,ITERATIVE}}.	ASSIST
ASSIST,STATEMENTS.	ASSIST
BASIC.	BASIC
BASIC,CARD IMAGES = ON/OFF.	BASIC
BASIC,COMMENTS = ON/OFF.	BASIC
BASIC,DEFINES = ON/OFF.	BASIC
BUILD LIBRARY {= <library name>}.	BASIC, STRUCTURAL
CREATE LIBRARY = <libname>.	(Universal)

JAVS COMMANDS (DEFAULTS UNDERLINED)	STEP
DEPENDENCE,BANDS.	DEPENDENCE
DEPENDENCE,BANDS = <number>.	DEPENDENCE
DEPENDENCE,GROUP,AUXLIB.	DEPENDENCE
DEPENDENCE,GROUP,LIBRARY.	DEPENDENCE
DEPENDENCE,GROUP,MODULES = <name-1>,<name-2>,...,<name-n>.	DEPENDENCE
DEPENDENCE,PRINT,INVOKES.	DEPENDENCE
DEPENDENCE,SUMMARY.	DEPENDENCE
DEPENDENCE,TREE.	DEPENDENCE
DESCRIBE = ON/OFF.	(Universal)
DOCUMENT{,JAVSTEXT=<text-name>{,MODULE=<name-1>,...}}.	ASSIST, DEPENDENCE, (Universal)
END.	(Universal)
END FOR.	(Universal)
FOR JAVSTEXT.	(Universal)
FOR LIBRARY.	(Universal)
FOR MODULE = <name-1>,<name-2>,...,<name-n>.	(Universal)
INSTRUMENT.	INSTRUMENT
INSTRUMENT,MODE = INVOCATION/DDPATHS/DIRECTIVES/FULL.	INSTRUMENT
INSTRUMENT,PROBE,DDPATH = <probe-name>.	INSTRUMENT
INSTRUMENT,PROBE,MODULE = <invocation-name>.	INSTRUMENT
INSTRUMENT,PROBE,TEST = <test-name>.	INSTRUMENT
INSTRUMENT,STARTTEST = <modname>,<textname>,<stmt. no.> {,<TESNAM>}{,<TFLAG>}.	INSTRUMENT
INSTRUMENT,STOPTEST = <modname>,<textname>, <stmt. no.>.	INSTRUMENT
JAVSTEXT = <text-name>.	(Universal)
MERGE.	(Universal)
MODULE = <name>.	(Universal)
OLD LIBRARY = <libname>.	(Universal)

AD-A064 872 GENERAL RESEARCH CORP SANTA BARBARA CALIF F/G 9/2
JOVIAL AUTOMATED VERIFICATION SYSTEM.(U)
DEC 78 C GANNON F30602-77-C-0115
UNCLASSIFIED RADC-TR-78-247 NL

2 OF 2
AD
A 064 872



END
DATE
FILMED
4-79
DDC

JAVS COMMANDS (DEFAULTS UNDERLINED)	STEP
PRINT,DDP.	(Universal)
PRINT,DDPATHS.	(Universal)
PRINT,DMT.	(Universal)
PRINT,JAVSTEXT = <text-name-1>, INSTRUMENTED = ALL.	(Universal)
PRINT,JAVSTEXT = <text-name>,INSTRUMENTED = <name-1>, <name-2>,...,<name-n>.	(Universal)
PRINT,JAVSTEXT = <text-name>.	(Universal)
PRINT,MODULE.	(Universal)
PRINT,SB.	(Universal)
PRINT,SDB.	(Universal)
PRINT,SLT.	(Universal)
PRINT,STB.	(Universal)
PROBE, JAVSTEXT = <text-name>{,MODULE = <name-1>,...}.	INSTRUMENT, (Universal)
PROBI,STARTTEST = <modname>,<textname>,<stmt. no.> {,TESNAM}{,TFLAG}.	INSTRUMENT, (Universal)
PROBI,STOPTEST = <modname>,<textname>,<stmt. no.>.	INSTRUMENT, (Universal)
PUNCH,JAVSTEXT = <text-name>.	(Universal)
PUNCH,JAVSTEXT = <text-name>,INSTRUMENTED = ALL.	(Universal)
PUNCH,JAVSTEXT = <text-name>,INSTRUMENTED = <name-1>, <name-2>,...,<name-n>.	(Universal)
PUNCH,MODULE.	(Universal)
START.	(Startup)
STRUCTURAL.	STRUCTURAL
STRUCTURAL,PRINT = SUMMARY/DEBUG.	STRUCTURAL
TEST{,MODULE = <name-1>,<name-2>,...<name-n>}.	ANALYZER, (Universal)

*

APPENDIX B

JAVS MACRO COMMANDS

B.1 INTRODUCTION

To facilitate the use of JAVS, four new commands were added to the existing set of processing commands. These "macro" commands combine the most commonly used commands from the standard set. The macro commands may be used one at a time, all together, or in combination with the standard set of commands. The combination of commands requires understanding the expansion of commands which each macro generates; thus, the user is urged to review Sec. B.2 carefully. The four macro commands are:

1. BUILD LIBRARY [=<name>].

2. PROBE ,JAVSTEXT = <text-name>[,MODULE = <name-1>, <name-2>, ...<name-n>] .

3. TEST[,MODULE = <name-1>, <name-2>, ..., <name-n>].

4. DOCUMENT [,JAVSTEXT = <text-name>[,MODULE = <name-1>, <name-2>, ... <name-n>]].

Brackets [] indicate optional information. Each option generates a different set of standard commands.

B.2 EXPANSION OF MACRO COMMANDS

Unless the user supplies the library identification and start commands, the first occurrence of a macro command in the command set generates the standard commands:

```
CREATE LIBRARY = TEST.
START.
```

or

```
OLD LIBRARY = TEST.
START.
```

All macros except BUILD LIBRARY generate the OLD LIBRARY command.

B.2.1 Syntax and Structural Analysis

BUILD LIBRARY. generates commands:

```
CREATE LIBRARY = TEST.
START.
BASIC.
FOR LIBRARY.
STRUCTURAL.
END FOR.
```

BUILD LIBRARY = <name>. generates commands:

```
CREATE LIBRARY = <name>.
START.
BASIC.
FOR LIBRARY.
STRUCTURAL.
END FOR.
```

B.2.2 Documentation Reports

DOCUMENT. generates commands:

```
ASSIST, CROSSREF, LIBRARY.
DEPENDENCE. GROUP, LIBRARY.
DEPENDENCE, GROUP, AUXLIB.
DEPENDENCE, SUMMARY.
FOR LIBRARY.
PRINT,MODULE.
DEPENDENCE, BANDS=5
DEPENDENCE, PRINT, INVOKES.
END FOR.
```

DOCUMENT, JAVSTEXT = <text name>. generates commands

```
ASSIST, CROSSREF, LIBRARY.
DEPENDENCE, GROUP, LIBRARY.
DEPENDENCE, GROUP, AUXLIB.
DEPENDENCE, SUMMARY.
JAVSTEXT = <text name>.
FOR JAVSTEXT.
PRINT, MODULE.
DEPENDENCE, BANDS = 5.
DEPENDENCE,PRINT,INVOKES.
END FOR.
```

DOCUMENT, JAVSTEXT = <text-name>, MODULE = <name-1>, ..., <name-n>. generates commands:

```
ASSIST, CROSSREF, LIBRARY.
DEPENDENCE, GROUP, LIBRARY.
DEPENDENCE, GROUP, AUXLIB.
DEPENDENCE, SUMMARY.
JAVSTEXT = <text name>.
FOR MODULE = <name-1>, <name-2>, ..., <name-n>.
PRINT, MODULE.
DEPENDENCE, BANDS = 5.
DEPENDENCE, PRINT, INVOKES.
END FOR.
```

B.2.3 Instrumentation

*

PROBE, JAVSTEXT = <text name>. generates commands:

```
JAVSTEXT = <text name>.
FOR JAVSTEXT.
INSTRUMENT.
END FOR.
PUNCH, JAVSTEXT = <text name>, INSTRUMENTED = ALL.
```

PROBE, JAVSTEXT = <text name>, MODULE = <name-1>, <name-2>, ..., <name-n>. generates commands:

```
JAVSTEXT = <text name>.
FOR MODULE = <name-1>, ..., <name-n>.
INSTRUMENT.
END FOR.
PUNCH, JAVSTEXT = <text name>, INSTRUMENTED = <name-1>,
..., <name-n>.
```

B.2.4 Test Boundary Insertion (quasi-macro commands)

In order to identify test cases and control the recording of data on the AUDIT file, the user must supply invocations to the data collection routine, PROBI. The invocations can be manually inserted prior to Test Execution, or they can be automatically inserted during instrumentation.

The PROBI commands cause JAVS to insert an invocation to PROBI for identifying a new test case or terminating the test. The commands are of the form:

```
PROBI,STARTTEST =  <m-name>, <t-name>, <no.>{,<TESNAM>,<TFLAG>}.
PROBI,STOPTEST = <m-name>, <t-name>, <no.>.
```

where

```
m-name = module name
t-name = Javstext name
   no. = statement number
TESNAM = test case identifier    DEFAULT = 8H(CASE    )
 TFLAG = tracing level           DEFAULT = 2
```

APPENDIX C

JAVS FILES

C.1 INTRODUCTION

The files used in JAVS processing are listed in Table C.1 together with important characteristics about each file. On systems which allocate files by number (e.g., GCOS) the file number is used; on those which allocate the file by name (e.g., GOLETA), the file name is used. The data structure column indicates the contents of the file. The mode indicates how JAVS references the file. The storage form and record format describe how the data is recorded. The recommended allocation suggests an appropriate type of system file, keeping in mind that random files must be on direct access devices and sequential files may be on either direct access devices or serial devices. The usage indicates how each file is utilized for different types of JAVS processing.

