

②

**TRANSL--THE PLANIT TRANSLATOR PROGRAM:
INSTALLATION AND APPLICATION**

AD A 076812

Charles H. Frye
The Northwest Regional Educational Laboratory

DDC
NOV 19 1978
RESERVE

Educational Technology and Training Simulation Technical Area



This document has been approved
for public release and sale; its
distribution is unlimited.

DDC FILE COPY

U. S. Army
THIS DOCUMENT IS BEST QUALITY PRACTICABLE.
THE COPY FURNISHED TO DDC CONTAINED A
SIGNIFICANT NUMBER OF PAGES WHICH DO NOT
REPRODUCE LEGIBLY.

Research Institute for the Behavioral and Social Sciences

May 1976

79 11 13 299

DISPOSITION FORM

For use of this form, see AR 340-13, the proponent agency is TAGCEN.

REFERENCE OR OFFICE SYMBOL

SUBJECT

PERI-TP

Clearance and Transmittal of Reports to DTIC

TO DDC-DAA-1
ATTN: Mr. Schrecengost

FROM ARI Rsch Pub Group

DATE 8 Nov 79

CM

Ms Price/48913

*for the books...
See 20*

1. The reports listed on Inclosure 1 are approved for public release with unlimited distribution (50 numbered ARI Research Memorandums, 74-1 thru 76-30).
2. These are among the previously unrecorded ARI reports which you identified to us 22 June 1979 as not in your retrieval system. The accompanying box contains at least one copy of each report for your retention and reproduction.

1 incl
List of reports, 1974-76

Helon S. Price

HELEN S. PRICE
Research Publications Group
Army Research Institute

x 9

Acquisition For	
Miss Gaski	<input checked="" type="checkbox"/>
DDC TAB	<input type="checkbox"/>
Enhanced Justification	<input type="checkbox"/>
By _____	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A	

14 ARI-RM-76-10

16 Army Project Number
20762717A721

Educational Technology and
Training Simulation
Contract Number DAHC 19-74-C-0038

9 Research Memorandum 76-10

6 TRANSL--THE PLANIT TRANSLATOR PROGRAM: INSTALLATION AND APPLICATION.

10 Charles H. Frye
~~Regional Educational Laboratory~~
~~Submitted by~~
James D. Baker Chief

The Northwest Regional Educational Laboratory

Educational Technology and Training Simulation Technical Area

11 May 1976

12 78

Approved by:

Joseph Zeidner, Director
Organizations and Systems Research
Laboratory

J. E. Uhlner, Technical Director
U.S. Army Research Institute for
the Behavioral and Social Sciences

Research Memorandums are informal reports on technical research problems.
Limited distribution is made, primarily to personnel engaged in research for the
Army Research Institute.

408010 Gu

* (Programming language for interactive teaching)

** AN/GYK-12

ABSTRACT

→ This document contains full information for installing and operating a program which is designed to translate the FORTRAN from the PLANIT system of programs into the TACPOL language for compilation on the ANGYK-12 computer.

PLANIT is an interactive training system which is now operational on the ANGYK-12, having been installed via the program herein described. While the normal method for installing the PLANIT system requires the existence of a FORTRAN compiler on the target computer, this was circumvented for the ANGYK-12 installation by developing the Translator (TRANSL) program to first translate the FORTRAN into TACPOL for use with the PSS-B compiler, the only compiler on the ANGYK-12.

** Although the installation of PLANIT is complete on the ANGYK-12 computer, future updates of PLANIT may necessitate additional use for the TRANSL program.

↖

BACKGROUND OF THE PLANIT USER TRAINING SYSTEM

Several explicit user requirements converged to generate the research which resulted in the documents contained in this set of reports. The need for some type of user training subsystem in support of tactical automatic data processing (ADP) system developments was clearly established during the evolutionary phase of the Army Tactical Operations System (TOS) development in Europe.¹ In 1974, after a decade of involvement in the development of tactical ADP systems, the Army Computer Systems Command summarized this experience into six "Lessons Learned."² One of these lessons was: A dedicated and trained user is required if tactical ADPS is to succeed.

One approach toward meeting this requirement is to apply techniques derived from modern educational technology and the computer sciences by embedding training subsystem packages within the operating system and then using the system itself to teach the user how to use the system. The approach was delineated in a concept paper,³ which was subsequently submitted, evaluated and found by key Army Personnel to have merit. As a consequence, a requirement was placed on the Army's Behavior and Systems Research Laboratory (BESRL--the predecessor of what is now the Army Research Institute) by what was then the Assistant Chief of Staff for Force Development (ACSFOR) and the Director of Army Research, Office of the Chief of Research and Development (OCRD),^{4,5} to effectuate the research necessary to test the concept.

¹Baker, J. D. "Human Factors Experimentation Within a Tactical Operations System (TOS) Environment." Proceedings: Office of Naval Research Sponsored Tri-Service Coordination Meeting, London, England, 20-21 February 1968.

²Memorandum from Headquarters, U.S. Army Computer Systems Command to Assistant Deputy Commander, CACDA, Ft. Leavenworth, KA; Deputy Commander, MASSTER, Fort Hood, TX; Project Manager, Army Tactical Data Systems, Fort Monmouth, NJ, dtd 30 January 1974, Subject: TSDG Lessons Learned.

³Memorandum from U.S. Army Behavior and Systems Research Laboratory to Assistant Chief of Staff for Force Development, dated 28 September 1971, Subject: Proficiency Maintenance Using Computer-Assisted Instruction in an Operational Setting.

⁴Memorandum from Assistant Chief of Staff for Force Development to Chief of Research and Development, dated 10 November 1971; with 18 November 1971 indorsement to Behavior and Systems Research Laboratory, Subject: Request for Research in Application of Tactical Data Systems for Training.

⁵Memorandum from Chief of Research and Development to Assistant Chief of Staff for Force Development, dated 29 Nov 1971, Subject: Request for Research in Application of Tactical Data Systems for Training.

The terms of the requirement actually levied, however, went well beyond the scope of the original concept and called for a simultaneous attack on all facets of the problem associated with testing the feasibility of the approach. In terms of broadened scope, the primary role of these systems is in support of tactical operations. Our original concept paper suggested a potential, select secondary role for these computerized tactical data systems, viz., that of directly supporting the system user by using the system itself, in a stand-alone mode, to teach the user how to use the system. The agencies structuring the research requirements saw a possible tertiary role for these systems. About the time they were structuring their requirements, the Army's Dynamic Training Board identified the maintenance of proficiency of Military Occupation Specialty (MOS) 11B40, the light weapons infantryman, as a glaring unit training problem and suggested that Computer-Assisted Instruction (CAI) as one technique for alleviating the situation.⁶ In addition, a subsequent Continental Army Command (CONARC) Task Group report on CAI identified the 11B40 MOS as a top contender for attention in the "non-technical" skills area.⁷ Consequently, the scope of the effort was expanded to encompass an examination of a tertiary role, i.e., in support of the system's parent unit by using these computers to meet individual and unit training requirements such as those associated with the 11B40 MOS. Additionally, in response to concern that the implementation of the Modern Volunteer Army concept might produce a need for general education development (GED) upgrading it was determined that an examination should be made of the feasibility of employing extant CAI GED on tactical computers in an operational setting. The assumption was made that accomplishment of these latter requirements would be tantamount to proving the feasibility of the secondary role concept as well. The test, therefore, would be a cost-effective undertaking since it would provide data directed toward answering a number of diverse questions concerned with a common training delivery system, viz., tactical computers.

Irrespective of whether it was the secondary or tertiary role concept being assessed, four major components were required: a test in a credible operational environment; appropriate hardware; functioning software and representative people-ware. The vehicle for this overall assessment was MASSTER⁸ Test FM 122, "IBCS: Automated Instruction." The hardware was a "given" viz., the Developmental Tactical Operations

⁶Report of the Board for Dynamic Training, Volume II. 17 December 1971, page 116.

⁷Headquarters, United States Continental Army Command Task Group Report and Computer Assisted Instruction. April 1972.

⁸MASSTER - Modern Army Selected Systems Test, Evaluation, and Review--is the Army's test bed for assessing equipment, concepts and doctrine. This activity is located at Fort Hood, Texas.

System (DEVLOS) which was then located at Fort Hood, Texas (Hoyt, et al⁹ provide a description of the hardware). Likewise, the people were a "given"--our student population would be MOS 11B40 personnel drawn from the 2nd Armored Division and 1st Cavalry Division located at Fort Hood. The question of what "software" approach to take (specifically, whether to use an existing student/author language) was key to the success or failure of Test 122. Clearly, the decision made at this juncture would determine whether we would hit the assigned "test window" in time to conduct the test. As a related issue, courseware development would largely depend upon the structure of the student/author language selected, so courseware development could not commence until this decision was made. The decision itself had to be correct and timely--and whatever decision was made would undoubtedly be risky.

To add to the difficulty in reaching a decision, it must be realized that it could not be made unilaterally. Conduct of a test of the complexity of MASSTER Test FM 122 required support from and coordination between a number of different agencies--key among them being mutual cooperation of the organization which had DEVLOS responsibility, the U.S. Army Computer Systems Command (USACSC), and the Army Research Institute (ARI). A Memorandum of Understanding¹⁰ was drawn up between these two organizations and, as the first USACSC task in this joint undertaking, a MASSTER Test 122 CAI Concept Paper¹¹ was to provide alternative concepts for implementing automated instruction materials on the DEVLOS in support of MASSTER Test 122. Concurrent with this effort, a contract was let by ARI with the System Development Corporation (SDC) to develop the courseware (i.e., the instructional materials which would be presented through CAI). The first task SDC had to accomplish was to provide alternative student/author language alternatives for generating the courseware and to determine which alternative provided the best likelihood of success under the test conditions and time constraints imposed. In essence, the combined results of these analytic studies were expressed as follows: "At this stage, many alternative design concepts can be formulated. However, due to time constraints on the implementation of any concept, the only alternative concept considered feasible...is the use of PLANIT."¹²

⁹Hoyt, W. G., Butler, A. K. and Bennik, F. D. "Application of Tactical Data Systems for Training: DEVLOS Feasibility Determination and Selection of an Instructional Operating System." ARI Technical Paper 267, October 1975.

¹⁰Memorandum of Understanding Between Commander, U.S. Army Research Institute and Commander, U.S. Army Computer Systems Command, Dated 5 June 1973.

¹¹Bunker-Ramo Technical Note "MASSTER Test 122--Computer Assisted Instruction (CAI) Concept Paper," February 1975, prepared for the U.S. Army Computer Systems Command.

¹²Ibid. 11, page 18.

PLANIT (Programming Language for Interactive Teaching) is an instructional system consisting of an author language and supporting computer programs for preparing, editing and presenting any subject matter suitable for individualized CAI presentation to students, as well as recording all relevant response data for immediate utilization and subsequent analyses. PLANIT was developed over an eleven year period under the aegis of the National Science Foundation (NSF) at a total investment cost of approximately \$740,000. The main goal of this NSF project was to produce a student/author language which would be fully transportable and guaranteed compatible with a large and diversified class of machines.¹³ We at ARI take professional pride in the fact that it was our early and subsequent work with PLANIT which validated this visionary transportability notion of NSF.¹⁴ We also take "economic" pride in the fact that we capitalized upon an already "hefty" U.S. Government investment to solve a problem, rather than slipping into the classic mold of "reinventing the wheel" by starting from scratch and building a separate student/author language tailored to the hardware/software system constraints.

To lower the curtain on MASSTER Test FM 122, the test was successfully conducted and demonstrated that it was feasible to use tactical computers in a stand-alone training mode to satisfy individual and unit training requirements. It was found that automated instruction in a field setting was enthusiastically accepted by the non-commissioned officers (NCO's) examined and, as a training medium, it proved to be more effective than the traditional study-method of training.^{15,16,17,18,19}

¹³Frye, C. H. "A Report on PLANIT: One Stage of Completion," Final Report for the National Science Foundation Grant No. EPP73-07319 A04, August 1975.

¹⁴For a complete account of the experiences of ARI in installing, using and evaluating PLANIT in an Army setting, including all the "warts and blemishes" uncovered during this endeavor, see: Johnson, C. "Implementation of PLANIT at the U.S. Army Research Institute for the Behavioral and Social Sciences," PLANIT Newsletter, July 1975.

¹⁵Hoyt, W. G. and Baker, J. D. The use of tactical computers to provide weapons and tactics training to combat NCO's: Results of a field test. Proceedings: Sixteenth Annual Conference of the Military Testing Association MTA, U.S. Coast Guard Institute, Oklahoma City, OK. 21-25 October 1974.

¹⁶Hoyt, W. G., Butler, A. K. and Bennik, F. D. Application of tactical data systems for training: Volume II - CAI/DEVLOS automation studies. ARI Technical Paper 267, October 1975.

¹⁷Hoyt, W. G., Butler, A. K. and Bennik, F. D. Application of tactical data systems for training: Volume I - Executive Summary. ARI Technical Paper ____, in preparation.

But the results of this test proved more than the preceding. They also indicated that the obvious Army needs mentioned at the outset of this preface, could be met by applying this technology to a real and present problem. It also went beyond the exploratory stage and satisfied a specific Army requirement. The U.S. Army Combat Developments Command (CDC)/Systems Analysis Group (now the U.S. Army Training and Doctrine Command/Combined Arms Combat Developments Activity, or TRADOC/CACDA) had levied the following requirement²⁰ on ARI:

The Proposed Material Need for the Tactical Operations System - TOS (Unclassified title, portions of contents classified CONFIDENTIAL) states: "During system non-tactical employment the equipment shall have the capability to permit the training of user personnel without affecting the mission ready capability of the system." While the need exists, no specific data are extant which can be brought to bear on this problem. The requested research will provide data which could impact on all TOS users and result in considerable savings in training costs related to the user's need to maintain proficiency in the use of these systems.

The 122 Test data satisfied the CDC requirement. The Proposed Material Need (MN) for TOS was found to be a viable concept and that MN remains to this day as a bonafide component of the TOS program.

As previously discussed, the results from MASSTER Test FM 122 demonstrated the viability of the embedded training subsystem concept in general and that tactical data systems could be used in a tertiary role, i.e., specifically, that these systems could be used in a stand-alone mode in support of individual and unit softskills training requirements. But conceptually our main goal had always been to embed system specific training packages within the operating system itself and then to use the system to teach the user how to use the system--the earlier noted secondary role for these systems.

-
- ¹⁸Hoyt, W. G., Butler, A. K. and Bennik, F. D. Application of tactical data systems for training: Volume III - Development of courseware and analysis of results for MOS 11B40. ARI Technical Paper ____, in preparation.
- ¹⁹Hoyt, E. G., Butler, A. K. and Bennik, F. D. Application of tactical data systems for training: Volume IV - Development of courseware and analysis of results for GED math. ARI Technical Paper ____, in preparation.
- ²⁰Letter, DARB-ARB 19 July 1972, Subject: New Research Requirements for the Human Resources Research and Development Program (RCS CSCRD 70 CRI); letter response from CDCSAG-AG1, same subject as above, dated 1 September 1972.

As a follow-on to Test 122, research was initiated under the aegis of the Product Manager, Computer Training Systems (PM CTS) through HRN 75-158 (and, subsequently, HRN 76-195) which tasked ARI to address the problem of reducing the novice user's difficulties by making tactical data systems (e.g., TOS², TACFIRE, TSQ-73, etc.) more "approachable" through applications of the embedded training concept.²¹

Because of its stage of development, the fact that its basic central processing unit would serve as the core for other Army Tactical Data Systems (ARTADS) to follow, and the fact that its operator training problems appeared to be amenable to reduction through the application of automated instructional technology, TACFIRE (the Army's field artillery tactical fire control system) was chosen by the PM CTS as the test vehicle for assessing the embedded training subsystems concept. The initial and specific requirements for the TACFIRE research were delineated in HRN 76-193, "Development and Evaluation of PLANIT Based Computer Embedded Training Packages for TACFIRE" which was prepared by personnel of the U.S. Army Field Artillery School, Fort Sill, OK.

Once again we were faced with the dilemma as to whether the best decision would be to develop a tailor-made student/author language smoothly fitted to the hardware/software constraints of the TACFIRE system, or to build upon our already successfully operating PLANIT system and attempt to install it on TACFIRE. The latter approach had many merits, among them: (1) it was an author language system with which we were familiar, while a customized system would be untested, costly and would require an extensive checkout; (2) a customized authoring system would be limited to a given TACFIRE configuration, whereas PLANIT would be transportable to the family of ARTADS systems, and (3) because of PLANIT's machine independent characteristics, courseware could be prepared on commercial computers and, after content checkout, easily installed on the tactical system, whereas a customized approach would tie-up the actual tactical system during courseware preparation.

The effort to install PLANIT on the AN/GYK-12 computer, the results of which are contained in this set of reports, was independently undertaken as Technology Based - Exploratory Development research and not as Advanced Development activity (i.e., it was not done in direct response to an explicit, stated user need). It serves as a classic example of what Dr. Malcolm R. Currie, Director of Defense Research and Engineering (DDR&E) was describing in the following statement to the Second Session of the 94th Congress: "The objective of the Technology Base is the advancement of technology applicable to future systems and subsystem

²¹Human Resource Need (HRN) 75-158, title: "User Training and Proficiency Maintenance in a Tactical Data Systems Environment," submitted as a research requirement for inclusion in the ARI FY 75 Advanced Development Work Program by the Product Manager, Computerized Training System, Fort Monmouth, NJ. HRN 76-195 was a revalidation of the requirements delineated in 75-158 for inclusion in the FY 76 Work Program.

options. These options (or new ideas) usually involve enhanced military capability, reduced cost, increased performance, better reliability and maintainability, more efficient use of resources or some combination of these attributes." Success in this effort would produce a broadly applicable, cost-effective vehicle for employing embedded training subsystem packages in a variety of military system settings.

It merits comment, however, that while this work was a Technology Based-Exploratory Effort, it had the potential for feeding into the Advanced Development program efforts associated with the user tasks presented in HRN 75-158, "User Training and Proficiency Maintenance in a Tactical Data Systems Environment," if the outcome were successful. Consequently, the PM-CTS was appraised of this effort at the outset and he, in turn, coordinated it with the Program Manager, Army Tactical Data Systems (PM ARTADS). During this coordination some valid points of criticism were raised²² concerning the PLANIT approach. The PM ARTADS recommended that ARI meet with system developers, users and training agencies as soon as sufficient data were available to determine whether, or not, PLANIT would operate on TACFIRE. At that time a determination would be made concerning implementation implications and to assess if, indeed, this were the most effective approach to take, given the potential for impact on TACFIRE system development efforts. In keeping with this recommendation, a Workshop was convened at ARI in Arlington, VA on 1 October 1974 and these items were covered in detail with personnel from all of the suggested groups in attendance. The interaction was found to be most beneficial to all concerned and the consensus of the group was to install the system described in this set of reports on the TACFIRE system at Fort Sill, OK, and to use it as the test vehicle for assessing the embedded training concept on that ARTADS system.

This historic overview of the events leading up to the production of the set of quite specialized reports may seem untoward in view of the projected, limited set of users of these documents. It is, however, a quite meaningful forum for discussing these events. Too frequently the question is raised as to how did a particular research product originate and was it utilized. The intent here is to show that the warp and woof of concepts and coordination, requirements and research are so intertwined that a simple one-to-one relationship (one response, one use) does not tell the story--only a view of the whole cloth will put it into proper perspective. Additionally, it exemplifies a point made in the previously cited presentation by the Director of Defense Research and Engineering to the 94th Congress when he said: "To deploy systems DOD must not only pursue advanced technology but must endure the long years of research required to bring an idea through growth problems to a finished, proven and useful end product."

²²Memorandum from Product Manager, Computer Training Systems (PM-CTS) to Program Manager, Army Tactical Data Systems (PM-ARTADS) 28 Jan 74, Subject: HRN 75-158 and 1st indorsement from PM-ARTADS to PM-CTS, same subject as above dated 7 February 74.

This set of reports provides detailed instructions for implementation and operation of PLANIT and auxiliary programs on the AN/GYK-12 computer. The set consists of a report on:

TRANSL - The PLANIT Translator Program: Installation and Application

PLANIT Support Programs - Operator/user manual

PLANIT Utility Program - Operator/user manual

PLANIT Support and Utility Programs - Test Procedure

PLANIT Support and Utility Programs - Flow Charts.

The first report contains the information for installing and operating a program which is designed to translate the FORTRAN from the PLANIT system of programs into the TACPOL language for compilation on the AN/GYK-12 computer. The second covers the general and specific aspects of leading and operating PLANIT on the AN/GYK-12 computer. The third document covers the general and specific aspects of operating the PLANIT utility programs which are a specialized group of routines developed to accomplish various tasks in support of the AN/GYK-12 computer installation of PLANIT. The fourth report covers the procedures used to verify that PLANIT Support and Utility Programs are functioning as per specifications. The fifth document provides the detailed flow charts of the computer logic of the PLANIT Support and Utility Programs.

The effort detailed in the first report (i.e., TRANSL) was accomplished under ARI Contract DAHCl9-74-C-0038 by the Northwest Regional Educational Laboratory, Portland, Oregon. The other four reports in the series were prepared by the Data Systems Division, Litton Systems Inc., Van Nuys, CA under ARI Contract No. DAHCl9-74-C-0064.

CONTENTS

	<u>Page</u>
INTRODUCTION	1
INSTALLATION OF TRANSL	6
APPLICATION OF TRANSL	10
OPERATION OF TRANSL	12
FINAL INSTALLATION CONSIDERATIONS	18

FIGURES

Figure 1. Typical PLANIT Installation Process	3
Figure 2. PLANIT Installation Process For The ANGYK-12	4
Figure 3. Example of LDBYTE and SBYTE Operation	9

APPENDICES

Appendix A. Translation Rules	
Appendix B. TRANSL Initialization Deck	
Appendix C. PLANIT Merge File Listing for Batchstream to the TRANSL Program	
Appendix D. Program Listing of TRANSL	

TRANSL -- THE PLANIT TRANSLATOR PROGRAM

INSTALLATION AND APPLICATION

INTRODUCTION

Soon after computers began to be used through time-shared, interactive typewriter terminals, experimental efforts were initiated to use this new facility for training purposes. Within a few years several systems were developed which made the computer more readily adaptable for this use. PLANIT (Programming Language for Interactive Teaching) is one such system. The user features and intended applications of PLANIT are documented elsewhere and will not be elaborated here. What is important is the fact that, unlike other such systems, PLANIT is a complete and fully portable time-sharing system suited particularly to training applications. These features have contributed to the selection of PLANIT for installation on a variety of military computers, some of which have non-standard features and lack the usual software support on which some interactive systems depend.

One such specialized military computer is the ANGYK-12 (or the Litton designation, L3050) which is designed to be used in several tactical systems applications such as the TACFIRE fire control system. Although the ANGYK-12 is a versatile and sophisticated computer, it lacks many of the facilities which one would expect to find on a commercial computer of comparable size. Instead of the usual menu of programming languages, it has only TACPOL (a PL-1 like language). The TACPOL language was designed especially for this computer. Although the similarities to conventional programming languages are many, there are some important differences such as the lack of any floating point numerical operations.

Because of the above considerations, PLANIT was found to be the only available training system which could be installed on the ANGYK-12 computer without extensive re-development efforts. However, PLANIT normally makes use of the FORTRAN IV language in the installation process to achieve its portability. Since FORTRAN IV was not available on the ANGYK-12, a special translator program was written which intercepts the FORTRAN statements in the usual PLANIT installation process and converts them to equivalent TACPOL statements.

It should be noted that the term, FORTRAN-to-TACPOL Translator is not being used because that has the connotation of a much more general program than the one needed for this particular task. Called TRANSL, this translator program is designed explicitly to translate the programs associated with the PLANIT system and particularly the PLANIT system FORTRAN statements.

Portability of the PLANIT system was achieved by writing the system code in a unique, machine-independent higher level language. Although this language bears some similarity to FORTRAN, it is necessary that each of the several thousand statements be preprocessed by a system generation program before the result is submitted to a FORTRAN compiler. In system generation, machine and configuration parameters are introduced into the code in such a way that the resulting FORTRAN statements will then be suitable for the intended machine. Therefore TRANSL will only be translating FORTRAN statements of the type that the PLANIT system generator produces. This reduces the magnitude of the translation task significantly and makes possible the guarantee of 100% success in the translation of these FORTRAN statements into TACPOL.

The PLANIT System Generator program is written in the FORTRAN IV language in such a way that it, too, can be installed easily anywhere that a FORTRAN IV compiler exists. Because of adherence to strict programming standards, the Generator can also be translated into TACPOL by using the TRANSL program. Expected translation failures will occur in four types of statements: READ, WRITE, FORMAT and END FILE. These statements are few and are conveniently clustered at the beginning of the program for ease in changing by hand.

The installation process for PLANIT and the PLANIT System Generator is documented in detail elsewhere. Figure 1 shows the general sequence to be followed in the normal installation of PLANIT. Note that the production of FORTRAN is simply an intermediate step in the installation process and has no further function after the program has been compiled. By the time PLANIT is running, all connections to FORTRAN are gone. Figure 2 shows the TACPOL translation to be simply an inserted step in the installation sequence. Note that there is no change at the top, i.e. the manner in which the system generation is done. Also note that, like FORTRAN, all connections to TACPOL disappear when PLANIT is ready to run. Note, too, that both installation methods require the writing of a similar locally designed interface program (MIOP). Therefore, the use of the TRANSL program is just another step in the same general process.