The JAVS Reference Manual[1] contains a detailed description of file usage for each processing step.

C.2 RANDOM FILES

LIBOLD and LIBNEW are used for the JAVS data base library. LIBWSP is always used for working space. On all of these random files, the JAVS Storage Manager allocates space for each JAVS table in contiguous groups of 500 words called "fragments." Each file is treated internally as a word-addressable file, although it may be recorded in another form (e.g., as fixed-length records). The wrapup summary at the end of each JAVS execution contains the current size for each of these files.

C.3 SEQUENTIAL FILES

COMMAN, COMMAC and COMAUX are used for JAVS commands. COMMAC and COMMAN have a card image record for each command line; COMAUX (always shorter than COMMAN) is used to store the commands within an iteration sequence. COMMAN and COMMAC must always be allocated for any processing step. COMAUX must be allocated whenever any FOR command is used.

LOUT contains all JAVS reports destined for the line printer. The number of records on LOUT depends on the number and types of reports produced. The example reports in the JAVS Reference Manual are useful in estimating the size of LOUT. LOUT is always needed in any processing step.

LPUNCH contains instrumented (or non-instrumented) source (in card image form) destined for the JOVIAL compiler. The card image source can be written at any time as long as it was saved on the database library. Usually, LPUNCH is written during the instrumentation activity.

AUDIT contains the Test Execution probe data. It is possible to record three types of records on AUDIT. The types of records actually recorded can be controlled during INSTRUMENT processing by the MODE option and during Test Execution by an argument value to PROBI. Clearly, for a fully instrumented program with complete tracing and a large number of DD-path executions, the number of records on AUDIT can become very large (to say nothing about the added processing time). AUDIT is used in Test Execution and ANALYZER.

TABLE C.1

FILES USED IN JAVS PROCESSING

FILE NUMBER	FILE NAME	DATA STRUCTURE	MODE (1)	STORAGE FORM (2)	RECORD FORMAT	RECOMMENDED ALLOCATION	PROCESSOR USAGE (3)							
							UNIVERSAL COMMANDS	BASIC	STRUCTURAL	INSTRUMENT	ASSIST	DEPENDENCE	TEST EXECUTION	ANALYZER
1 (4)	LIBOLD	library	B	R	system standard (6)	permanent file (8)	R	R	R	R	R	R		R
2 (5)	LIBNEW	library	B	R	system standard (6)	permanent file (8)	R/W	R/W	R/W	R/W	R/W	R/W		R/W
3	LIBWSP	workspace	B	R	system standard (6)	scratch file (9)	R/W[4]	R/W[4]	R/W[4]	R/W	R/W	R/W		R/W
4 (10)	COMAUX	iteration commands	H	S	card image	scratch file (9)		R/W	R/W	R/W	R/W	R/W		R/W
5	COMMAC	user commands	H	S	card image	system card reader	R	R	R	R	R	R		R
6	LOUT	reports	H	S	128 characters/ line maximum	system printer	W	W	W	W	W	W	W(7)	W
7 (11)	LPUNCH	instrumented source	H	S	card image	system punch/ permanent file (8)	W							
8	AUDIT	probe test data	B	S	8 machine words	permanent file (8)/ magnetic tape							W	R
9	READER	JOVIAL source	H	S	card image	card file/ permanent file (8)		R						
10	COMMAN	user commands	H	S	card image	scratch file (9)	R/W	R/W	R/W	R/W	R/W	R/W		R/W

Notes:
(1) B = binary; H = BCD
(2) R = random; S = sequential
(3) R = read only; R/W = read and/or write; W = write only
(4) required with OLD LIBRARY command
(5) required with CREATE LIBRARY or ALTER LIBRARY command; required for ASSIST and ANALYZER
(6) installation dependent
(7) output on standard print file
(8) permanent file must be previously created by a FILSYS activity (see GCOS File System reference manual or Control Cards Reference Manual, under SPRMFL)
(9) mass storage for scratch file (see GCOS reference manual or Control Cards Reference Manual, under SFILE)
(10) required with any FOR command
(11) required with any PUNCH command

In general there is no way to estimate the size of the AUDIT file, since it depends on the execution behavior of the program being analyzed.

READER contains the JOVIAL source (in card image form) to be analyzed by JAVS. READER is used by the Syntax Analyzer (BASIC).

APPENDIX D

SAMPLE JOB STREAMS FOR RADC

D.1 PERFORM SYNTAX AND STRUCTURAL ANALYSES

```
$     IDENT      <userid>,<user name>,<acc't. no.>
$     USERID     <userid>$<password>
$     SELECT     BFCBGRC1/JAVSXQ
$     PRMFL      02, R/W,R,<userid>/<library file name>
$     PRMFL      09, R,S,<userid>/<source file name>
BUILD LIBRARY.
$     ENDJOB
```

Notes:

* (1) The LIMITS control card imbedded in the SELECT stream specifies .99 CP hour and 40,000 lines of output. The user can modify these limits by placing the following control card before the JAVS command:

```
$      LIMITS      <time>,53K,-5K,<lines>
```

* (2) To obtain the JAVS enhanced listing for each module during this activity, follow BUILD LIBRARY with:

```
FOR LIBRARY.
PRINT,MODULE.
END FOR.
```

(3) To obtain JAVS documentation reports within the same activity, follow BUILD LIBRARY with:

```
DOCUMENT.
```

(4) The default library name is TEST. The user may provide any library name (up to eight characters) but in doing so must provide the library identification and start commands in subsequent JAVS processing jobs to reflect the non-default library name. See page B-2 for more information.

D.2 OBTAIN JAVS DOCUMENTATION REPORTS

```
$      IDENT      <userid>,<user name>,<acc't. no.>
$      USERID     <userid>$<password>
$      SELECT     BFCBGRC1/JAVSXQ
$      PRMFL      01, R,R,<userid>/<library file name>
DOCUMENT.
$      ENDJOB
```

Notes:

(1) The LIMITS control card imbedded in the SELECT stream specifies .99 CP hour and 40,000 lines of output. The user can modify these limits by placing the following control card before the JAVS command:

```
$      LIMITS      <time>,53K,-5K,<lines>
```

(2) To obtain the JAVS control flow picture for each module, in addition to the other documentation reports, <u>follow</u> the DOCUMENT command with:

```
FOR LIBRARY.
ASSIST,PICTURE.
END FOR.
```

or <u>precede</u> the DOCUMENT command with:

```
OLD LIBRARY = TEST.†
START.
FOR LIBRARY.
ASSIST,PICTURE.
END FOR.
```

† The default library name is TEST. The user may provide any library name (up to eight characters) but in doing so must provide the library identification and start commands in subsequent JAVS processing jobs to reflect the non-default library name. See page B-2 for more information.

D.3 INSTRUMENT A START-TERM SEQUENCE (JAVSTEXT)[+]

```
$     IDENT      <userid>,<user name>,<acc't. no.>
$     USERID     <userid>$<password>
$     SELECT     BFCBGRC1/JAVSXQ
$     PRMFL      01, R,R,<userid>/<library file name>
$     PRMFL      07, W,S,<userid>/<instrumented source file name>
OLD LIBRARY = TEST.[++]
START.
PROBI,STARTTEST = <options>.
PROBI,STOPTEST = <options>.
PROBE,JAVSTEXT = <text name>.
$     ENDJOB
```

Notes:

(1) See D.2 note (1).

(2) Library identification and start commands are required for PROBI commands.

(3) To manually insert the PROBI test case boundary calls, remove the first four commands.

(4) To obtain a listing of the instrumented modules, follow the PROBE command with:

```
PRINT,JAVSTEXT = <text name>,INSTRUMENTED = ALL.
```

[+] This job stream does not save the probed code on the database library.

[++] The default library name is TEST. The user may provide any library name (up to eight characters) but in doing so must provide the library identification and start commands in subsequent JAVS processing jobs to reflect the non-default library name. See page B-2 for more information.

D.4 INSTRUMENT A START-TERM SEQUENCE (JAVSTEXT)†

```
$      IDENT      <userid>,<user name>,<acc't. no.>
$      USERID     <userid>$<password>
$      PRMFL      H*,R,R,BFCBGRC1/JAVSXQ
$      PRMFL      02, R/W,R,<userid>/<library file name>
$      PRMFL      07, W,S,<userid>/<instrumented source file name>
$      LIMITS     <time>,53K,-5K,<lines>
ALTER LIBRARY = TEST.††
START.
PROBI,STARTTEST = <options>.
PROBI,STOPTEST =  <options>.
PROBE,JAVSTEXT =  <text name>.
$      ENDJOB
```

See D.3 notes.

†This job stream saves the probed code on the database library which can substantially increase the library's file size.

††The default library name is TEST. The user may provide any library name (up to eight characters) but in doing so must provide the library identification and start commands in subsequent JAVS processing jobs to reflect the non-default library name. See page B-2 for more information.

D.5 COMPILE INSTRUMENTED SOURCE TEXT

```
$     IDENT      <userid>,<user name>,<acc't. no.>
$     USERID     <userid>$<password>
$     LOWLOAD
$     OPTION     FORTRAN
$     JOVIAL     NOPT,NDECK,NAME/PRPOOL/,POOLOU/JP/
$     FILE       JP,X1S,2L
$     SELECT     BFCBGRC1/COMPILEB
$     LIMITS     1,49K
$     SELECT     BFCBGRC2/PRPOOL
$     JOVIAL     POOLIN/JP{,user COMPOOLS}/,NOPT,
$     ETC        NAME/<name>/
$     FILE       JP,X1R,2L
$     SELECT     BFCBGRC1/COMPILEB
$     LIMITS     <time>,<size>,,<lines>
$     PRMFL      S*,R,S,<userid>/<instrumented source file>
$     PRMFL      C*,W,S,<userid>/<instrumented object file>
.
.     user COMPOOL perm files
.

$     ENDJOB
```

Notes:

(1) Currently the backup JOCIT compiler (version 042275) is being used. If the user wishes to use another version of the JOCIT compiler, the JAVS data collection routines must be recompiled using the same version.