PLANIT INSTALLATION PROCESS

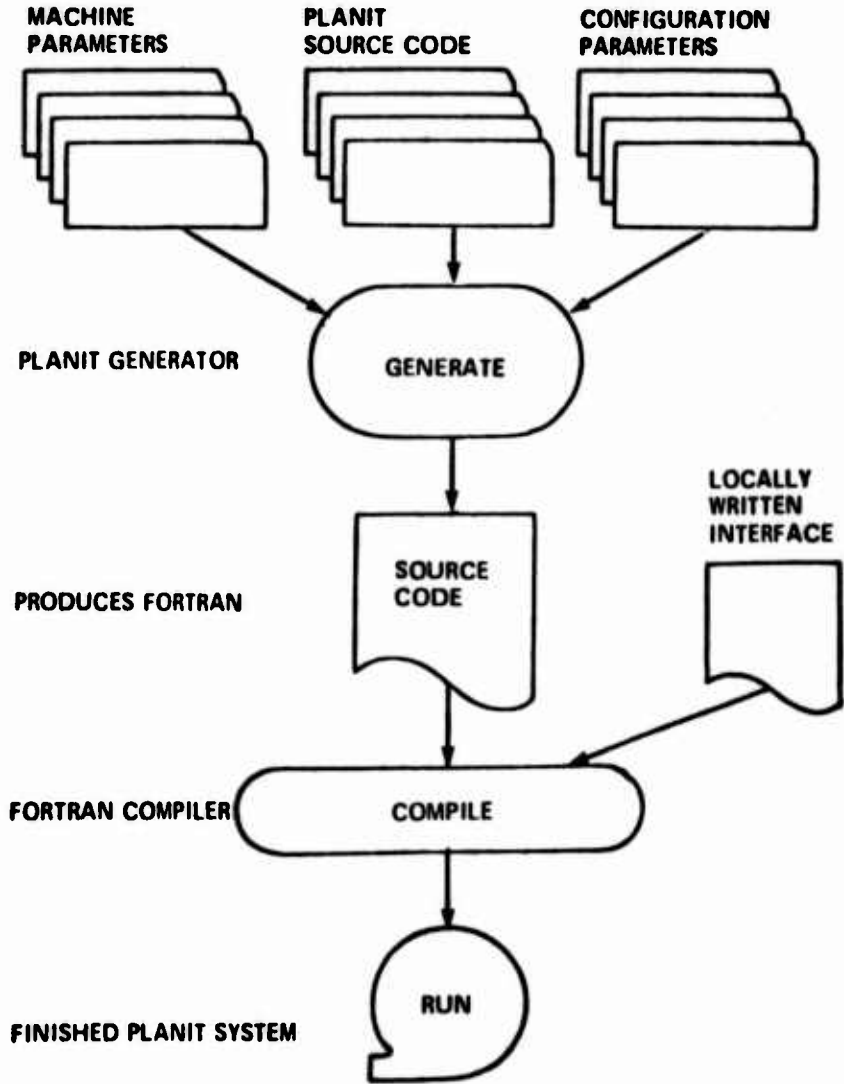


Figure 1. A typical sequence of steps for installing the PLANIT system on a commercial computer. The three top files plus the PLANIT Generator are furnished. Installers modify the parameters to fit local needs prior to system generation. A local FORTRAN IV compiler is required.

PLANIT INSTALLATION PROCESS

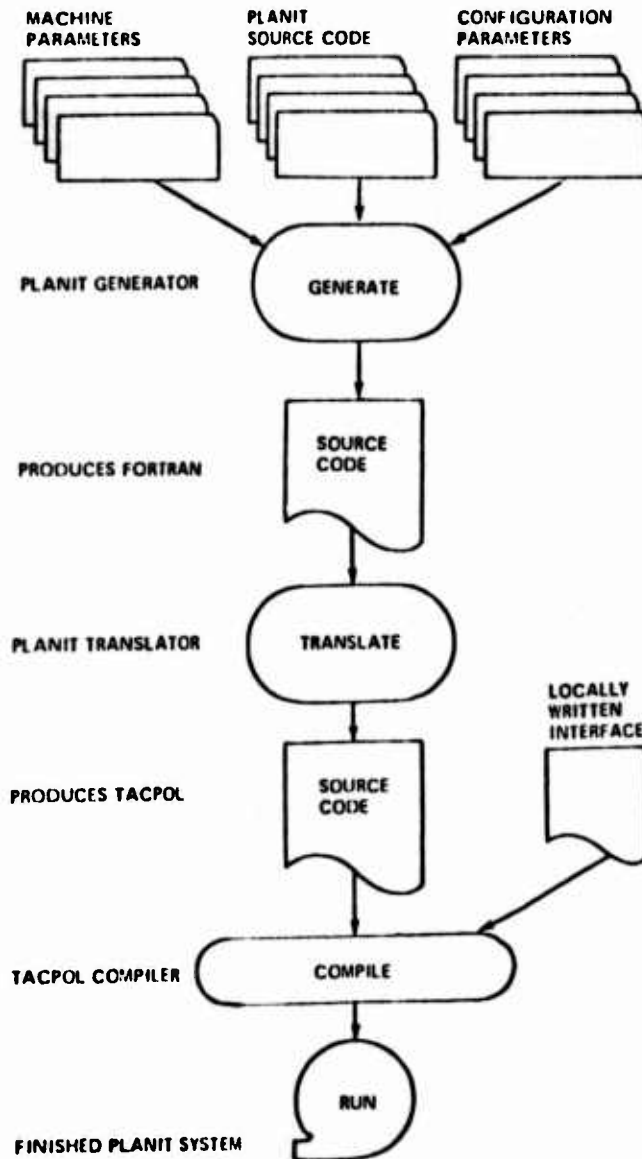


Figure 2. The sequence of steps for installing PLANIT on the ANGYK-12 computer. The three top files plus the PLANIT Generator and the PLANIT Translator are furnished. Parameters have already been set appropriately. A local TACPOL compiler is required.

Finally, for convenience, TRANSL was coded in FORTRAN IV according to the same standards which were followed in the coding of the PLANIT System Generator program. This makes the TRANSL program, like the Generator program, completely portable even though it was written specifically for the ANGYK-12 implementation of PLANIT. The coding of TRANSL in this fashion has two advantages: 1) it could be modified for use in another specialized application at a considerable saving, and 2) the procedures for installing and running TRANSL are nearly identical to those for the Generator. Thus, for the reader who is already familiar with procedures for installing the PLANIT System Generator, the only differences will be that TRANSL uses only one input file instead of two and writes to one output file but not to the line printer (as does the Generator program). Otherwise, the commented section in the listing of TRANSL which describes the few changes which may be necessary is very similar to the Generator and the requirements for the LDBYTE and SBYTE subroutines are identical for both programs (as well as for PLANIT itself). Thus TRANSL can be used to translate itself into TACPOL if such a version is desirable.

INSTALLATION OF TRANSL

The TRANSL program will require one input file and one output file. Both files will be 80-column card images on disk, tape, or whichever physical device is desired. The device assignment is made in the usual FORTRAN manner. The input file is assigned the FORTRAN logical file number one and the output file, number two. These assignments are made in a FORTRAN READ and WRITE statement, respectively, near the top of the program. There is only one of each.

The input file will contain card images of FORTRAN statements and the output file will contain the equivalent TACPOL statements. Diagnostics, if any, will be inserted appropriately in the output file and a count of such diagnostics (if greater than zero) will be added to the end of the file. The content and format of the diagnostics will be discussed later but the user will probably want to plan on obtaining a listing of the output file before going to the next step during actual installation.

The TRANSL program listing begins with several lines of comments which summarize the information presented here. Especially important in these comments are the few lines enclosed between two rows of asterisks (*****). Between these lines are grouped all of the FORTRAN statements which should normally be examined during the installation of the TRANSL program.

The first statement to be considered is "IBYTE=4". This statement signifies that there are four bytes (characters) to the computer word in the design of the machine on which this program is being run. That number is correct for the ANGYK-12 computer as well as for many others. If it is incorrect it must be changed.

If there is a need to open either of the files, that should be done just before or just after the IBYTE=4 statement.

The "END FILE 2" statement causes the writing of the end-of-file mark on the output file and it is closed. This statement may need changing to conform to local requirements. File 1 may also be closed at this point if necessary.

The READ and WRITE statements are the last to be checked in this section. The card image is passed through a common buffer, INBUFF. Both the READ and WRITE statements operate with implicit DO-loops to specify exactly one card of 80 columns at a time. The FORMAT card at label 1000 (above) must agree with the READ and WRITE statements

such that exactly one card image will be read (or written) at a time.

In rare instances a FORTRAN compiler may require the dimension of INBUFF to exactly fit one card. If so, then the dimension of the array INQUE must also be adjusted to the new number and a statement below the asterisk line which now reads "IBFDIM=30" must be changed so that the new dimension number replaces the 30. With these changes, the implicit DO-loops could also be eliminated if the compiler objects to them.

In summary, the changes to be expected are in statements which are located between the two lines of asterisks, and these changes will probably be limited to the two statements, "IBYTE=4" and "1000 FORMAT(20A4)" where the number of bytes per computer word differ from four. If the bytes per word equal four, the Translator program should run just as it is.

The final installation requirement of the TRANSL program is the writing of the LDBYTE and SBYTE subroutines. These two subroutines are called throughout TRANSL as well as throughout the Generator and PLANIT. The calls are the same in all cases--"CALL LDBYTE" or "CALL SBYTE"--with no argument string in the calling sequence. The parameters for doing the prescribed work are taken from COMMON in the manner described below. Note that the description will be the same for these subroutines used by the Generator and PLANIT.

Subroutines LDBYTE and SBYTE both use the same set of three parameters, called IBYT1, IBYT2 and INDEX. The first entry in COMMON shows the locations of the three parameters. Suppose an array, IO, was equivalenced to COMMON. Then IO(IO(1)-3) would be the same as IBYT1, IO(IO(1)-2) would be the same as IBYT2, and IO(IO(1)-1) as INDEX. In other words, the first entry in COMMON contains the "index" of the COMMON location just beyond INDEX. Thus, LDBYTE and SBYTE must use the same COMMON as TRANSL, and the first entry in COMMON will locate the three parameters.

The interpretation of each of the three parameters is as follows:

- IBYT1 - The byte number of the first character in a string of contiguously packed characters in COMMON.
- IBYT2 - The byte number of the last character in the string described above.
- INDEX - The COMMON "index" number to which (or from which) the characters will be unpacked (packed)

Figure 3 illustrates the operation of LDBYTE and SBYTE with the byte numbering and indexing scheme. The bytes (character positions) in COMMON assume sequential numbers such that if COMMON was completely filled with one long string of characters tightly packed with no gaps, then the first character would be in position one (leftmost byte of the first word in COMMON) and the numbers would increase sequentially throughout COMMON. The index, on the other hand, refers to the word number of COMMON where the first word has the number one assigned to it. If COMMON was treated as one long array where each word was an entry in the array, then the index would be equivalent to the entry number, beginning at one.

The subroutine LDBYTE is to unpack characters beginning at byte number IBYT1 through byte number IBYT2, placing a copy of the unpacked characters, one byte per word, back into COMMON beginning at INDEX. The unpacked characters are to be right-justified in the word with leading binary zeros. The only COMMON locations which are to be changed (including the parameters themselves) are the words which receive the unpacked characters.

The subroutine SBYTE does the reverse operation from LDBYTE. It must make a string of the rightmost character of each successive word entry beginning at INDEX and pack that string into COMMON beginning at IBYT1 byte position and ending at IBYT2 byte position without changing byte locations on either end of the newly packed string or elsewhere in COMMON.

Figure 3 assumes a computer with four bytes per word and shows an example where:

```

IBYT1-22
IBYT2-27
INDEX-41

```

The repositioning of the copied characters which each subroutine must perform is indicated by the arrows.

The LDBYTE and SBYTE subroutines may be written in any form that is most convenient on the target computer. However, if the same ones are to be used for PLANIT, then care should be taken to insure fast and efficient operation.