(2) The CP time in the second LIMITS control card should be approximately 1.5 times the CP time required to compile the uninstrumented text.

(3) The core size in the second LIMITS control card should be approximately 1.25 times the size required to compile the uninstrumented text.

*

D.6 TEST EXECUTION AND POST-TEST ANALYSIS

```
$     IDENT      <userid>,<user name>,<acc't. no.>
$     USERID     <userid>$<password>
$     LOWLOAD
$     OPTION     FORTRAN
$     SELECT     BFCBGRC1/JPROBESX
$     SELECT     <userid>/<instrumented object file>
$     SELECT     BFCBGRC1/EXECUTEB
$     LIMITS     <time>,<size>,,<lines>
$     PRMFL      08, W,S,<userid>/<AUDIT file>
.
.     user files and data
.
$     SELECT     BFCBGRC1/JAVSXQ
$     LIMITS     <time>, 53K,-5K,<lines>
$     PRMFL      01, R,R,<userid>/<library file>
$     PRMFL      08, R,S,<userid>/<AUDIT file>
OLD LIBRARY = TEST.†
START.
ANALYZER,MODLST.
ANALYZER,DDPATHS.
TEST.
$     ENDJOB
```

Notes:

(1) To obtain a printed listing of the input data, if the data were on a perm file, insert the following control cards before selecting JAVSXQ:

```
$     CONVER
$     INPUT      NMEDIA
$     PRMFL      IN,R,S,<userid>/<data file>
$     PRINT      OT
```

(2) The EXECUTEB select stream supplies JOVIAL system routines used by the backup JOCIT compiler. This can be changed to EXECUTE if all software modules were compiled using a newer version of JOCIT.

(3) Additional instrumented files can be loaded.

(4) In an overlay environment, JPROBESX must be loaded in the main link.

(5) The AUDIT file can be a scratch disk file or magnetic tape, instead of a perm file.

† The default library name is TEST. The user may provide any library name (up to eight characters) but in doing so must provide the library identification and start commands in subsequent JAVS processing jobs to reflect the non-default library name. See page B-2 for more information.

D.7 RETESTING ASSISTANCE (REACHING SETS)

```
$     IDENT      <userid>,<user name>,<acc't. no.>
$     USERID     <userid>$<password>
$     SELECT     BFCBGRC1/JAVSXQ
$     PRMFL      01, R,R,<userid>/<library file name>
OLD LIBRARY = TEST.+
START.
ASSIST,REACHING SET,<target>,{options}.
.
.
.
$     ENDJOB
```

+ The default library name is TEST. The user may provide any library name (up to eight characters) but in doing so must provide the library identification and start commands in subsequent JAVS processing jobs to reflect the non-default library name. See page B-2 for more information.

D.8 SELECT STREAMS

In the event that the user wishes to modify the control cards nested in the two JAVS SELECT control cards, the expansions are as follows:

```
$     SELECT BFCBGRC1/JAVSXQ                    contains

$     PROGRAM RLHS
$     PRMFL   H*,R,R,BFCBGRC1/HSTAR/JAVSHS
$     LIMITS  99,53K,-5K,40000
$     FILE    03,X3R,10R
$     FILE    10,C1R,1L
$     FILE    04,X4R,2L
$     FILE    02,X2R,20R
```

```
$     SELECT BFCBGRC1/JPROBESX                  contains

$     SELECT  BFCBGRC2/BPRCMPL
$     SELECT  BFCBGRC2/BPROBE
$     SELECT  BFCBGRC2/BPROBI-X
$     SELECT  BFCBGRC2/BPROBM-X
$     SELECT  BFCBGRC2/BPROBD-X
```

APPENDIX E

TIME AND SIZE ESTIMATIONS

TABLE E.1

FILE SIZE ESTIMATION

File #	File	Estimation*
01, 02	Library	15-20 llinks/module
	(LIBOLD,	300 llinks/1000 source statements
	LIBNEW)	4-5 times source file size (llinks)
03	LIBWSP	10 llinks
04	COMAUX	2 llinks
07	LPUNCH	8 llinks/module
	(instrumented source)	100 llinks/1000 source statements
		2 times uninstrumented source file size (llinks)
	Instrumented	.3 times LPUNCH (llinks)
	object file	2 times uninstrumented object file size
08	AUDIT	Minimum size (no execution tracing) is: (number of probed DD-paths x number of test cases x .2 llinks) Maximum size is dependent upon execution behavior and level of tracing
09	READER	.04 llinks/source card
10	COMMAN	1 llink

*These estimations are derived from testing the SAC Force Management Information System.

TABLE E.2

CP Time Estimation

Task/Command	CP Hour/Module	CP Hour/ 1000 Statements
BUILD LIBRARY	.010	.17
DOCUMENT	.006	.1
PROBE	.005	.07
Compile instrumented code	.001	.02
Test Execution	1.5 times execution time for uninstrumented program	
TEST*	3-6 times Test Execution time .01 CP hour/module	

*Very rough estimates, since TEST CP time depends heavily on the size of the AUDIT file.

APPENDIX F

JAVS INSTALLATION REQUIREMENTS

TABLE F.1

JAVS INSTALLATION AT RADC

DATE	June 1978
VERSION	3.0 overlay
COMPUTER	HIS 6180
OPERATING SYSTEM	GCOS version G update 3
COMPILER	JOCIT JOVIAL/J3 version 042275
NON-STANDARD FEATURES	JOVIAL MONITOR, FORTRAN random I/O
CONFIGURATION	batch, linked

PROCESSING CORE REQUIREMENTS:

Program File	Type	Load Size	Process
HSTAR/JAVSHS	H*	53K	Complete JAVS overlay
JPROBESX	select	4K	Test Execution
JAVSXQ	select		select H* and work files

FILES:

Number	Name	Allocation	Usage	Description (1)
1	LIBOLD	save	R/W	300 W/records, R, U, F
2	LIBNEW	save	R/W	300 W/record, R, U, F
3	LIBWSP	scratch	R/W	300 W/record, r, U, F
4	COMAUX	scratch	R/W	BCD, card image
5	COMMAC	card reader	R	system input, BCD
6	LOUT	printer	W	system output
7	LPUNCH	compile	W	BCD, card image
8	AUDIT	save	R/W	Binary, 8 W/record
9	READER	source	R	BCD, card image
10	COMMAN	scratch	R/W	BCD, card image

Note:

(1) W = words, R = random, U = unpartioned, F = fixed length

TABLE F.2

JAVS INSTALLATION AT SAC HEADQUARTERS

DATE	July 1978
VERSION	3.0 overlay
COMPUTER	HIS 6180
OPERATING SYSTEM	WWMCCS
COMPILER	JOCIT JOVIAL/J3 version 042275
NON-STANDARD FEATURES	JOVIAL MONITOR, FORTRAN random I/O
CONFIGURATION	batch, linked

PROCESSING CORE REQUIREMENTS:

Program File	Type	Load Size	Process
HSTAR /JAVSHS	H*	53K	Complete JAVS overlay
JPROBESX	select	4K	Test Execution
JAVSXQ	select		select H* and work files

FILES:

Number	Name	Allocation	Usage	Description (1)
1	LIBOLD	save	R/W	300 W/records, R, U, F
2	LIBNEW	save	R/W	300 W/record, R, U, F
3	LIBWSP	scratch	R/W	300 W/record, R, U, F
4	COMAUX	scratch	R/W	BCD, card image
5	COMMAC	card reader	R	system input, BCD
6	LOUT	printer	W	system output
7	LPUNCH	compile	W	BCD, card image
8	AUDIT	save	R/W	Binary, 8 W/record
9	READER	source	R	BCD, card image
10	COMMAN	scratch	R/W	BCD, card image

Note:

(1) W = words, R = random, U = unpartioned, F = fixed length.

APPENDIX G

JAVS UTILIZATION CHECKLIST

Prepare source code:

a. Insert JAVSTEXT directive as the first statement of each START-TERM sequence.

   If the START-TERM is a program, CLOSE, or PROC use:

   ".JAVSTEXT <name> COMPUTE (<COMPOOL name>)"

   The parenthetical name informs JAVS that one or more COMPOOLs are referenced, although the COMPOOL name is not checked for validity.

   If the START-TERM is a COMPOOL use:

   ".JAVSTEXT <name> PRESET"

   See page 3-5 in the User's Guide and page 1-7 in the Reference Manual for examples.

b. If JAVS computation directives (ASSERT, EXPECT, TRACE, OFFTRACE) are to be used, insert them into the source code following normal JOCIT programming rules for placement and expression syntax. See Sec. 1.5 in the Reference Manual for description of these directives.

Create files

a. Complete JAVS processing of the source code will require creation of the following files:

File code	Name	Type	Contents
01,02	LIBNEW LIBOLD	Random	JAVS data base library
07	LPUNCH	Sequential	Instrumented source
08	AUDIT	Sequential	Execution Trace

   See page E-2 of the User's Guide for size estimations of these files.

b. Additional files which the user may wish to use are the sequential C* file containing the instrumented object code (see page D-6, User's Guide) and a random access H* file containing the user's program with instrumented code.

3. Perform syntax and structural analyses

   a. Execute the job stream on page D-2, User's Guide (or one which employs BASIC and STRUCTURAL keywords; see Sec. 5 of the Reference Manual under these keyword headings).

   b. Check JAVS output for any errors. The complete list of errors is in Appendix B of the Reference Manual.

   c. If necessary, modify source and reprocess this step.

4. Obtain JAVS documentation reports

   - Execute the job stream on page D-3, User's Guide or

   - Use any of the PRINT, ASSIST and DEPENDENCE commands to produce the desired documentation reports. See Sec. 5 of the Reference Manual under the appropriate keyword headings for sample commands and output.

5. Instrument the source code

   a. For each START-TERM (JAVSTEXT) execute the job stream on page D-4, User's Guide.

   b. The PROBI commands direct JAVS to insert calls to the PROBI data collection routine to initiate and terminate test cases at specified statements in the source code (statement numbers appear in the JAVS module listings). See Sects. 5.3 and 6, User's Guide and page 2-15, Reference Manual for PROBI description. The test case initiation and termination PROBI calls can be inserted manually (e.g., under the GCOS EDIT system) in the instrumented or uninstrumented source code, or they can be inserted at the direction of the PROBI command. See page 2-17, Reference Manual for a sample listing of probed code.

6. Compile the instrumented source text

   Use the job stream on page D-6, User's Guide, supplying any COMPOOLs (in addition to the JAVS PRPOOL) required for compilation.

7. Load, execute and analyze program coverage

   a. Use the job stream on page D-7, User's Guide as a basis for determining the control cards needed for executing and analyzing the program.

b. The job control cards required for loading and executing the program differ from the user's normal sequence as follows:

   (1) The JAVS data collection routines must be loaded (JPROBESX). If the user's program is in overlay form, load JPROBESX in the main link.

   (2) Load the instrumented object code instead of the original object code (in the appropriate link, if overlaid).

   (3) Supply the AUDIT file (file code 08) for the execution trace results.

c. The JAVS post-test analysis (following user files and data) control cards include the AUDIT file (written during execution) and the JAVS data base library (LIBOLD on file code 01).

Figure G.1 shows the typical flow of operations in using JAVS. JAVS commands and the data base library are used in all activities except the compilation. The files used in the figure are the library (02,01), LPUNCH (07), object (C*), and AUDIT (08).

REFERENCES

1. C. Gannon and N. B. Brooks, JAVS Technical Report, Vol. 2: Reference Manual, General Research Corporation CR-1-722/1, June 1978.

2. N. B. Brooks and C. Gannon, JAVS Technical Report, Vol. 3: Methodology Report, General Research Corporation CR-1-722, December 1976.

APPENDIX C

UPDATES TO REFERENCE MANUAL

ABSTRACT

The JOVIAL Automated Verification System (JAVS) is a control-path testing tool which analyzes source programs written in the J3 dialect of the JOVIAL language. From the user's viewpoint, JAVS consists of a sequence of processing steps which (1) builds a database containing syntactical and structural information about his JOVIAL source text, (2) prepares documentation reports describing inter- and intra-module characteristics of the source code, (3) measures control-path coverage during program execution, and (4) assists in pinpointing untested source code and preparing additional test data.

The purpose of this document is to describe in detail JAVS processing and each of the JAVS commands. The organization of the Reference Manual follows a top-down approach. Starting with a system level description in the first section; each subsequent section describes a subset of the total system in detail.

PREFACE

This Reference Manual supersedes the JAVS Reference Manual, dated November 1976. Change bars in the page margins indicate additions and changes to the 1976 manual; asterisks denote deletions.

LIST OF JAVS REPORTS

- Revised Methodology for Comprehensive Software Testing. This report describes the methodology which underlies and is supported by JAVS. The methodology is tailored to be largely independent of implementation and language. The discussion in the text is intended to be intuitive and demonstrative. Some of the methodology is based upon the experience of using JAVS to test a large information management system. A long-term growth path for automated verification systems that supports the methodology is described.

- JAVS Technical Report: Vol. 1, User's Guide. This report is an introduction to using JAVS in the testing process. Its primary purpose is to acquaint the user with the innate potential of JAVS to aid in the program testing process so that an efficient approach to program verification can be undertaken. Only the basic principles by which JAVS provides this assistance are discussed. These give the user a level of understanding necessary to see the utility of the system. The material on JAVS processing in the report is presented in the order normally followed by the beginning JAVS user. Adequate testing can be achieved using JAVS macro commands and the job streams presented in this guide. The Appendices include a summary of all JAVS commands and a description of JAVS operation at RADC with both sample command sets and sample job control statements.

- JAVS Technical Report: Vol. 2, Reference Manual. This report describes in detail JAVS processing and each of the JAVS commands. The Reference Manual is intended to be used along with the User's Guide which contains the machine-dependent information such as job control cards and file allocation. Throughout the Reference Manual, modules from a sample JOVIAL program are used in the examples. Each JAVS command is explained in detail, and a sample of each report produced by JAVS is included with the appropriate command. The report is organized into two major parts: the first four sections describing the JAVS system and the fifth section containing the description of each JAVS command in alphabetical order.

  The Appendices include a complete listing of all error messages directly produced by JAVS processing.

- JAVS Computer Program Documentation: Vol. 1, System Design and Implementation. This report contains a description of JAVS software design, the organization and contents of the JAVS data base, and a description of the software for each JAVS component: its function, each of the modules in the component, and the global data structures used by the component. The report is intended primarily as an informal reference for use in JAVS software maintenance as a companion to the Software Analysis reports described below. Included in the appendices are the templates for probe code inserted by instrumentation processing for both structural and directive instrumentation and an alphabetical list of all modules in the system (including system routines) with the formal parameters and data type of each parameter.

- JAVS Computer Program Documentation: Vol. 2, Software Analysis. This volume is a collection of computer output produced by JAVS standard processing steps. The source for each component of the JAVS software has been analyzed

to produce enhanced source listings of JAVS with indentation and control structure identification, inter-module depencence, all module invocations with formal and actual parameters, module control structure, a cross reference of symbol useage, tree report for each leading module, and report showing size of each component. It is intended to be used with the System Design and Implementation Manual for JAVS software maintenance. The Software Analysis reports, on file at RADC, are an excellent example of the use of JAVS for computer software documentation.

* • JAVS Final Report. The final report for the project describes the design, implementation and testing of the JAVS syntax analyzer. Background information regarding all JAVS contracts is provided as well as procedures for installing the complete JAVS software package. This report contains, as appendices, the June 1978 updated pages for the User's Guide and Reference Manual published as RADC-TR-77-126, Vols. I and II, April 1977.

```
JAVS TEST CASE
*** MONITORED HOLLERITH DATA ID          = TWO
*** MONITORED HOLLERITH DATA ITER1A      =  300
*** MONITORED HOLLERITH DATA ITER2A      =  199

RESULT GT 4
*** MONITORED HOLLERITH DATA ID          = ONE
*** MONITORED HOLLERITH DATA ITER1A      =   45
*** MONITORED HOLLERITH DATA ITER2A      =   14

RESULT GT 4
```

Figure 1.4. Output from Execution of Sample Program

1.4 TEXT IDENTIFICATION

JAVS automatically separates a START-TERM sequence of JOVIAL source which contains executable JOVIAL statements (i.e., not a COMPOOL) into modules of invokable code (e.g., a PROC, a CLOSE). After the source code has been instrumented, it is reassembled into the START-TERM sequence in preparation for compilation and Test Execution. If more than one START-TERM sequence is contained on the library, it is necessary to identify each sequence separately with a unique name. JAVS has provision for naming a START-TERM sequence and distinguishing between types of text through the use of a special form of comment called a JAVSTEXT directive:

".JAVSTEXT<name><type>[(<related-text-list>)][<description>]"

where

<name>	is the name given to the START-TERM sequence (name must be 8 or less characters, in which the first 6 characters must be unique among the JAVSTEXT names)
<type>	is COMPUTE for an executable START-TERM text (e.g., a program) or PRESET for nonexecutable text (e.g., a COMPOOL)
<related-text-list>	is an optional list of related text names separated by commas (e.g., the name of a COMPOOL text used during compilation with a program text)
<description>	is optional descriptive information

The JAVSTEXT directive should be used with each START-TERM sequence as the first line of text in the START-TERM sequence.

In the example program, the JAVSTEXT directive

".JAVSTEXT EXCOMPL PRESET"

is used to inform JAVS that a COMPOOL is being processed. The JAVSTEXT directive

".JAVSTEXT EXPROGM COMPUTE (EXCOMPL)"

is used to inform JAVS that an executable START-TERM sequence which references a COMPOOL (EXCOMPL) is being processed. This directive is required for a COMPOOL text, but may be omitted for executable text.

2.3 MODULE INSTRUMENTATION

This step performs the instrumentation of modules which have been processed through BASIC and STRUCTURAL. The primary result of this step is a set of probed modules which can then be compiled in instrumented form for use in Text Execution (see Sec. 2.4). Only modules which have executable statements (e.g. modules other than COMPOOLS) are instrumented. The instrumented modules are logically identical to the original source code. The probe statements are saved on the library along with data generated by BASIC and STRUCTURAL (e.g., the source text, etc.). INSTRUMENT processing is shown in Fig. 2.5.

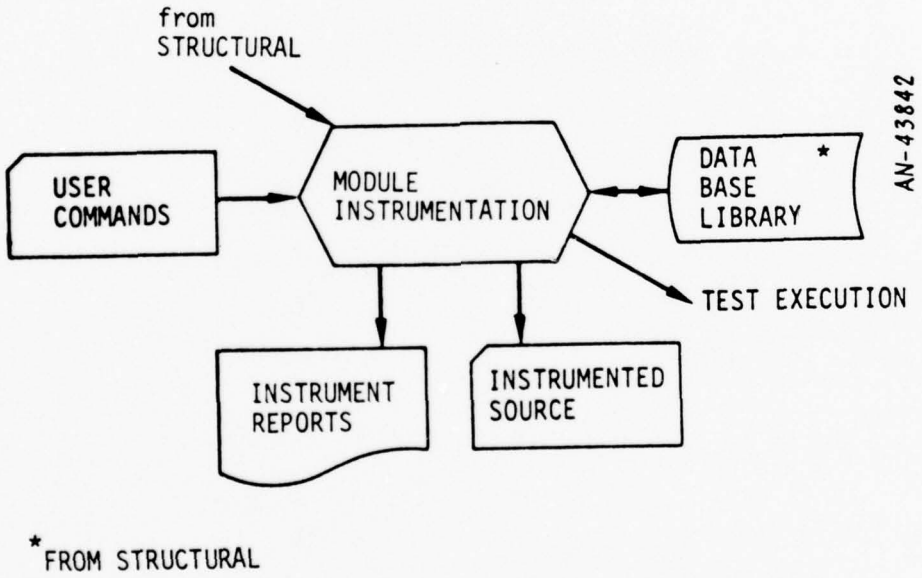


Figure 2.5. INSTRUMENT Processing

There are two types of instrumentation generated during INSTRUMENT:

1. Structural Instrumentation. Software probes are automatically inserted into the source text at each invocation point, each return point, and each statement which begins a DD-path. Each probe includes a call to special auditing routines which capture and record information concerning flow of control in the executing module(s).