LDBYTE

SBYTE

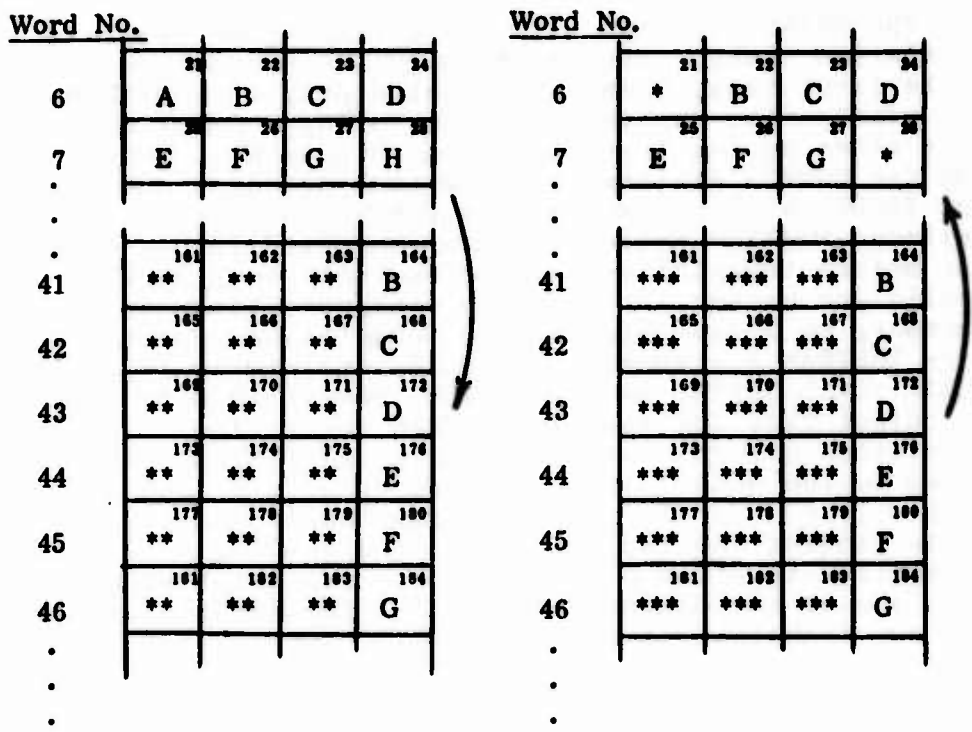


Figure 3. Example of the operations of LDBYTE and SBYTE, both using the same parameter sequence: 22, 27, and 41. Byte numbers are shown in the upper right-hand corner of each cell. One asterisk (*) signifies that the previous value is unchanged, two asterisks (**) designate binary zero (not necessarily the character code for zero), and three asterisks (***) designate values which have no effect whatever on the operation.

APPLICATION OF TRANSL

The TRANSL program accepts FORTRAN IV statements, one at a time, and produces TACPOL statements as required. The TACPOL is suitable for the PSS-B compiler.

The TRANSL program actually performs two separate translation tasks. Each task is run as a one-pass job. In the first task, TRANSL translates only the FORTRAN code statements, passing over the FORTRAN COMMON statements after noting them for proper placement. Other than for COMMON statements, this task produces one or more TACPOL statements for each FORTRAN statement. Continued FORTRAN statements are assembled into a single line before translation, and excessively long TACPOL results are broken into as many continued lines as are necessary as a last step in the translation process. A few FORTRAN statements (including arithmetic IF, CALL LDBYTE, CALL SBYTE, and CALL MIOP) result in more than one output line for the one FORTRAN input line. If the translation attempt fails, the original FORTRAN line is copied to the output file followed by a full line of slashes (i.e. ///////////////). The slashes are added as an aid to finding the faulty line in the listing. No further explanation is given of the failure. The problem must either be in one of the documented types of FORTRAN statements that will not translate (i.e. READ, WRITE, FORMAT, END FILE) or the occurrence of incorrect FORTRAN or that which does not conform to PLANIT standards. An error of the latter type might be expected to occur if TRANSL is used to translate a FORTRAN program which is not part of the PLANIT package.

The second task which TRANSL is designed to perform is the creation of TACPOL declare (DCL) statements which will be used by the Compool Generator. These two tasks are separate because a new Compool generation will not be needed for every new translation; only for the first one and each thereafter where any data declarations change. In performing the second task, the TRANSL program must scan all of the FORTRAN code for each initial reference to an item name, and construct a TACPOL data declaration statement for that name. The names may be found in COMMON statements or in the program statements since FORTRAN does not require declaration of simple item names prior to their use. However, TRANSL will not find the needed information in FORTRAN DIMENSION or EQUIVALENCE statements since those are not used in PLANIT coding.

In respect to this second task, the production of the DCL statements, the FORTRAN from the PLANIT system code

has requirements which differ from the FORTRAN from the PLANIT Generator code. The former has the requirement that all data be declared in COMMON; the latter does not have that requirement. A parameter is set for the TRANSL program to indicate which of the two cases it is processing. If it is processing PLANIT system code and encounters an item name which was not previously declared in a COMMON statement, it will insert a warning statement in the output file which says, UNDECLARED ITEM NAME ON THE NEXT LINE////////// (the slashes mark the line as before). This is simply a check on the PLANIT system code to make sure that all data items are declared as they are supposed to be. However, if it is processing PLANIT Generator code, instead of the warning upon encountering such an undefined item name, TRANSL will simply output an appropriate DCL statement. Note that the FORTRAN for the TRANSL source code is like the PLANIT Generator code and would be translated in the same manner.

One additional manipulation must be done with a few of the data items. Each compiler language has its own list of reserved words. Of course those being used by the PLANIT programs do not conflict with any on the FORTRAN reserved word list. However, some did conflict with the TACPOL reserved word list, namely "VALUE" and "L" which therefore the TRANSL program converts to "XVALUE" and "H", respectively, wherever they are encountered.

The remainder of the work of the TRANSL program can be understood most easily in a table which shows the translation rules for each FORTRAN form. The table is shown in Appendix A. Also included is a list of special operation characters for the TRANSL program, all of which occur in column one of the input line.

OPERATION OF TRANSL

In general, TRANSL is a batch program with one input file and one output file. When the program is started, it runs to completion without interruption.

The input file of FORTRAN statements must have 88 cards (lines) inserted prior to the first FORTRAN statement and one card appended to the end. The 88 prefixed cards constitute an initialization deck for the TRANSL program and the information on all but the second card is fixed and not subject to change. Comment cards may be added in addition to the 88 if desired so long as the letter C occurs in the first column. Otherwise, the initialization cards are of four general types, identifiable by the character in column one: (1) the character set card (heads the initialization deck), (2) the operation code card is second in order, having a digit in column one, (3) followed by 85 initialization cards each of which have a blank in columns one and two (with an intermix of comments permissible), (4) ended by a card with a \$ character in column one.

The single card which must be added to the end of the FORTRAN statements has a \$ character in column one.

Appendix B shows the 88-card initialization deck. Since only the second card contains variable data, it will be the only one that gets further description. Called the operation code, this second card designates which of the four tasks described in the previous section that the TRANSL program is currently expected to execute. The parameter is a single digit in column one having the value, one through four (i.e. 1, 2, 3 or 4). Columns two through 80 on this card are disregarded and may be used as a comment field. The four tasks which are designated will be denoted, Task 1, Task 2, Task 3, and Task 4 corresponding to the parameter settings of one, two, three and four, respectively. Each of the tasks is appropriate for a specific translation need with respect to one or more of the programs in the PLANIT package. These will be described by task number.

Task 1. Task 1 translates PLANIT system FORTRAN code into TACPOL statements. Note that the usual FORTRAN code as it is normally output from the PLANIT Generator program does not have the necessary initialization cards for the TRANSL program nor the ending \$ card. However, the PLANIT Generator program does have the facility which allows these cards to be included in the Merge File at the time of system generation. By including the necessary cards in the Merge File, the output from the Generator program is immediately ready to be accepted by the TRANSL program, making a continuous batchstream operation.

A sample Merge File is shown in Appendix C. Notice that the required TRANSL initialization cards are all present but are each shifted to the right five columns to make room for the characters, "*INS " which serves as a directive to the Generator program to insert the cards appropriately. Appendix C is called a sample because some of the parameter cards may change according to the desires of the installer, however the information on the cards which correspond to Appendix B, the initialization deck for the TRANSL program, will remain static. There will be no attempt to explain the parameter cards in Appendix C in this document since that is part of the normal PLANIT installation effort which is documented elsewhere.

The sample Merge File in Appendix C also shows some special &ENDPROG cards which are inserted at the end of each PLANIT subprogram, to be used by the TACPOL compiler in separating the overlays into appropriate subprograms. This will be understood more completely in the TACPOL compiler documentation.

For the execution of Task 1, it is sufficient to say that given a proper Merge File structure prior to the system generation of the FORTRAN, the result is ready to input to the TRANSL program and, in turn, its output is ready for the TACPOL compiler. If no mistakes have been made in the process, 100 percent translation can be expected. However, it may be wise to list the TACPOL at this point to confirm the successful translation. If any errors are flagged, the bad lines can usually be fixed without redoing the process. However, if undefined item names happen to be flagged, then the trouble could point to an error in the original PLANIT code.

Task 2. Task 2 is used to translate the PLANIT Generator program from FORTRAN into TACPOL. It will similarly translate its own FORTRAN source code into TACPOL. The translation of each of these FORTRAN programs should only be a onetime affair as opposed to repeated usage for Task 1. Therefore, a certain amount of post-translation hand repairs is tolerable. The expected repairs are of two kinds: translation failures and rearrangement of code. These are quite well-defined, as follows:

Translation will fail on four kinds of FORTRAN statements, READ, WRITE, FORMAT and END FILE. The number of these which will be encountered in the translation will be small and each will be flagged. The compiler personnel who were responsible for the TACPOL compiler felt that it would be preferable to make the necessary repairs after translation than to attempt to make TRANSL do the work. There will also probably be a need to add appropriate OPEN

calls since FORTRAN does this implicitly. The place where these should be added is documented in comments in the program listing.

The second expected failure is in the placement of subroutines with respect to the main program. This will be no problem for the TRANSL program because it has no subroutines. However, the PLANIT Generator program has two and, like most FORTRAN programs, they follow the main program. In TACPOL programs, procedures (i.e. TACPOL equivalent of subroutines) must precede the main program. Therefore, the translation of the PLANIT Generator code must be rearranged, placing the two procedures (subroutines) ahead of the first main program statement and, since the first subroutine also calls the second, the order of these must also be reversed. Therefore, with respect to the arrangement of the main program and the two subroutines in FORTRAN, the TACPOL version must be rearranged, placing the last procedure (subroutine) first, then working backward to the next procedure, and finally, the main program. Again, making the TRANSL program do this automatically would not be worthwhile, especially in view of how little it would get used.

Note that the initialization deck for Task 2 is identical for all tasks except for the change of the digit in column one of the second card from a "1" to a "2" (or "3" or "4" in the case of the other tasks). Also note that no batchstream is involved as in Task 1 since the code being translated is already FORTRAN and is not generated. Therefore, the 88-card deck as is shown in Appendix B (with the second card changed) is to be appended to the FORTRAN program ahead of the first FORTRAN card, and a terminal \$ card (\$ in column one) is to be appended to the end. The program is then ready to be input to the translator. The output should be listed, the translation failures (READ, WRITE, FORMAT and END FILE) fixed, OPEN calls added if necessary, code rearranged if necessary, and the DCL statements from Task 3 incorporated appropriately. The result is then ready for the TACPOL compiler.

Task 3. The purpose of Task 3 is to produce data declaration (DCL) statements to go with the TACPOL code which was produced under Task 2. The input file for Task 3 is set up identical to that for Task 2 except for the task number change on the second card. The result will be a set of DCL statements headed by a TACPOL PROC card. The PROC card will be of the form, PLANIT: PROC; where PLANIT is the program name. That name may be changed if desired. In the case of translated code for the translator program,

since it contains no subroutines, the entire set of DCL statements and the PROC card should be added just as they are to the front of the TACPOL code. If the translation failures have already been fixed, that program will be ready to compile.

The presence of subroutines in the PLANIT Generator program code makes the proper placement of the DCL statements more difficult. In general, the DCL statements which correspond to the COMMON items, together with the DCL statements for those items which are unique to the main program should be placed at the head of the main program. Use the PROC card here, too, as the first card of the main program. The subroutines each get only those DCL statements which are unique to the subroutines; they do not receive a copy of the items which were made up from COMMON statements. Again, the DCL statements immediately follow the PROC card. Two procedures will be explained for making the proper placement of the DCL statements, mainly in order to convey the intent. Either procedure may be followed.

Procedure one would be to execute all of Task 3 on the entire input file as described above. The resulting DCL statements would then have to be sorted and placed as follows: Mark the output lines in the DCL statements where they start repeating due to the encounter of a new set of COMMON statements. All statements from the PROC card to the first mark, inclusive, are to be placed at the head of the main program. Delete the next set of cards after the first mark including all that correspond to the COMMON statements which were translated. Place only the remaining DCL statements to the next mark after the PROC card in the subroutine (procedure) from which they came. The statements which corresponded to COMMON are omitted because TACPOL uses them globally from the main program if the DCL statement is not repeated in the procedure. Otherwise, item separation will occur. Repeat the process for the remaining set of DCL statements.

Alternately, the TRANSL program could be run three times in Task 3 for the Generator source code, first separating the FORTRAN into main, and each of the two subroutines. Run TRANSL for the last subroutine as if it were an entire program. Delete the DCL statements that correspond to the COMMON statements and place the remaining DCL statements ahead of the corresponding TACPOL procedure (after the PROC card). Repeat the process for the next subroutine. For the main program, keep all of the DCL statements including those from COMMON and head them with an appropriate PROC card (which will become the program name) and place them at the head of the TACPOL main code section.