2. JAVS Directive Instrumentation. Software probes are manually inserted in the source text for JAVS directives (Sec. 1.5) which are automatically expanded into JOVIAL code to monitor the results of assignment and exchange statements during Test Execution. Each directive controls execution-time output which is interspersed with normal program output.

3 JAVS CONSTRAINTS

JAVS imposes certain restrictions on the size of the database library, the command language, and the source text to be analyzed. Most of the limitations based on size are generous (e.g., the maximum number of nested IF statements is 100). Some of the limitations, such as those in the Test Execution process, can be raised by recompiling parts of JAVS. Some of the restrictions are based upon computer page dimensions and cannot be changed.

A number of the constraints which affect the incoming JOVIAL source code are founded in the principle of analyzing invokable modules separately. The dialects of JOVIAL recognized by JAVS allow certain constructs involving jumps to global labels, invocations of externally declared switches, and TEST statements where the FOR variable is external to the module. Some of these constructs require modification of the JOVIAL source code, and some cause a warning message to be issued to make the user aware of limitations in the DD-path test measurement report and execution tracing.

The constraints are listed in sections (Sec. 3.2 through 3.8) appropriate to a particular JAVS processing step. Universal constraints (Sec. 3.1) affect all of JAVS processing. The terminology used in describing the constraints requires knowledge of JOVIAL and JAVS. The user should refer to the JOCIT Compiler Users Manual[3] for JOVIAL terms and to the index in this manual for direction to the descriptions and references to JAVS terms.

3.1 UNIVERSAL CONSTRAINTS

1. Maximum 3 continuation cards for any given JAVS command.
2. Maximum 24 commas in any given JAVS command.
3. Maximum 150 modules selected for any given JAVS command iteration loop.
4. Maximum 250 word-pairs in a statement block.*
5. Maximum 100 statements on any DD-path.
6. Maximum 25 internal data base tables during any JAVS execution.
7. 100 JAVS errors produce a fatal error.
8. Maximum 250 known modules during any one JAVS execution (i.e., total modules in one or both libraries).
9. In order to change a previously made library, the name specified for an ALTER LIBRARY must be the same as when it was a CREATE LIBRARY.
10. Maximum 150 modules per JAVSTEXT.
11. Maximum 80 characters per card image read.
12. Maximum 128 characters per line of output.
13. No analysis is performed on a DIRECT code sequence.
14. The recognized dialects of JOVIAL include JOCIT JOVIAL, JOVIAL/J3, CDC JOVIAL, and Honeywell JOVIAL. The combination of these is processed by JAVS.

*BASIC automatically separates long source statements into a sequence of statements. The first statement is given the statement type according to the original JOVIAL source statement. Subsequent statements are of type continuation (CONT).

3.2 JAVS BASIC CONSTRAINTS

BASIC processing is capable of handling quite large source text files. Unusually large programs may have to be processed by several successive executions of BASIC, each operating on a separate file of START-TERM texts.

The following implementation constraints are the current ones which must be observed during BASIC processing:

1. Each module placed on the same library must have a unique module name for a given JAVSTEXT name. For this purpose only the first eight characters of any name are used. The first six characters should be unique (a compiler restriction).

2. A PROC must contain at least one executable statement (e.g. RETURN).

3. Statement labels in direct code are not analyzed. A reference to such a label in JOVIAL code is treated as a reference to an external undefined label.

4. The maximum number of nested modules is 150.

5. The maximum number of unique symbols (names and constants) is 4096.

6. A basic element may not exceed 500 characters (does not include literals).

7. A JOVIAL symbol may not exceed 4095 characters.

8. A comment if saved will be truncated to the maximum JOVIAL symbol length if it exceeds that length.

9. BASIC guarantees that saved comments terminate with a double prime (i.e., a double prime will be generated).

10. Only the first 72 columns of source text line are analyzed.

11. COMPOOLS must have a JAVSTEXT directive stating the PRESET type.

12 A statement name following a TERM (main program only) will not determine the first executable statement.

3.3 JAVS STRUCTURAL CONSTRAINTS

1. Control should not transfer from one module to a label or switch declared in another module. Control transfers of this type are treated as RETURNs.

2. A TEST statement must not appear in the range of a single factor FOR statement sequence.

3. Function calls with side effects (i.e., two successive calls to the function which produce different results) are not permitted in FOR, IF, IFEITH, and ORIF statements.

4. The maximum depth of nesting of control or compound statements (IFEITH, ORIF, IF, FOR, BEGIN) is 100.

5. The maximum number of DD-paths which can begin at a statement is 100.

6. The maximum number of statements on a single DD-path is 100.

7. CLOSE invocations which appear as switch points in SWITCH declarations are treated as null switch points.

8. SWITCH invocations which appear as switch points (nested switches) in SWITCH declarations are treated as null switch points.

9. Invocations of externally declared switches are treated as RETURNs.

10. The first three factor FOR statement in a parallel FOR is assumed to be the controlling FOR. If there are no three factor FOR statements in the parallel FOR, the first FOR statement is chosen as the controlling FOR for the purpose of constructing interstatement pointers in the JAVS statement descriptor blocks.

11. CLOSE declarations within a FOR statement may not use a TEST statement to reference the active FOR variable.

3.4 JAVS INSTRUMENT CONSTRAINTS

1. The first three factor FOR statement in a parallel FOR is assumed to be the controlling FOR and is instrumented.

2. Invocations of externally declared switches are not instrumented.

3. The maximum number of nested IF statements is 100.

4. Subscript expressions in SWITCH invocations are limited to 72 characters.

5. Item names and switch names are limited to 30 characters in length.

6. The maximum number of variables allowed in a single TRACE directive is 18.

7. The maximum number of variables in TRACE directives is limited to 999 per module.

8. The maximum number of nested IFEITH and/or IFEITH/ORIF statements is 100.

9. The maximum DD-path number is 9999.

10. The maximum length of the TEXT in an ASSERT directive is 72 characters.

11. The functional modifiers BIT, CHAR, MANT, NENT, POS and BYTE may not appear in the numeric formulas for FOR statements; i.e., no side-effects are allowed in the initial value formula.

12. FOR variables may not appear in TRACE or EXPECT directives.

3.5 JAVS ASSIST CONSTRAINTS

1. Maximum of 100 DD-paths per reaching set path.
2. Maximum of 100 outways per decision.
3. Maximum of 1200 DD-paths per analyzed module for reaching set.
4. Maximum of 2400 statements per analyzed module for reaching set.
5. Maximum of 200 statements in reaching set.
6. Maximum 100 JAVSTEXTs specified for cross-reference mapping.
7. Maximum of 125 modules specified for cross-reference mapping.

4.3.3 Multiple Module Iteration

In many applications, the user will want to repeat a command for a number of different modules. The three forms of command iteration structure are described below.

4.3.3.1 Selected Module Iteration.

The following sequence selects a number of modules, by name, and iterates a block of commands (which cannot contain another iteration) once for each specified module:

```
FOR MODULE = <name-1>,<name-2>,...,<name-n>.

(any set of commands)

END FOR.
```

4.3.3.2 ALL-Modules Iteration for Specified JAVSTEXT (START-TERM Text).

The following sequence selects each known module within the "current text" and iterates a block of commands (which cannot contain another iteration) once for each known module of that text:

```
FOR JAVSTEXT.

(any set of commands)

END FOR.
```

4.3.3.3 ALL-Modules Iteration for Library.

The following sequence selects each known module on the library and iterates a block of commands (which cannot contain another iteration) once for each known module:

```
FOR LIBRARY.

(any set of commands)

END FOR.
```

4.3.4 Module Selection Constraint

The maximum number of modules which may be specified by a single iteration is 150 (see Sec. 3).

4.4 PROCESS OPTION COMMANDS

Processing steps BASIC, STRUCTURAL, INSTRUMENT and ANALYZER have option commands which define the action taken when the process execution command is given. The option commands follow the START command and precede the appropriate execution command (see Sec. 4.5).

4.4.1 BASIC Option Commands

The BASIC options are specified by the following commands (with the default value assumed in case the option command is not given prior to the appearance of the BASIC execution command):

BASIC,CARD IMAGES = ON/OFF.	DEFAULT = ON.
BASIC,COMMENTS = ON/OFF.	DEFAULT = ON.
BASIC,DEFINES = ON/OFF.	DEFAULT = ON.

BASIC options which have been selected in the command sequence remain in effect until they are reset by a later command. See Sec. 5 for a complete definition and example of each BASIC option command.

The user will note that module selection commands do not apply until after a module has been added to a library during BASIC processing by the BASIC verb. The name assigned to a module when it is added to a library is the first eight characters of the program name, procedure name, or close name of the module. The first six characters of the names should be unique.

It is important that the command sequence involving one or more occurrences of the BASIC execution command be terminated with the END command so that LIBNEW is closed properly. The user may employ the standard print and punch commands (Sec. 5) after the BASIC command and after module or JAVSTEXT specification (Sec. 4.3).