The source of the confusion lies in the fact that FORTRAN expects its COMMON data to be listed repeatedly, a copy in the main program and each of the subroutines. Local data is often not declared. TACPOL, on the other hand, implements the COMMON data concept in procedures (subroutines) by omitting the declaration and thereby using the declaration in the calling code by default, calling it a globally defined item. Items which are declared in the procedure are strictly local items to that procedure.

FORTRAN programmers will recognize that in COMMON statements, the exact list of item names need not repeat. However, one of the constraints for the coding of PLANIT-related programs is that all COMMON lists within a program be identical and that no item is explicitly declared in any place other than COMMON (i.e. there is no use of DIMENSION, EQUIVALANCE, TYPE, etc.). To deviate from these conventions would confuse the translator program which only reinforces the conclusion that TRANSL is not a general purpose FORTRAN-to-TACPOL translator.

Task 4. Task 4 generates the DCL statements for the PLANIT system TACPOL code generated in Task 1. The essential difference between Task 3 and Task 4 is in the constraint that all items used by the PLANIT code must be explicitly defined (declared) in the COMMON list. Those found in the COMMON list will be translated into TACPOL DCL statements. Any found in the code which were not previously declared in the COMMON list will (unlike Task 3) produce an error in the output file. To run Task 4, the input file from Task 1 can be changed such that the second card shows a "4" instead of a "1" and the TRANSL job be run again. This time the result should be DCL statements. Since each overlay (subprogram) will have its own COMMON list, several DCL lists will result only one of which will be useful.

Task 4 can be shortened considerably. Since all error checking was already done in Task 1, the input file for Task 4 can be shortened to any point beyond the end of the first listing of COMMON. Therefore, insert a "\$" card ("\$" in column one) after the last COMMON statement of the first COMMON listing in the input file and TRANSL will stop when the \$ is encountered. The result will be a single list of DCL statements, corresponding line-by-line with the COMMON statements. This is all that is needed. Note that this file into which the "\$" card is being inserted is the output file from the PLANIT Generator program. The Generator Merge file is set up in such a way so that the output file is right for input

to the TRANSL program. It is the second card of this file that gets changed from a "1" to a "4" and also a "\$" card must be inserted beyond the end of the first listing of COMMON. It might be helpful to note that the placement of the "\$" card is not critical so long as it follows the COMMON cards. Therefore it could be arbitrarily inserted at, say, line 600 which is well beyond the COMMON cards, perhaps saving the listing of the file to find the exact place.

Having generated a set of DCL statements which correspond line-for-line with any of the several identical COMMON lists, the output file (of DCL cards) will then be properly formatted and ready to be inserted into the procedure which generates the COMPOOL for the TACPOL version of the PLANIT system. The operation of the COMPOOL generator is described elsewhere, being a part of the standard ANGYK-12 software facility.

FINAL INSTALLATION CONSIDERATIONS

These are some observations that seemed best to be presented near the end, when the reader has more of the total plan in view.

PROC Cards. It was noted that while both Task 3 and Task 4 produce DCL statements, only Task 3 generates a PROC card to head the statements (in the form, PLANIT: PROC;). This is because the PROC card is needed in that position to head the main program of the kind that is translated in Tasks 2 & 3, namely, the PLANIT Generator and the Translator. However, it may be desirable to change the procedure names to something more appropriate like: GENER: PROC; or TRANSL: PROC;. The PROC card is not generated in Task 4 because these DCL statements are destined for the COMPOOL generator. In this case, the needed PROC card is supplied through the Merge File prior to generation and is placed appropriately during translation in Task 1.

Special Subroutine (Procedure) Calls. The TRANSL program makes the appropriate translation for FORTRAN subroutines and the calls to them provided that no argument strings exist in the calling sequences. It is noted here for information that the call format produced by the TRANSL program will be considerably different in Task 1 than in Task 2. In Task 1, what amount to subroutine calls actually invoke a level change, initiating a new program. This is peculiar to the special PLANIT operating system which was developed for this computer. In Task 2, the calling format is standard TACPOL. The cause of special interest is the calling sequence for the special procedures, LDBYTE and SBYTE. In the PLANIT system, as translated by Task 1, these are separate programs operating at their own level. However, for the PLANIT Generator and the Translator programs, both of which use LDBYTE and SBYTE routines which are identical to those for PLANIT, they are placed at the head of the respective programs in normal TACPOL fashion and are invoked through the standard calling format which is produced under Task 2. Therefore, the TACPOL versions of the PLANIT Generator and the Translator programs should each begin with these two TACPOL procedures, LDBYTE and SBYTE.

In summary, the final procedure arrangement for the TACPOL version of the PLANIT Generator program should be (from first to last):

1. LDBYTE
2. SBYTE
3. NUMBER (Last FORTRAN subroutine)
4. BRCKET (First FORTRAN subroutine)
5. Main program

The Main program should start with its PROC statement, followed by the DCL statements corresponding to FORTRAN COMMON and then the DCL statements for the items local to the Main program. Each of the procedures should begin with appropriate PROC statements, followed by only those DCL statements that are for items local to that procedure.

The final arrangement for the TACPOL version of the Translator program is comparable, as follows:

1. LDBYTE
2. SBYTE
3. Main program

The arrangement of the PROC cards and DCL statements is the same as above.

APPENDIX A

**Table of rules by which the TRANSL program translates
FORTRAN particles into corresponding TACPOL particles.**

<u>FORTRAN Particle</u>	(becomes)	<u>TACPOL Particle</u>	<u>Comment</u>
1. Label (e.g. 1)		Q1:	FORTRAN labels begin in column 1, TACPOL in column 2.
2. CONTINUE		(deleted)	
3. DO (e.g. DO 1 I=1,10)		DO I-1 BY 1 TO 10	An END; will be placed appropriately after label 1.
4. DO (e.g. DO 2 I=J,K,3)		DO I-J BY 3 TO K	An END; will be placed appropriately after label 2.
5. IF (e.g. IF(...) expression)		IF(...) THEN expression;	
6. IF (e.g. IF(...) 1,2,3)		IF(... LT 0) THEN GOTO Q1; IF(... EQ 0) THEN GOTO Q2; GOTO Q3;	Note: one of the three statements will be omitted if its corresponding label appears on the next line.
7. COMMON			Only noted for data check. Not translated to output.
8. First occurrence of an integer item name (e.g. I)		DCL I BIN FIXED;	
9. First occurrence of a real item name (e.g. X)		DCL X BIN FIXED (62,15);	
10. CALL (e.g. CALL BRCKET)		CALL BRCKET;	
11. CALL LDBYTE CALL SBYTE CALL MIOP			In Task 1, these three calls are replaced by direct code which causes an appropriate program level change in accordance with the PLANI's Operating System which was developed for the ANGYK-12.
12. SUBROUTINE (e.g. SUBROUTINE PLAN1)		PLAN1: PROC;	Use of argument strings is invalid.

<u>FORTRAN Particle</u>	(becomes)	<u>TACPOL Particle</u>	<u>Comment</u>
13. FORMAT			Flagged and not translated.
14. READ			Flagged and not translated.
15. WRITE			Flagged and not translated.
16. PAUSE		GOTO QREND;	The label QREND: will be added just before the END; statement.
17. STOP		(deleted)	STOP can only occur just prior to the END statement.
18. END		END;	
19. ENDFILE			Flagged and not translated.
20. GO TO (e.g. GO TO 1)		GOTO Q1;	
21. GOTO (e.g. GOTO 1)		GOTO Q1;	
22. ABS(...)		ABS(...)	
23. IABS(...)		ABS(...)	
24. ALOG(...)		LONG(LN(SHORT(...)))	
25. SQRT(...)		LONG(SQRT(SHORT(...)))	
26. SIN(...)		SINFUN(...)	The SIN and COS calls are translated to calls on special SINFUN and COSFUN procedures which convert the radian arguments into "BAMS" for the TACPOL compiler.
27. COS(...)		COSFUN(...)	The SINFUN and COSFUN procedures are inserted in the Generator jobstream appropriately.

	<u>FORTRAN Particle</u>	(becomes)	<u>TACPOL Particle</u>	<u>Comment</u>
28.	RETURN		GOTO QREND;	
29.	.NE. .LT. .LE. .EQ. .GE. .GT.		NE LT LE EQ GE GT	
30.	.AND. .OR.)AND()OR(
31.	--IABS		--1*ABS	Correction for compiler error.
32.	--ABS		--1.0L*ABS	Correction for compiler error.
33.	Real division, e.g. X/Y or (...)/Y		TRUNC(...,30)/Y	Correct for shifting of decimal places.
34.	Computed GOTO, e.g. GO TO(1,2,3),I		CALL SWITCH(I,Q1,Q2,Q3);	
35.	Mode conversion, e.g. X-I I-X		X-LONG(I) I-SHORT(X)	The arguments of LONG and SHORT can be arbitrary expressions.
36.	Real literal numeric conversion, e.g. 2. 2.0 2.1		2.0L 2.0L 2.1L	
37.	Reserved word conversion: VALUE L		XVALUE H	VALUE and L have special meaning in TACPOL.

<u>FORTRAN Particle</u> (becomes)	<u>TACPOL Particle</u>	<u>Comment</u>
38. Comments: the line beginning with "C" in column one.	The characters /* go in columns 2 & 3 (column 1 blanked) and the characters */ go in columns 71 & 72.	
39. The letter "B" in column one.	Blank column one and output the line with no further modification.	
40. Column one not a digit, blank, "B" or "C"	Output the line as is.	
41. Column one a digit or blank.	FORTRAN line to be translated according to the above rules.	
42. Complete FORTRAN line.	Append a semicolon (;) to the end of the translated line.	
43. Characters **	Replace entire line with NDKTR=18;	TACPOL does not accept variable exponents.
44. \$\$ (end of job card)	&ENDPROG	TACPOL end of job card.

APPENDIX B

Listing of the initialization cards which must be inserted in the input file so that they precede the first FORTRAN statement to be translated.

0123456789ABCDEF GHIJKL MNOPQRST UVWXYZ--S/() ? : ; ' " # \$ %
1 NTTYPE PARAMETER (CCL. 1 ONLY). DESCRIPTION FOLLOWS:

CONTINUE	00001000
IF	00001100
COMMON	00001200
CALL	00001300
SUBROUTINE	00001400
FCRMAP	00001500
REACT	00001600
WRITE	00001700
DELETE	00001800
STOP	00001900
SPDC	00002000
ENCFIL	00002100
GC	00002200
GOIC	00002300
ABS	00002400
TABS	00002500
ALCT	00002600
SCRT	00002700
SCRS	00002800
RETLM	00002900
N/U	00003000
N/U	00003100
VALUE	00003200
N/U	00003300
N/U	00003400
N/U	00003500
LE	00003600
LE	00003700
LE	00003800
LE	00003900
OR	00004000
N/U	00004100
MSGTAL GR MSGDR NOT LARGE ENOUGH.	00004200
LE	00004300
LE	00004400
LE	00004500
LE	00004600
LE	00004700
LE	00004800
LE	00004900
LE	00005000
LE	00005100
LE	00005200
LE	00005300
LE	00005400
LE	00005500
LE	00005600
LE	00005700
LE	00005800
LE	00005900
LE	00006000

L

APPENDIX C

Sample listing of a PLANIT Merge file with the Translator initialization deck embedded in such a way that the resulting FORTRAN will lead off with the required initialization deck. This constitutes a batchstream for going from the original PLANIT code through FORTRAN to TACPOL with no intervening manipulation.


```

IF COMMON
CALL SURROUTINE
FCRYAT
READ
WRITE
PAUSE
STOP
ENDPROC
ENCFILE
GOTO
GOTO
ABS
TARS
ALOG
SIN
CCS
RETURN
N/U
N/U
VALUE
L
N/U
N/U
N/U
N/E
.LT
.LE
.EQ
.GE
.GT
.CR
.AND
N/U
N/U
MSGTBL OR P$GDR NOT LARGE ENOUGH.
EQ
GT
BY 1 TO
END
THENX
XX 0
XX 0.0L
CALL SWITCH(
SHORT(TRUNC(
LCNG(
))
)
FUN
N/U
L$SHORT
LN
CCL
CCL
QREND: ENC
GOTO OR ENC

```

```

0006100
00002200
00006300
00006400
00006500
00006600
00006700
00006800
00006900
00007000
00007100
00007200
00007300
00007400
00007500
00007600
00007700
00007800
00007900
00008000
00008100
00008200
00008300
00008400
00008500
00008600
00008700
00008800
00008900
00009000
00009100
00009200
00009300
00009400
00009500
00009600
00009700
00009800
00009900
00010000
00010100
00010200
00010300
00010400
00010500
00010600
00010700
00010800
00010900
00011000
00011100
00011200
00011300
00011400
00011500
00011600
00011700
00011800
00011900
00012000

```

BIN FIXED: (62,15):


```

00224100
00224200
00224300
00224400
00224500
00224600
00224700
00224800
00224900
00225000
00225100
00225200
00225300
00225400
00225500
00225600
00225700
00225800
00225900
00226000
00226100
00226200
00226300
00226400
00226500
00226600
00226700
00226800
00226900
00227000
00227100
00227200
00227300
00227400
00227500
00227600
00227700
00227800
00227900
00228000
00228100
00228200
00228300
00228400
00228500
00228600
00228700
00228800
00228900
00229000
00229100
00229200
00229300
00229400
00229500
00229600
00229700
00229800
00229900
00230000

```

```

CCCE: X+R'1'.6
LCF X+2+R'1'.7
SALC =32.6
SARC =32.6
MPF RADBAMS-S.7
SALC =16.6
RPF R'1'.1
XPF CNE-S.6
ADF SINDATA+R'1'.6
SCF SINDATA+R'1'.6
XFN FX'4CC00000'
GEN FO.155154943831
ENC SINCATA = SIN(SINDATA);
CCCE: SINDATA+R'1'.6
LCF S+2.6
XPF =X'.7FFF'.6
SARC =47.6
ACFC Z+R'1'.6
SDF Z+2+R'1'.7
ENC
RETURN (2);
END: /* SINFUN PROC */
COSFUN: PROC(X) FIXED(62,15);
DCL Z BIN FIXED (62,15);
DCL Z CCSDATA BIN FIXED (31,30);
CCCE: X+R'1'.6
LCF X+2+R'1'.7
SALC =32.6
SARC =32.6
MPF RADBAMS-S.7
SALC =16.6
RPF R'1'.1
XPF S+2.6
ADF CNE-S.6
SCF COSDATA+R'1'.6
XFN FX'4CC0000'
GEN FC.155154943831
ENC COSDATA = COS(COSDATA);
CCCE: COSDATA+R'1'.6
LCF S+2.6
XPF =X'.7FFF'.6
SARC =47.6
ACFC Z+R'1'.6
SDF Z+2+R'1'.7
ENC
RETURN (2);
END: /* COSFUN PROC */
IGOTO=IGOTO-<CALC-1>
GO TO<CALC>.1.1.<EVAL>.<CNOFFS>00
.<COMPGCTO>.<LOGRITHM>
.<DEFLANGIN>.<MATRIXDEFINE>.<MATRIXARITHM>.<SET>

```



```

*INS GENCPROG
SUBROUTINE PLAN4
*COMPEN
IGOTO=IGOTC-<STUSRCH-1>
GC TO<STUSRCH>,<LOGIN>,<ADDEL>,<ATTACH>,<SAVE>
1 <LOCKER>,<RESTART>,<FRERE>),IGOTC
2
1 IGOTO=0
RETURN
*PLANIT FILE --- INC: STUSRCH
*PLANIT FILE --- INC: LOGIN
*PLANIT FILE --- INC: ADDEL
*PLANIT FILE --- INC: ATTACH
*PLANIT FILE --- INC: SAVE
*PLANIT FILE --- INC: LOCKER
*PLANIT FILE --- INC: RESTART
*PLANIT FILE --- INC: FRERE
*ENC
*INS GENCPROG
SUBROUTINE PLAN5
*COMPEN
IGOTO=IGOTC-<COMMAND-1>
GO TO<COMMAND>),IGOTC
1 <SPool>),IGOTC
2
1 IGOTO=0
RETURN
*PLANIT FILE --- INC: COMMAND
*PLANIT FILE --- INC: EDIT1
*PLANIT FILE --- INC: EDIT2
*PLANIT FILE --- INC: EDIT3
*PLANIT FILE --- INC: BUILD
*PLANIT FILE --- INC: REORDER
*PLANIT FILE --- INC: SPCOL
*ENC
*INS GENCPROG
SUBROUTINE PLAN6
*COMPEN
IGOTO=IGOTC-<SYSTEM-1>
GO TO<SYSTEM>),<DELETE>,<UNLOAD>),IGOTC
2
1 IGOTO=0
RETURN
*PLANIT FILE --- INC: SYSTEM
*PLANIT FILE --- USE: OCTAL CODE
*PLANIT FILE --- USE: OCTAL CODE
*PLANIT FILE --- INC: DELETE
*PLANIT FILE --- INC: UNLOAD
*ENC
*INS GENCPROG
SUBROUTINE PLAN7
*COMPEN
IGOTO=IGOTC-<TEXT-1>
GO TO<TEXT>),<RENUM>,<PRESET>),IGOTO
2
1 IGOTO=0
RETURN
*PLANIT FILE --- INC: EDTEXT
*PLANIT FILE --- INC: RENUM

```

```

00036100
00036200
00036300
00036400
00036500
00036600
00036700
00036800
00036900
00037000
00037100
00037200
00037300
00037400
00037500
00037600
00037700
00037800
00037900
00038000
00038100
00038200
00038300
00038400
00038500
00038600
00038700
00038800
00038900
00039000
00039100
00039200
00039300
00039400
00039500
00039600
00039700
00039800
00039900
00040000
00040100
00040200
00040300
00040400
00040500
00040600
00040700
00040800
00040900
00041000
00041100
00041200
00041300
00041400
00041500
00041600
00041700
00041800
00041900
00042000

```



```

*PLAN IT FILE -- INC. PRESET
*PLAN IT FILE -- INC. OF FILE
*FNC
*INS GENPROG
*COMPCN SCAROUTINE PLANB
IGOTO-IGCTC-<LOGOUT-1>
GG TO<LOGCLT>.<SHOW>.<DISPLAY>.<ACCOUNT>
1 IGTOTO=>.<WRITEP>.<CONFIRM>.<MODISK>.<FIX>}.IGCTO
2 IGTOTO=>
RETURN
*PLAN IT FILE -- INC. LOGOUT
*PLAN IT FILE -- INC. SHOW
*PLAN IT FILE -- INC. DISPLAY
*PLAN IT FILE -- INC. ACCOUNT
*PLAN IT FILE -- INC. WRITEP
*PLAN IT FILE -- INC. CONFIRM
*PLAN IT FILE -- INC. MODISK
*ENC
*INS $$$
$$$

```

```

CCC42100
0JC42200
00042300
00042400
00042500
00042600
00042700
00042800
00042900
00043000
00043100
00043200
00043300
00043400
00043500
00043600
00043700
00043800
00043900
00044000
00044100
00044200
00044300
00044400
00044500

```

APPENDIX D**Listing of the TRANSL program.**

```

PROGRAM TRANSL
*****
THIS PROGRAM TRANSLATES THE FORTRAN PROGRAM PRODUCED BY THE PLANIT
GENERATOR INTO THE TACPOL LANGUAGE. NO ADDITIONAL WORK
SHOULD BE NECESSARY ON THE RESULTING PLANIT CODE. HOWEVER,
FOR THE TRANSLATING THE FORTRAN OF THE GENERATOR PROGRAM ITSELF.
OR THE FORTRAN IN THIS PROGRAM. INTO TACPOL, CERTAIN STATE-
MENTS MUST BE FIXED BY HAND, INCLUDING: READ, WRITE, FCRR
AND END FILE. ALSO A DIFFERENT OPTION MUST BE USED FOR
GENERATING THE DCL STATEMENTS SINCE NOT ALL OF THE ITEMS WILL
HAVE BEEN DECLARED IN COMMON AS THEY ARE IN THE PLANIT CODE.
SEE THE DESCRIPTION OF ITEM TYPE, BELOW.
ITEM TYPE IS A RUN OPTION PARAMETER WITH FOUR LEGAL VALUES
(1-4) SELECTING ONE OF THE FOLLOWING OPTIONS:
1. OUTPUT TACPOL CODE. DO NOT OUTPUT DCL STATEMENTS. FLAG
   ANY ITEMS FOUND IN THE CODE WHICH HAVE NOT BEEN DECLARED
   IN A PRIOR COMMON STATEMENT.
2. OUTPUT TACPOL CODE. DO NOT OUTPUT DCL STATEMENTS. DO
   NOT FLAG UNDECLARED ITEMS WHICH ARE FOUND IN THE COOF.
3. OUTPUT DCL STATEMENTS. DO NOT OUTPUT CODE. DO NOT FLAG
   UNDECLARED ITEMS WHICH ARE FOUND IN THE CODE.
4. OUTPUT DCL STATEMENTS. DO NOT OUTPUT CODE. FLAG ANY
   ITEMS FOUND IN THE CODE WHICH HAVE NOT BEEN DECLARED
   IN A PRIOR COMMON STATEMENT.
THE TRANSLATOR WILL REPORT THREE KINDS OF ERRORS. ALL
MIXED IN THE LINES OF THE OUTPUT FILE. THEY WILL BE
FLAGGED AND CAN USUALLY BE FIXED WITHOUT REPEATING
THE JOB. THE THREE TYPES OF ERRORS ARE:
1. A MESSAGE ON THE LAST LINE AND ABRCT. THIS
   ERROR IS UNRECOVERABLE AND MUST BE FIXED AND
   THE JOB RUN AGAIN.
2. THE UNTRANSLATED INPUT LINE MAY BE PRINTED
   IMMEDIATELY FOLLOWED BY A LINE OF SLASHES
   (//////////). THIS INDICATES A REJECT
   OF THE INPUT LINE FOR TRANSLATION. AN EXAM-
   PLE OF THIS LINE WILL USUALLY REVEAL THE
   PROBLEM. WHETHER A FORTRAN ERROR OR A STRING
   THAT THE TRANSLATOR IS NOT EQUIPPED TO TRANS-
   LATE. THIS KIND CAN OFTEN BE FIXED.
3. A MESSAGE MAY APPEAR WHICH SAYS:
   UNDECLARED ITEM NAME ON THE NEXT LINE. OPTION OF
   WHICH CAN RESULT FROM CHOOSING ITEM TYPE OPTION OF
   ONE OR FOUR. THIS IS SPECIFICALLY IN COMMON AND
   WHERE ALL ITEMS ARE TO BE DECLARED IN COMMON AND
   THE TRANSLATOR HAS DETECTED A FAILURE.
   MESSAGE CAN BE DELETED AND IGNORED BUT THE EFFECT
   OF THE UNDECLARED ITEM TO PLANIT SHOULD BE CHECKED.
AT THE END OF THE JOB, A STATEMENT WILL PRINT THE NUMBER
OF FLAGGED LINES (IF THERE ARE ANY). THEY CAN THEN BE

```

```

00000100
00000200
00000300
00000400
00000500
00000600
00000700
00000800
00000900
00001000
00001100
00001200
00001300
00001400
00001500
00001600
00001700
00001800
00001900
00002000
00002100
00002200
00002300
00002400
00002500
00002600
00002700
00002800
00002900
00003000
00003100
00003200
00003300
00003400
00003500
00003600
00003700
00003800
00003900
00004000
00004100
00004200
00004300
00004400
00004500
00004600
00004700
00004800
00004900
00005000
00005100
00005200
00005300
00005400
00005500
00005600
00005700
00005800
00005900
00006000

```

C: SPOTTED EASILY IN THE LISTING OR BY USING A PROGRAM TO
 C: SEARCH FOR /// CHARACTERS. IF THE PROGRAM INPUT LINE
 C: IS MORE THAN ONE CARD LONG, ONLY THE LAST CARD WILL BE
 C: PRINTED AND THE LISTING SHOULD BE CONSULTED FOR THE
 C: FULL LINE.
 C:
 C:

```

COMMON FIRST(1)
COMMON IBLANK
COMMON MSGTBL(20C)
COMMON JMRK
COMMON IBYTE1
COMMON IBYTE2
COMMON INDEX
COMMON LBYTE(500)
COMMON ICHPRE(50C)
COMMON IFLINE(10C)
COMMON INBUFF(30)
COMMON INQUE(30)
COMMON ICHK(6)
COMMON LABDO(5)
COMMON LISTN(150C)
COMMON LKARDUM
COMMON KARSET(58)
COMMON MAPDUM
COMMON MASP(255)
COMMON MSGDR1(90)
COMMON MSGDR2(90)
COMMON JFLG
COMMON MPAREN(5)
COMMON KINDST(5)
COMMON NFN1(5)
COMMON NFN2(5)

```

C: *****
 C: THE FOLLOWING STATEMENTS (BETWEEN RC'S OF ASTERISKS)
 C: WOULD VARY ON A DIFFERENT COMPUTER. ALSO, THE READ, WRITE,
 C: FORMAT AND END FILE STATEMENTS WILL HAVE TO BE CHANGED FOR
 C: TRANSLATION INTO TACPOL.
 C:
 C: BYTES PER WORD OF THE COMPUTER RUNNING THIS PROGRAM.
 C: IBYTE=4
 C: GO TO 9

```

C: 1000  FFORMAT(20A4)
C:
C: 1  CONTINUE
C: 2  END OF THE TRANSLATE JOB. END FILE STATEMENT ON NEXT
C: 3  LINE CLOSSES THE OUTPUT FILE. CHANGE OR DELETE IT IF
C: 4  C: NECESSARY
C: 5  END FILE 2
C: 6  GO TO 158
C: 7  CONTINUE
C: 8  READ(1,1000) (INBUFF(I), I=1,NLMBUF)
C: 9  GO TO 4
C: 10 CONTINUE
C: 11 WRITE(2,1000) (INBUFF(I), I=1,NLMBUF)
C: 12 GO TO 183
C: *****
C: 13 CONTINUE
C: 14 C: QUEJE INTO BUFFER.