4.4.2 STRUCTURAL Option Command

The STRUCTURAL print option is specified by the following command (with the specified default value assumed in case the command is not given prior to the appearance of the STRUCTURAL verb):

STRUCTURAL,PRINT = SUMMARY/DEBUG. (DEFAULT = SUMMARY)

5 JAVS COMMANDS

This section contains a complete list of JAVS commands, in alphabetical order, along with the JAVS processing step in which each command is used. The term "universal" indicates that the command can be used in any processing step.*

Following the list of commands is a description, accompanied with sample output and command sets, of each JAVS command.

*The overlay version of JAVS allows all commands to be "universal."

JAVS COMMANDS (DEFAULTS UNDERLINED)	STEP
ALTER LIBRARY = <libname>.	(Universal)
ANALYZER.	ANALYZER
ANALYZER,ALL.	ANALYZER
ANALYZER,ALL MODULES.	ANALYZER
ANALYZER,CASES = <number>.	ANALYZER
ANALYZER,DDPATHS.	ANALYZER
ANALYZER,DDPTRACE.	ANALYZER
ANALYZER,FACTOR = <percent-increase>.	ANALYZER
ANALYZER,HIT.	ANALYZER
ANALYZER,MODLST.	ANALYZER
ANALYZER,MODTRACE.	ANALYZER
ANALYZER,MODULE = <name-1>,<name-2>,...,<name-n>.	ANALYZER
ANALYZER,NOTHIT.	ANALYZER
ANALYZER,SUMMARY.	ANALYZER
ANALYZER,TIME.	ANALYZER
ASSIST,CROSSREF,JAVSTEXT = <text-name-1>,<text-name-2>,..., <text-name-n>.	ASSIST
ASSIST,CROSSREF,LIBRARY.	ASSIST
ASSIST,PICTURE.	ASSIST
ASSIST,PICTURE{,CONTROL}{,NOSWITCH}.	ASSIST
ASSIST,REACHING SET,<number-to>{,<number-from>} {,PICTURE{,ITERATIVE}}.	ASSIST
ASSIST,STATEMENTS.	ASSIST
BASIC.	BASIC
BASIC,CARD IMAGES = ON/OFF.	BASIC
BASIC,COMMENTS = ON/OFF.	BASIC
BASIC,DEFINES = ON/OFF.	BASIC
BUILD LIBRARY {= <library name>}.	BASIC, STRUCTURAL
CREATE LIBRARY = <libname>.	(Universal)

*

*

JAVS COMMANDS (DEFAULTS UNDERLINED)	STEP
DEPENDENCE,BANDS.	DEPENDENCE
DEPENDENCE,BANDS = <number>.	DEPENDENCE
DEPENDENCE,GROUP,AUXLIB.	DEPENDENCE
DEPENDENCE,GROUP,LIBRARY.	DEPENDENCE
DEPENDENCE,GROUP,MODULES = <name-1>,<name-2>,...,<name-n>.	DEPENDENCE
DEPENDENCE,PRINT,INVOKES.	DEPENDENCE
DEPENDENCE,SUMMARY.	DEPENDENCE
DEPENDENCE,TREE.	DEPENDENCE
DESCRIBE = ON/OFF.	(Universal)
DOCUMENT{,JAVSTEXT=<text-name>{,MODULE=<name-1>,...}}.	ASSIST, DEPENDENCE, (Universal)
END.	(Universal)
END FOR.	(Universal)
FOR JAVSTEXT.	(Universal)
FOR LIBRARY.	(Universal)
FOR MODULE = <name-1>,<name-2>,...,<name-n>.	(Universal)
INSTRUMENT.	INSTRUMENT
INSTRUMENT,MODE = INVOCATION/DDPATHS/DIRECTIVES/FULL.	INSTRUMENT
INSTRUMENT,PROBE,DDPATH = <probe-name>.	INSTRUMENT
INSTRUMENT,PROBE,MODULE = <invocation-name>.	INSTRUMENT
INSTRUMENT,PROBE,TEST = <test-name>.	INSTRUMENT
INSTRUMENT,STARTTEST = <modname>,<textname>,<stmt. no.> {,<TESNAM>}{,<TFLAG>}.	INSTRUMENT
INSTRUMENT,STOPTEST = <modname>,<textname>, <stmt. no.>.	INSTRUMENT
JAVSTEXT = <text-name>.	(Universal)
MERGE.	(Universal)
MODULE = <name>.	(Universal)
OLD LIBRARY = <libname>.	(Universal)

*

JAVS COMMANDS (DEFAULTS UNDERLINED)	STEP
PRINT,DDP.	(Universal)
PRINT,DDPATHS.	(Universal)
PRINT,DMT.	(Universal)
PRINT,JAVSTEXT = <text-name-1>, INSTRUMENTED = ALL.	(Universal)
PRINT,JAVSTEXT = <text-name>,INSTRUMENTED = <name-1>, <name-2>,...,<name-n>.	(Universal)
PRINT,JAVSTEXT = <text-name>.	(Universal)
PRINT,MODULE.	(Universal)
PRINT,SB.	(Universal)
PRINT,SDB.	(Universal)
PRINT,SLT.	(Universal)
PRINT,STB.	(Universal)
PROBE, JAVSTEXT = <text-name>{,MODULE = <name-1>,...}.	INSTRUMENT, (Universal)
PROBI,STARTTEST = <modname>,<textname>,<stmt. no.> {,TESNAM}{,TFLAG}.	INSTRUMENT, (Universal)
PROBI,STOPTEST = <modname>,<textname>,<stmt. no.>.	INSTRUMENT, (Universal)
PUNCH,JAVSTEXT = <text-name>.	(Universal)
PUNCH,JAVSTEXT = <text-name>,INSTRUMENTED = ALL.	(Universal)
PUNCH,JAVSTEXT = <text-name>,INSTRUMENTED = <name-1>, <name-2>,...,<name-n>.	(Universal)
PUNCH,MODULE.	(Universal)
START.	(Startup)
STRUCTURAL.	STRUCTURAL
STRUCTURAL,PRINT = SUMMARY/DEBUG.	STRUCTURAL
TEST{,MODULE = <name-1>,<name-2>,...<name-n>}.	ANALYZER, (Universal)

*

ALTER LIBRARY = <name>. ALTER LIBRARY = <name>.

Description

This option specifies that LIBNEW is a previously generated library to be modified during the current run. LIBNEW is used as a read/write library.

Rule

The name of LIBNEW must be the same one used when it was first created.

Example Command Sets

(1)
```
ALTER LIBRARY = REFMAN.
START.
FOR LIBRARY.
STRUCTURAL.
END FOR.
END.
```

This command set will modify the library created by the BASIC processing step and add the STRUCTURAL data to the library for each module on the library.

(2)
```
ALTER LIBRARY = REFONE.
START.
BASIC.
END.
```

This command set augments a previously built library with the BASIC processing of additional source text.

ANALYZER.

Description

The ANALYZER execution command causes processing of the post-test analysis. If no ANALYZER report options are present, no report is produced.

Rules

For a single ANALYZER command,

(1) Either option MODTRACE or DDPTRACE can be used, but not both.

(2) A maximum of 100 test cases is analyzed.

(3) Only one ANALYZER execution command can be used per set of commands.

Example Command Sets

(1)
```
OLD LIBRARY = REFMAN.
START.
ANALYZER,ALL MODULES.
ANALYZER,ALL.
ANALYZER.
END.
```

This set of commands will produce the SUMMARY, HIT, NOTHIT, TIME, DDPATHS, MODLST, DDPTRACE post-test analysis reports for all modules on the library.

(2)
```
OLD LIBRARY = TRFMAN.
START.
ANALYZER,CASES = 50.
ANALYZER,MODULE = EXPROGM,EXMPL1.
ANALYZER,SUMMARY.
ANALYZER,NOTHIT.
ANALYZER,DDPATHS.
ANALYZER,DDPTRACE.
ANALYZER.
END.
```

BASIC,COMMENTS = <option>.

BASIC,COMMENTS = <option>.

Description

BASIC,COMMENTS = ON/OFF. (DEFAULT = ON.)

This command allows user control over the treatment of comments in the module source text. The default action is to include all comments in the library. Comments may be excluded from the statement text table of all modules by the command

BASIC,COMMENTS = OFF.

If comments are included, more space is required on the library and all subsequent processing of the statement table requires more computer time.

Sample Command Set

```
CREATE LIBRARY = REFMAN.
START.
BASIC,COMMENTS = OFF.
BASIC.
END.
```

BASIC,DEFINES = <option>.

BASIC,DEFINES = <option>.

Description

BASIC,DEFINES = ON/OFF. (DEFAULT = ON.)

This command allows the user control over BASIC usage of DEFINE variables. The default option is to replace the occurrence of each DEFINE variable with its definition just as the JOVIAL compiler does. Subsequent BASIC processing utilizes the results of that definition both in statement analysis and in the text stored for each statement on the library. It is essential that the default option be used if the modules are to be processed by any other JAVS standard processing step.

Replacement of DEFINE variables by their definition may be suppressed with the command:

BASIC,DEFINES = OFF.

Text so processed may be written to LPUNCH in the structured (i.e., indented) format with the command:

PUNCH,JAVSTEXT = <text-name>.

This is useful in using JAVS to punch a copy of reformatted source text. To get a reformatted listing of the source text, the PRINT,JAVSTEXT command can be used.

Description

This macro command has two forms:

```
BUILD LIBRARY.        or

BUILD LIBRARY = <name>.
```

In the first command form, the library name given to LIBNEW is TEST. In the second form, the user specifies the library name. Both command forms generate the following JAVS standard commands:

```
CREATE LIBRARY = <name>.     (or TEST)
START.
BASIC,COMMENTS = ON.
BASIC.
FOR LIBRARY.
STRUCTURAL.
END FOR.
```

The macro command processor generates the JAVS "END" command if it is not present as the last command.

Rules

(1) See BASIC and STRUCTURAL constraints in Sec. 3.2 and 3.3.

(2) The library name should be 8 or less characters.

(3) This macro command should not be used if the source program includes a COMPOOL (constraint 2 on page 3-3).

Example Command Sets

(1)
```
BUILD LIBRARY = REFMAN.
DOCUMENT.
```

This command set will create a new library, perform syntax and structural analyses and produce a series of JAVS reports for all modules in the JOVIAL source program. These reports are useful for documentation, maintenance, testing and retesting the source program.

(2)
```
BUILD LIBRARY.
PROBE,JAVSTEXT = EXPROGM.
PRINT,JAVSTEXT = EXPROGM,INSTRUMENTED = ALL.
```

---

*Can be used only with the overlay version of JAVS.

BUILD LIBRARY. (Cont.)

This command set shows a mixture of JAVS macro and standard commands. The first macro command will create a new library called TEST and perform syntax and structural analyses on all modules in the JOVIAL source program. The PROBE macro command will instrument JAVSTEXT EXPROGM, using the default instrumentation options, and write the instrumented text onto file LPUNCH. The last command will print the instrumented text.

Description

This macro command has three forms:

(1) DOCUMENT.

(2) DOCUMENT,JAVSTEXT = <text-name>.

(3) DOCUMENT,JAVSTEXT = <text-name>,
MODULE = <name-1>,<name-2>,...<name-n>.

Each form of the DOCUMENT macro command generates a series of JAVS standard commands which produce reports useful for program documentation, maintenance, and testing.

The expansion of each form of the DOCUMENT macro command is as follows:

(1)
```
ASSIST,CROSSREF,LIBRARY.
DEPENDENCE,GROUP,LIBRARY.
DEPENDENCE,GROUP,AUXLIB.
DEPENDENCE,SUMMARY.
FOR LIBRARY.
PRINT,MODULE.
DEPENDENCE,BANDS = 5.
DEPENDENCE,PRINT,INVOKES.
END FOR.
```

This form (DOCUMENT.) is for documenting the entire library.

(2)
```
ASSIST,CROSSREF,LIBRARY.
DEPENDENCE,GROUP,LIBRARY.
DEPENDENCE,GROUP,AUXLIB.
DEPENDENCE,SUMMARY.
JAVSTEXT = <text-name>.
FOR JAVSTEXT.
PRINT,MODULE.
DEPENDENCE,BANDS = 5.
DEPENDENCE,PRINT,INVOKES.
END FOR.
```

This form (DOCUMENT,JAVSTEXT = <text-name>.) is for documenting module interdependencies for the entire library, producing a library-wide cross reference of symbols, and generating module documentary reports for the specified JAVSTEXT.

DOCUMENT. (Cont.)

(3)
```
ASSIST,CROSSREF,LIBRARY.
DEPENDENCE,GROUP,LIBRARY.
DEPENDENCE,GROUP,AUXLIB.
DEPENDENCE,SUMMARY.
JAVSTEXT = <text-name>.
FOR MODULE = <name-1>,<name-2>,...,<name-n>.
PRINT,MODULE.
DEPENDENCE,BANDS = 5.
END FOR.
```

This form (DOCUMENT,JAVSTEXT = <text-name>, MODULE = ...) produces the same report as in (2), except that the module reports are generated only for the specified modules in the JAVSTEXT.

Note

The macro command processor will generate the commands:

```
OLD LIBRARY = TEST.
START.
```

if the first JAVS command is a macro command (keywords BUILD LIBRARY, DOCUMENT,PROBE,TEST). The macro command processor will generate the "END" command, if it is not present.

Rules

(1) A maximum of 100 JAVSTEXTs can be used in a cross reference mapping.

(2) A maximum of 100 modules can be used in the GROUP reports.

(3) A maximum of 23 modules can be specified in the second form of the DOCUMENT macro.

(4) The DOCUMENT macro command requires that syntax and structural analyses have already been performed on the entire library.

Example Command Sets

(1)
```
BUILD LIBRARY.
DOCUMENT.
```

This command set will create a new library called TEST and produce JAVS documentation reports for all modules on the library.

(2)
```
OLD LIBRARY = REFMAN.
START.
DOCUMENT,JAVSTEXT = EXPROGM,MODULE = EXMPL1,EXMPL2,EXMPL3.
FOR MODULE = EXMPL1,EXMPL2,EXMPL3.
PRINT,DDPATHS.
END FOR.
```

END. (Cont.)

5. Type of module where

   CMPL is a COMPOOL

   PROG is a Program

   PROC is a Procedure

   CLSM is a CLOSE of global scope

   CLSP is a CLOSE of local scope

6. Scope of an executable module where

   EXT is a program or an external procedure

   INT is an internal procedure (i.e., within a program or external procedure)

   GLBL is a CLOSE within a program or external procedure

   LOCL is a CLOSE within an internal procedure

7. Number of lines of probed text inserted in the module by INSTRUMENT

8. Number of statements in the module

9. Number of executable statements

10. The statement number of the first executable statement

11. Word pairs which measure the amount of library space occupied by the source text

12. Number of tokens in the module

13. Number of symbols defined in module

14. Number of symbols referenced in module

The remainder of the statistics are more applicable to users who wish to follow a testing strategy of analyzing the complexity of their codes and applying testcases to more comprehensively exercise those modules which exhibit a greater tendency to be error prone. These statistics are listed below:

END. (Cont.)

15. Number of other modules invoked by module

16. Number of DD-paths

17. Number of DD-path/statement block entries

18. Number of formal input parameters

19. Number of formal output parameters (-1 if the module is a function)

20. YES if DIRECT code is present in module; NO otherwise

21. Statement-based complexity is a summation of the complexity of all statements in the module.†

22. DD-path based complexity is a summation of the complexity of all DD-paths in the module. This is used as an indicator of the structural complexity of the module.†

These statistics can be used by the tester to develop a rational approach to program verification of modules which have the greatest tendency toward error (i.e., those which are most complex).

The Library Information report shown in the sample output includes the name for each library, the type of access, the date created, the total size and other useful information.

† These items are provided for future extensions to JAVS and are not presently computed.

Description

This command produces a listing of the detailed, internal symbol table (STB) descriptors for each symbol defined in the current module. Normally the list includes only those symbols essential to structural analysis of the module. A BASIC processing option causes all symbols defined in the module to be entered in the STB (see BASIC SYMBOLS = ON). The properties of each symbol and pointers to other JAVS-internal tables are included. This output is intended for JAVS system maintenance only.

Sample Output

```
SYMBOL TABLE LISTING

MODULE <EXMPL1  >, JAVSTEXT <EXPROGM >, PARENT MODULE <EXPROGM >

NO.  SYMBOL    1    2    3    4    5  6  7    8  9  10  11  12 13  14  15 16  17  18 19  20  22     23     24  27  29  30
---------------------------------------------------------------------------------------------------------------------------
  1 PICK       2 LOCL INSW            4  0                      0  0   0   0  0   0   0  0   0  27  EXMPL1              13
  2 LABEL1     1 LOCL LABL            4 14                      0  0   0   0  0   0   0  0   0  30  EXMPL1              14
  3 LABEL2     1 LOCL LABL            4 15                      0  0   0   0  0   0   0  0   0  29  EXMPL1              15
---------------------------------------------------------------------------------------------------------------------------
```

Example Command Set

```
OLD LIBRARY = REFMAN.
START.
FOR LIBRARY.
PRINT,STB.
END FOR.
END.
```

PROBE.

Description

The PROBE macro command has two forms:

(1) PROBE,JAVSTEXT = <text name>.

(2) PROBE,JAVSTEXT = <text name>, MODULE = <name-1>,<name-2>,...,
<name-n>.

Each form of the PROBE macro command generates a series of JAVS standard commands which cause instrumentation to be performed.

The expansion of each form of the PROBE macro into JAVS standard commands is as follows:

(1) JAVSTEXT = <text-name>.
FOR JAVSTEXT.
INSTRUMENT.
END FOR.
PUNCH,JAVSTEXT = <text name>,INSTRUMENTED = ALL.

This form (PROBE,JAVSTEXT = <text name>.) should be used for instrumentation of a specified JAVSTEXT.

(2) JAVSTEXT = <text name>.
FOR MODULE = <name-1>,...,<name-n>.
INSTRUMENT.
END FOR.
PUNCH,JAVSTEXT = <text name>,INSTRUMENTED = <name-1>,...,<name-n>.

This form (PROBE,JAVSTEXT = <text name>, MODULE = <name-1>....) should be used if only selected modules of a given JAVSTEXT are to be instrumented.

PROBE. (Cont.)

With all forms of the PROBE macro, the user can specify automatic insertion of the PROBI (test case initiation and test termination data collection routine) calls by preceding the PROBE command with:

```
PROBI,STARTTEST = <modname>,<textname>,<stmt. no.>,{,TESNAM}{,TFLAG}.

PROBI,STOPTEST = <modname>,<textname>,<stmt. no.>.
```

These commands are described in Sec. 5 under their own heading. They perform the same function as the INSTRUMENT,STARTTEST and INSTRUMENT,STOPTEST commands. The existence of the PROBI commands aids the macro command processor which will insert the STARTTEST and STOPTEST commands in the generated series of commands.

Note

The macro command processor will generate the commands:

```
OLD LIBRARY = TEST.
START.
```

if the first JAVS command is a macro command (keywords BUILD LIBRARY,DOCUMENT, PROBE,TEST). The macro command processor will generate the "END" command, if it is not present.

Rules

(1) See INSTRUMENT constraints in Sec. 3.4.

(2) Maximum of 150 modules selected in the first form of the PROBE macro command.