```

```

00006100
00006200
00006300
00006400
00006500
00006600
00006700
00006800
00006900
00007000
00007100
00007200
00007300
00007400
00007500
00007600
00007700
00007800
00007900
00008000
00008100
00008200
00008300
00008400
00008500
00008600
00008700
00008800
00008900
00009000
00009100
00009200
00009300
00009400
00009500
00009600
00009700
00009800
00009900
00010000
00010100
00010200
00010300
00010400
00010500
00010600
00010700
00010800
00010900
00011000
00011100
00011200
00011300
00011400
00011500
00011600
00011700
00011800
00011900
00012000

```

00012100
 00012200
 00012300
 00012400
 00012500
 00012600
 00012700
 00012800
 00012900
 00013000
 00013100
 00013200
 00013300
 00013400
 00013500
 00013600
 00013700
 00013800
 00013900
 00014000
 00014100
 00014200
 00014300
 00014400
 00014500
 00014600
 00014700
 00014800
 00014900
 00015000
 00015100
 00015200
 00015300
 00015400
 00015500
 00015600
 00015700
 00015800
 00015900
 00016000
 00016100
 00016200
 00016300
 00016400
 00016500
 00016600
 00016700
 00016800
 00016900
 00017000
 00017100
 00017200
 00017300
 00017400
 00017500
 00017600
 00017700
 00017800
 00017900
 00018000

```

D3 5 I=1,NUMBUF
J=INBUFF(I)
INBUFF(I)=INQUE(I)
INQUE(I)=J
CONTINUE
IFERR=0
IBYT1=IBUFF+INCR
IBYT2=IBUFF+71
INDEX=IMLBYT+MREAD
CALL LDBYTE
IF(JSW-LE.0) GO TO 11
MREAD=MREAD-INCR+72
CONTINUE
THE END OF THE IMPLT LINE.
IF(LBYTE(MREAD)-NE. KARSET(5C)) GO TO 8
MREAD=MREAD-1
IF(MREAD.GT.C) GO TO 6
CONTINUE
MREAD=0
IF(CHK(1).EQ.KARSET(55)) GO TO 189
LSTCHK=KARSET(47)
GO TO 2
CONTINUE
IF(JSW.EQ.1) GO TO 13
C: BUFFERS LOADED AND END FOUND. CHECK FOR CCNTINUATIONS
C: AND IF LABELS
IBYT1=IBQUE
IBYT2=IBYT1+5
INDEX=IMCHK
CALL LDBYTE
IF(CHK(1).NE.5C) GO TO 3E
IF(MAP(I).NE.5C) GO TO 3E
J=CHK(6)
IF(MAP(J).GT.25) GO TO 3E
C: CONTINUATION CARD IF NOT TRUE.
IF(MREAD+66.GE. MAXLIN) GO TO 152
INCR=6
GO TO 2
CONTINUE
C: INITIALIZATION SECTION.
C: IN THE RARE EVENT THAT ARRAY DIMENSIONS ARE TO BE
C: CHANGED, THE FOLLOWING ASSIGNMENTS MUST ALSO BE
C: CHANGED CORRESPONDINGLY.
C: DIMENSION OF MSGTBL:
MSGDIM=200
C: SIZE OF WORKING LINE (DIMENSIONS OF LBYTE AND ICMPRE):
MAXLIN=500
C: DIMENSION OF IRLINE:
IRLDIM=150
C: BUFFER LENGTHS (DIMENSION OF INBUFF AND INQUE):
IRFCIM=30
C: SIZE OF ITEM DICTIONARY (ONE-THIRD OF DIMENSION OF LISTN):
MAXNAM=500
C: END OF ARRAY DIMENSION ASSIGNMENTS. ARRAYS BEFCH LISTN
C: MAY HAVE DIMENSIONS CHANGED, IF NECESSARY, WITHOUT THE
C: NEED FOR ADJUSTMENTS HERE.
C: WORD NUMBER ASSIGNMENTS.

```

```

00018100
00018200
00018300
00018400
00018500
00018600
00018700
00018800
00018900
00019000
00019100
00019200
00019300
00019400
00019500
00019600
00019700
00019800
00019900
00020000
00020100
00020200
00020300
00020400
00020500
00020600
00020700
00020800
00020900
00021000
00021100
00021200
00021300
00021400
00021500
00021600
00021700
00021800
00021900
00022000
00022100
00022200
00022300
00022400
00022500
00022600
00022700
00022800
00022900
00023000
00023100
00023200
00023300
00023400
00023500
00023600
00023700
00023800
00023900
00024000

```

```

I4LBYT=MSGDIM*7
I4CMP=I4LBYT*MAXLIN*7
I4BUFF=I4CMP*MAXLIN*IFLDIM
I4QUE=I4BUFF*IBFDIM
I4CHK=I4QUE*IRFDIM
NUMBER ASSIGNMENTS
I4BUFF=I4BUFF*IBYTE-IBYTE+1
I4CMP=I4CMP*IBYTE-IBYTE+1
I4QUE=I4QUE*IBYTE-IBYTE+1

C: BYTE
C:
10
IFIRST(1)=I4LBYT
DO 10 I=1, 255
MAP(I)=58
CNTINUE
MAPDUM=58
L INE=0
INCRE=0
INTRY=0
MSGN=0
JMPK=0
JFLG=0
NJMERR=0
NJMDO=0
IFRTRN=0
JSM=-1
IREAD=1
NTYPE=1
NJMBUF=75/IBYTE+1
GO TO 7
CNTINUE
C: SET UP TRANSLATION TABLES.
JSM=JSM+1
IF(JSM-LE 0) GO TO 2
DO 12 I=1, 56
J=LBYTE(I)
KAPSET(I-1)=J
MAP(I)=I-1
ICMPRE(I)=LBYTE(I)
CNTINUE
C: STORE BLANKS
I4LBYT=IBYTE+1
I4BYT2=2*IBYTE
INDEX=I4CMP
CALL SBYTE
GO TO 7
CNTINUE
I=LBYTE(1)
I=MAP(I)
IF(I-LE 9) GO TO 14
IF(I-LE 5) GO TO 15
IF(I-NE IZATIDN INPLT DECK.
C: STORE MESSAGES
IBYT1=MSGDR(2*MSGN)+1
IF(MSGN-EQ 0) IBYT1=2*IBYTE+1
IBYT2=IBYT1+NRAD-3
INDEX=I4LBYT+2
CALL SBYTE
MSGN=MSGN+1
MSGDR(1*MSGN)=IBYT1

```

000 24100
 000 24200
 000 24300
 000 24400
 000 24500
 000 24600
 000 24700
 000 24800
 000 24900
 000 25000
 000 25100
 000 25200
 000 25300
 000 25400
 000 25500
 000 25600
 000 25700
 000 25800
 000 25900
 000 26000
 000 26100
 000 26200
 000 26300
 000 26400
 000 26500
 000 26600
 000 26700
 000 26800
 000 26900
 000 27000
 000 27100
 000 27200
 000 27300
 000 27400
 000 27500
 000 27600
 000 27700
 000 27800
 000 27900
 000 28000
 000 28100
 000 28200
 000 28300
 000 28400
 000 28500
 000 28600
 000 28700
 000 28800
 000 28900
 000 29000
 000 29100
 000 29200
 000 29300
 000 29400
 000 29500
 000 29600
 000 29700
 000 29800
 000 29900

```

MSGDR2(MSGN1=18Y1) GO TO 7
IF(JMK .EQ. JFLG) GO TO 7
N5=40
GO TO 196
C: STORE TRANSLATION OPTION PARAMETER.
  14  CNT INUE
  CNT INUE
  NTYPE=1
  15  CNT INUE
  CNT INUE
  JSH=2
  IF(NTYPE .NE. 3) GO TO 7
  N5=79
  GO TO 196
  16  MARK OFF ITEM NAME AND COMPARE STRINGS TO PRESTORED
  C: NAMES . START AT LOOP TO N1. N2, N3 MARK OFF LIST OF
  C: WORDS . TO BE COMPARED. N2 MARKS THE FOUND NAME.
  C: N4 IS THE RETURN CODE.
  17  NI=LOOP
  CNT INUE
  NI=NI+1
  IF(N1 .GT. NREAD) GO TO 18
  I=I+1
  IF(LBYTE(N1))
  IF(MAP(I) .LE. 35) GO TO 17
  CNT INUE
  NI=NI-1
  C: CHECK BASIC WORD LIST.
  18  CNT INUE
  IBYT1=MSGDR1(N2)
  IBYT2=MSGDR2(N2)
  IF(18Y1 .NE. NI-LOOP) GO TO 27
  CHARACTER COUNT OK
  INDEX=INDEX+1
  CALL LDBYTE
  DO 20 I=LOOP,N1
  IF(LBYTE(I) .NE. ICMPE(I)) GO TO 27
  CNT INUE
  20  MATCH FOUND.
  IF(N2 .LE. 25) GO TO 54
  C: MATCH ON COMMAND WORD.
  IF(N3 .EQ. 38) GO TO 125
  C: MATCH ON RELATIONAL.
  IF(N3 .EQ. 70) GO TO 55
  C: MATCH ON CALL NAME.
  C: OTHERWISE CHECK FOR NAME CHANGE.
  C: VALUE TO XALUE.
  IF(N2 .EQ. 26) LBYTE(LOOP)=KARSET(33)
  C: L TO H.
  IF(N2 .NE. 27) GO TO 21
  LBYTE(LOOP)=KARSET(17)
  GO TO 22
  CNT INUE
  21  CHECK ADDR AGAINST NEW WORD LIST.
  IF(LBYTE(LOOP))
  IF(MAP(I) .GE. 1E .AND. MAP(I) .LE. 23) GO TO 22
  C: FLOATING ITEM.
  IF(IEQ .EQ. 0) KINDL=1
  IF(IEQ .NE. 0) KINDR=1
  
```

```

000 30100
000 30200
000 30300
000 30400
000 30500
000 30600
000 30700
000 30800
000 30900
000 31000
000 31100
000 31200
000 31300
000 31400
000 31500
000 31600
000 31700
000 31800
000 31900
000 32000
000 32100
000 32200
000 32300
000 32400
000 32500
000 32600
000 32700
000 32800
000 32900
000 33000
000 33100
000 33200
000 33300
000 33400
000 33500
000 33600
000 33700
000 33800
000 33900
000 34000
000 34100
000 34200
000 34300
000 34400
000 34500
000 34600
000 34700
000 34800
000 34900
000 35000
000 35100
000 35200
000 35300
000 35400
000 35500
000 35600
000 35700
000 35800
000 35900
000 36000