(3) Maximum of 23 modules selected in the second form of the PROBE macro command.

(4) Use PROBI (not INSTRUMENT,STARTTEST and STOPTEST) commands with the PROBE macro.

Example Command Sets

(1)
```
BUILD LIBRARY.
PROBE,JAVSTEXT = EXPROGM.
DOCUMENT.
PRINT,JAVSTEXT = EXPROGM,INSTRUMENTED = ALL.
```

This command set will create a new library, perform syntax and structural analyses, instrument the JAVSTEXT EXPROGM, document an entire library, and

PROBE. (Cont.)

print the instrumented JAVSTEXT so that the user can see where the PROBI calls should be manually placed.

(2)
```
OLD LIBRARY = TEST.
START.
PROBI,STARTTEST = EXPROGM,EXPROGM,0.
PROBI,STARTTEST = EXMPL1,EXPROGM,9.
PROBI,STOPTEST = EXPROGM,EXPROGM,0.
PROBE,JAVSTEXT = EXPROGM.
```

This command set will instrument JAVSTEXT EXPROGM in the existing library TEST. In addition, the PROBI calls will be automatically inserted. The first PROBI,STARTTEST command will cause an invocation to PROBI to be inserted prior to the first executable statement in module EXPROGM. The second PROBI, STARTTEST command will cause a PROBI invocation to be placed immediately before statement 9 of module EXMPL1. These two PROBI invocations will initiate a new test case whenever the PROBI call is executed.

The PROBI,STOPTEST command will cause the test termination PROBI invocation to be placed at all exits from module EXPROGM. The PROBE macro command will instrument the JAVSTEXT EXPROGM and write the instrumented text, including the invocations to PROBI, onto file LPUNCH.

Note that the library definition and startup commands are included in this example. Even though the default library name is specified, these commands are required because PROBI is not a macro command.

PROBI,STARTTEST = <options>.

PROBI,STARTTEST = <options>.

Description

This command performs the same function as:

INSTRUMENT,STARTTEST = <options>.

Refer to this heading for the command description and example command sets.

Rules

(1) This command is to be used only in conjunction with the PROBE macro command and must precede the PROBE command.

(2) Maximum of 10 PROBI,STARTTEST commands with a single PROBE macro command.

(3) Command options must be given in the order shown in the command description.

(4) Modules specified in PROBI,STARTTEST commands must be selected by the PROBE macro command.

(5) Library identification and startup commands must be included in the command set.

PROBI,STOPTEST = <options>.

PROBI,STOPTEST = <options>.

Description

This command performs the same function as:

INSTRUMENT,STOPTEST = <options>.

Refer to this heading for the command description and example command sets.

Rules

(1) This command is to be used only in conjunction with the PROBE macro command and must precede the PROBE command.

(2) Only one PROBI,STOPTEST command with a single PROBE macro command.

(3) Command options must be given in the order shown in the command description.

(4) The module specified in the PROBI,STOPTEST command must be selected by the PROBE macro command.

(5) Library identification and startup commands must be included in the command set.

Description

The TEST macro command has two forms:

(1) TEST.

(2) TEST,MODULE = <name-1>,<name-2>,...,<name-n>.

The TEST macro command generates a series of JAVS ANALYZER commands for post-test analysis.

The expansion of the two forms of TEST is as follows:

(1)
```
ANALYZER,ALL MODULES.
ANALYZER,SUMMARY.
ANALYZER,NOTHIT.
ANALYZER,MODTRACE.
ANALYZER.
```

(2)
```
ANALYZER,MODULE = <name-1>,...,<name-n>.
ANALYZER,SUMMARY.
ANALYZER,NOTHIT.
ANALYZER,MODTRACE.
ANALYZER.
```

Additional post-test analysis reports can be produced by preceding the TEST macro with ANALYZER process option commands. In this case, library identification and startup commands must be included in the command set.

Note

The macro command processor will generate the commands:

```
OLD LIBRARY = TEST.
START.
```

if the first JAVS command is a macro command (keywords BUILD LIBRARY,DOCUMENT, PROBE,TEST). The macro command processor will generate the "END" command, if it is not present.

Rules

(1) The ANALYZER,DDPTRACE standard command cannot be used in conjunction with the TEST macro.

TEST. (Cont.)

(2) ANALYZER option commands must precede the TEST macro, if they are used.

(3) See ANALYZER constraints in Sec. 3.8.

Example Command Sets

(1)
```
TEST.
FOR LIBRARY.
PRINT,DDP.
END FOR.
```

(2)
```
OLD LIBRARY = REFMAN.
START.
ANALYZER,MODLST.
ANALYZER,DDPATHS.
TEST,MODULE = EXPROGM,EXMPL1.
```

B.2 BASIC ERROR MESSAGES

The error messages resulting from BASIC source code analysis are:

Error Number	Explanation
1	Basic element contains too many characters. Element truncated in saved text. Resubmit with corrected text.
2	Illegal interval text character. System error.*
3	Illegal external text character. System error.*
4	Nested DEFINE reference. Reference partially expanded. Resubmit with nested DEFINE reference corrected.
5	JOVIAL symbol too long. Symbol truncated in saved text. Resubmit with corrected text.
6	Too many symbols (names and constants) in text. Fatal error. Resubmit with text partitioned into more START-TERM sequences.
7	Module nesting exceeds limit. Change module nesting structure.
8	Too many ENDs. Resubmit with corrected text.
9	Loop in basic element analysis. System error.*
10	Loop in internal text character analysis. System error.*
11	Loop in JOVIAL element analysis. System error.*
12	Loop in external character analysis. System error.*

* System errors should be reported with output listing card images processed.

B-4 to B-10 *

Error Message	Corrective Action
Nested switch call treated as fall through case	Replace switch invocation in switch declaration by a label and rerun BASIC and STRUCTURAL
Never found PROC statement	JOVIAL procedure must contain PROC statement. Rerun BASIC and STRUCTURAL
Pointer LQ 0	Rerun STRUCTURAL
Possible infinite loop detected	Rewrite module to remove infinite loop. Rerun BASIC and STRUCTURAL
Stack overflow	Increase the stack size by recompiling STRUCTURAL. Rerun STRUCTURAL.
Stack underflow	Rerun BASIC and STRUCTURAL
Switch name missing	Check SWITCH syntax and rerun BASIC and STRUCTURAL
Test stack overflow	Increase the size of the test stack by recompiling STRUCTURAL. Rerun STRUCTURAL
Too many paths begin at statement	Increase the size of the DD-path queue by recompiling STRUCTURAL and rerun STRUCTURAL. Or modify module to have fewer than 100 switch points in the switch declaration and rerun BASIC and STRUCTURAL.
Too many statements on DD-path	Increase the size of the DD-path statement list by recompiling STRUCTURAL and rerun STRUCTURAL. Or modify module to contain a dummy decision to reduce the number of statements on a DD-path to fewer than 100 and rerun BASIC and STRUCTURAL.
Undefined GOTO	Check location of label referenced in GOTO. Rerun BASIC and STRUCTURAL

B.4 INSTRUMENT ERROR MESSAGES

Error Message	Corrective Action
Bad module number	Specify a module number within the library. Rerun INSTRUMENT
BEGIN-END block contains no statements	Informative message
Could not match parentheses	Check syntax of statement. Rerun BASIC, STRUCTURAL, and INSTRUMENT
Delimiter stack overflow	Increase the size of the delimiter stack by recompiling INSTRUMENT. Rerun INSTRUMENT.
Directive not recognized	Check syntax of directive. Rerun BASIC, STRUCTURAL and INSTRUMENT
External switch call not instrumented	Declare switch in module. Rerun BASIC, STRUCTURAL, and INSTRUMENT
GOTO label not found	Check syntax of GOTO statement. Rerun BASIC, STRUCTURAL, and INSTRUMENT
Input string too long--truncated from head	Change size of internal buffers by recompiling INSTRUMENT. Rerun INSTRUMENT
Invalid statement type in chain	Check syntax of IFEITH statement. Rerun BASIC, STRUCTURAL, and INSTRUMENT
Invalid switch type	Check syntax of switch declaration. Rerun BASIC, STRUCTURAL, and INSTRUMENT
Last statement in module is not TERM or END	Add a TERM or END statement to the module and rerun BASIC, STRUCTURAL and INSTRUMENT
No DD-path for statement	Rerun STRUCTURAL and INSTRUMENT
Number of switch labels <= 0	Rerun STRUCTURAL and INSTRUMENT
ORIF statement number not on false branch stack	Rerun INSTRUMENT
Pointer out of range	Rerun STRUCTURAL and INSTRUMENT

REFERENCES

1. C. Gannon, N. B. Brooks, JAVS Technical Report: Vol. 1, "User's Guide," General Research Corporation CR-1-722/1, June 1978.

2. C. Gannon, N. B. Brooks, JAVS Technical Report: Vol. 2, "Reference Manual," General Research Corporation CR-1-722/1, June 1978.

3. E. F. Miller, Jr., Methodology for Comprehensive Testing, General Research Corporation CR-1-465, June 1975.

4. N. B. Brooks, C. Gannon, JAVS Technical Report: Vol. 3, "Methodology Report," General Research Corporation CR-1-722, December 1976.

5. J. P. Benson et al., JAVS Computer Program Documentation, Vol. 1, "System Design and Implementation Manual," General Research Corporation CR-1-782, June 1978.

6. R. A. Melton, D. M. Andrews, FAVS Fortran Automated Verification System User's Manual, General Research Corporation CR-1-754, December 1977.

7. D. M. Andrews, J. P. Benson, Advanced Software Quality Assurance Software Quality Laboratory User's Manual, General Research Corporation CR-4-770, May 1978.

MISSION
of
Rome Air Development Center

RADC plans and conducts research, exploratory and advanced development programs in command, control, and communications (C^3) activities, and in the C^3 areas of information sciences and intelligence. 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.