```

```

22 CONTINUE 1 .OR. IEQ .NE. C) GO TO 26
   IF(N4)NE. 1 .OR. IEQ .NE. C) GO TO 26
   ICHPRE(1)=IBLANK
   ICHPRE(2)=IBLANK
   ICHPRE(3)=IBLANK
   ICHPRE(4)=IBLANK
   ICHPRE(5)=IBLANK
   ICHPRE(6)=IBLANK
   ICHPRE(7)=IBLANK
   ICHPRE(8)=IBLANK
   ICHPRE(9)=IBLANK
   ICHPRE(10)=IBLANK
   ICHPRE(11)=IBLANK
   ICHPRE(12)=IBLANK
   ICHPRE(13)=IBLANK
   ICHPRE(14)=IBLANK
   ICHPRE(15)=IBLANK
   ICHPRE(16)=IBLANK
   ICHPRE(17)=IBLANK
   ICHPRE(18)=IBLANK
   ICHPRE(19)=IBLANK
   ICHPRE(20)=IBLANK
   ICHPRE(21)=IBLANK
   ICHPRE(22)=IBLANK
   ICHPRE(23)=IBLANK
   ICHPRE(24)=IBLANK
   ICHPRE(25)=IBLANK
   ICHPRE(26)=IBLANK
   ICHPRE(27)=IBLANK
   ICHPRE(28)=IBLANK
   ICHPRE(29)=IBLANK
   ICHPRE(30)=IBLANK
   ICHPRE(31)=IBLANK
   ICHPRE(32)=IBLANK
   ICHPRE(33)=IBLANK
   ICHPRE(34)=IBLANK
   ICHPRE(35)=IBLANK
   ICHPRE(36)=IBLANK
   ICHPRE(37)=IBLANK
   ICHPRE(38)=IBLANK
   ICHPRE(39)=IBLANK
   ICHPRE(40)=IBLANK
   ICHPRE(41)=IBLANK
   ICHPRE(42)=IBLANK
   ICHPRE(43)=IBLANK
   ICHPRE(44)=IBLANK
   ICHPRE(45)=IBLANK
   ICHPRE(46)=IBLANK
   ICHPRE(47)=IBLANK
   ICHPRE(48)=IBLANK
   ICHPRE(49)=IBLANK
   ICHPRE(50)=IBLANK
   ICHPRE(51)=IBLANK
   ICHPRE(52)=IBLANK
   ICHPRE(53)=IBLANK
   ICHPRE(54)=IBLANK
   ICHPRE(55)=IBLANK
   ICHPRE(56)=IBLANK
   ICHPRE(57)=IBLANK
   ICHPRE(58)=IBLANK
   ICHPRE(59)=IBLANK
   ICHPRE(60)=IBLANK
   ICHPRE(61)=IBLANK
   ICHPRE(62)=IBLANK
   ICHPRE(63)=IBLANK
   ICHPRE(64)=IBLANK
   ICHPRE(65)=IBLANK
   ICHPRE(66)=IBLANK
   ICHPRE(67)=IBLANK
   ICHPRE(68)=IBLANK
   ICHPRE(69)=IBLANK
   ICHPRE(70)=IBLANK
   ICHPRE(71)=IBLANK
   ICHPRE(72)=IBLANK
   ICHPRE(73)=IBLANK
   ICHPRE(74)=IBLANK
   ICHPRE(75)=IBLANK
   ICHPRE(76)=IBLANK
   ICHPRE(77)=IBLANK
   ICHPRE(78)=IBLANK
   ICHPRE(79)=IBLANK
   ICHPRE(80)=IBLANK
   ICHPRE(81)=IBLANK
   ICHPRE(82)=IBLANK
   ICHPRE(83)=IBLANK
   ICHPRE(84)=IBLANK
   ICHPRE(85)=IBLANK
   ICHPRE(86)=IBLANK
   ICHPRE(87)=IBLANK
   ICHPRE(88)=IBLANK
   ICHPRE(89)=IBLANK
   ICHPRE(90)=IBLANK
   ICHPRE(91)=IBLANK
   ICHPRE(92)=IBLANK
   ICHPRE(93)=IBLANK
   ICHPRE(94)=IBLANK
   ICHPRE(95)=IBLANK
   ICHPRE(96)=IBLANK
   ICHPRE(97)=IBLANK
   ICHPRE(98)=IBLANK
   ICHPRE(99)=IBLANK
   ICHPRE(100)=IBLANK

```

```

23 CONTINUE
24 CONTINUE
C:TOP 4 ANY ITEM NAME ENTRIES. PRINT ERROR MSG AND STCP.
   N5=66
   GO TO 196
25 CONTINUE
C: ITEM NAME AS YET UNDECLARED.
   J=INTRV+3
   LISTN(J-2)=ICPRE(1)
   LISTN(J-1)=ICPRE(2)
   LISTN(J)=ICPRE(3)
   IF(KCOMMON)NE.0) GO TO 142
   IF(NTYPE)EQ.1) GO TO 142
   IF(N4)EQ.1) GO TO 142
   CONTINUE
   GO TO(45,62,66,111),N4
27 CONTINUE
   N2=N2+1
   IF(N2)LE.N3) GO TO 15
C:NO MATCH.
   IF(N2)EQ.38) GO TO 126
   IF(N2)EQ.39) GO TO 126
   IF(N2)EQ.40) GO TO 126
   IF(N2)EQ.41) GO TO 126
   IF(N2)EQ.42) GO TO 126
   IF(N2)EQ.43) GO TO 126
   IF(N2)EQ.44) GO TO 126
   IF(N2)EQ.45) GO TO 126
   IF(N2)EQ.46) GO TO 126
   IF(N2)EQ.47) GO TO 126
   IF(N2)EQ.48) GO TO 126
   IF(N2)EQ.49) GO TO 126
   IF(N2)EQ.50) GO TO 126
   IF(N2)EQ.51) GO TO 126
   IF(N2)EQ.52) GO TO 126
   IF(N2)EQ.53) GO TO 126
   IF(N2)EQ.54) GO TO 126
   IF(N2)EQ.55) GO TO 126
   IF(N2)EQ.56) GO TO 126
   IF(N2)EQ.57) GO TO 126
   IF(N2)EQ.58) GO TO 126
   IF(N2)EQ.59) GO TO 126
   IF(N2)EQ.60) GO TO 126
   IF(N2)EQ.61) GO TO 126
   IF(N2)EQ.62) GO TO 126
   IF(N2)EQ.63) GO TO 126
   IF(N2)EQ.64) GO TO 126
   IF(N2)EQ.65) GO TO 126
   IF(N2)EQ.66) GO TO 126
   IF(N2)EQ.67) GO TO 126
   IF(N2)EQ.68) GO TO 126
   IF(N2)EQ.69) GO TO 126
   IF(N2)EQ.70) GO TO 126
   IF(N2)EQ.71) GO TO 126
   IF(N2)EQ.72) GO TO 126
   IF(N2)EQ.73) GO TO 126
   IF(N2)EQ.74) GO TO 126
   IF(N2)EQ.75) GO TO 126
   IF(N2)EQ.76) GO TO 126
   IF(N2)EQ.77) GO TO 126
   IF(N2)EQ.78) GO TO 126
   IF(N2)EQ.79) GO TO 126
   IF(N2)EQ.80) GO TO 126
   IF(N2)EQ.81) GO TO 126
   IF(N2)EQ.82) GO TO 126
   IF(N2)EQ.83) GO TO 126
   IF(N2)EQ.84) GO TO 126
   IF(N2)EQ.85) GO TO 126
   IF(N2)EQ.86) GO TO 126
   IF(N2)EQ.87) GO TO 126
   IF(N2)EQ.88) GO TO 126
   IF(N2)EQ.89) GO TO 126
   IF(N2)EQ.90) GO TO 126
   IF(N2)EQ.91) GO TO 126
   IF(N2)EQ.92) GO TO 126
   IF(N2)EQ.93) GO TO 126
   IF(N2)EQ.94) GO TO 126
   IF(N2)EQ.95) GO TO 126
   IF(N2)EQ.96) GO TO 126
   IF(N2)EQ.97) GO TO 126
   IF(N2)EQ.98) GO TO 126
   IF(N2)EQ.99) GO TO 126
   IF(N2)EQ.100) GO TO 126

```



```

I=MAP(I)
IF(I.NE.42)GO TO 3C
C:DECIMAL PCINT. 42) GO TO 3C
IF(N3.NE.0)GO TO 22
N3=N1
GO TO 25
CONTINUE
IF(I.GT.5)GO TO 21
N2=10*N2+I
GO TO 29
CONTINUE
C:END OF NUMBER.
N1=N1-1
IF(N1.NE.N3)GO TO 33
C:NUMBER ENDS IN DECIMAL POINT.
IF(N1.EQ.NREAD)GO TO 33
I=LBYTE(N1+1)
IF(MAP(I).LT.1C.OR.MAP(I).GT.35)
GO TO 22
C:DECIMAL GOES WITH SOMETHING ELSE.
N3=0
CONTINUE
N1=N1-1
CONTINUE
IF(N3.EQ.0)GO TO 27
C:NUMBER CONTAINS DECIMAL POINT.
IF(LOOP.EQ.N1)GO TO 192
C:DECIMAL POINT ONLY IF TRUE.
IF(IEQ.EQ.0)KINDL=1
IF(IEQ.NE.0)KINDR=1
CONTINUE
N1=N1+1
NREAD=NREAD+1
I=VREAD
CONTINUE
I=I-1
IF(I.LT.N1)GO TO 36
LBYTE(I+1)=LBYTE(I)
GO TO 35
C:INSERT L AT END OF LITERAL FLOATING PCINT NUMBER.
CONTINUE
LBYTE(N1)=KARSET(21)
C:INSERT TRAILING ZERO IF NECESSARY.
IF(N1.NE.N3+1)GO TO 37
LBYTE(N1)=KARDUM
GO TO 34
CONTINUE
GO TO(40,55,66,45),N4
C:FORTRAN LINE READY. SCAN FROM COLUMN CNE.
INCR=0
IGN=0
KINDL=0
KINDR=0
IDPTH=0
IFQ=0
IFST=0
KOMMON=0
JWRK=0
NUMEND=0

```

00036100
00036200
00036300
00036400
00036500
00036600
00036700
00036800
00036900
00037000
00037100
00037200
00037300
00037400
00037500
00037600
00037700
00037800
00037900
00038000
00038100
00038200
00038300
00038400
00038500
00038600
00038700
00038800
00038900
00039000
00039100
00039200
00039300
00039400
00039500
00039600
00039700
00039800
00039900
00040000
00040100
00040200
00040300
00040400
00040500
00040600
00040700
00040800
00040900
00041000
00041100
00041200
00041300
00041400
00041500
00041600
00041700
00041800
00041900
00042000

```

00042100
00042200
00042300
00042400
00042500
00042600
00042700
00042800
00042900
00043000
00043100
00043200
00043300
00043400
00043500
00043600
00043700
00043800
00043900
00044000
00044100
00044200
00044300
00044400
00044500
00044600
00044700
00044800
00044900
00045000
00045100
00045200
00045300
00045400
00045500
00045600
00045700
00045800
00045900
00046000
00046100
00046200
00046300
00046400
00046500
00046600
00046700
00046800
00046900
00047000
00047100
00047200
00047300
00047400
00047500
00047600
00047700
00047800
00047900
00048000

```

```

ICHR=LBYTE(I)
IF(MAP(ICHR) .NE. 12) GO TO 39
C: COMMENT BRACKET WITH CHARACTER S.
LBYTE(1)=KARSET(39)
LBYTE(2)=KARSET(38)
LBYTE(70)=KARSET(36)
LBYTE(71)=KARSET(35)
VREAD=71
GO TO 165
CONTINUE
IF(MAP(ICHR) .GT. 9) GO TO 42
C: LABEL PREFIX A Q AND SUFFIX A COLON.
LJOP=1
N4=1
GO TO 28
CONTINUE(N1+1) .NE. KARSET(50)) GO TO 192
IF(LBYTE(N1+1)=KARSET(46))
C: CHECK DO FOR PROPER END STATEMENTS.
41 CONTINUE
IF(NUMDO .EQ. C .OR. LABDO(NUMDO) .NE. N2) GO TO 44
C: COUNT NUMBER OF ENDS TO APPEND.
NUMDO=NUMDO-1
NUMEND=NUMEND+1
GO TO 41
CONTINUE
IF(MAP(ICHR) .EQ. 50) GO TO 43
C: DON'T TRANSLATE LINE, DELETE 'B' PREFIX.
IF(MTYPE .GE. 3) GO TO 7
IF(MAP(ICHR) .EQ. 11) LBYTE(1)=KARSET(50)
GO TO 193
CONTINUE
C: FORTRAN STATEMENT FIELD.
IF(LBYTE(6) .NE. KARSET(50)) GC TO 192
44 CONTINUE
N1=6
CONTINUE
N5=1
LJOP=N1+1
C: SCAN VARIABLE FOR CRACKING THE FORTRAN LINE.
IF(LJOP .LE. NREAD) GO TO 51
C: END OF FORTRAN LINE REACHED.
IF(N5 .NE. 1) GO TO 192
IF(KOMMON .NE. 0) GO TO 7
IF(IEQ .EQ. 0 .OR. KINDL .EQ. KINOR) GO TO 49
C: FLOAT TO FIX CONVERSION (OR VICE VERSA) NECESSARY.
N1=IEQ+1
N2=0
N3=0
N4=K INDL +51
N5=10
GO TO 152
CONTINUE
N1=NREAD+1
N2=0
N4=K INDL +53
IF(NUMEND .EQ. 0) GO TO 73
CONTINUE
N3=2
48

```

```

N5= 1
GO TO 152
C: APPEND ENDS.
49 CONTINUE
IF (NUMENC .EQ. 0) GO TO 164
N4= 45
CONTINUE
NUMENC=NUMEND-1
N1= NREAD+1
N2= 0
GO TO 48
CONTINUE
N1= LOOP
I= L BYTE (LOOP)
I= MAP (I)
IF (I .EQ. 50) GO TO 46
C: SKIP BLANKS
GO TO (52, 61, 63, 65, 66, 58, 75
, 83, 106, 80, 143, 145, 58), N5
C: LOOP VARIABLE OF SCANNER SET TO NEXT NON-PLANK.
52 CONTINUE
IF (IBGN .NE. 0) GO TO 125
IF (I .LT. 10 .OP. I .GT. 35) GO TO 192
IBGN= LOOP
C: FIRST CHARACTER IN STATEMENT FIELD MUST BE A LETTER.
53 CONTINUE
N2= 1
N3= 30
N4= 1
GO TO 16
CONTINUE
C: BASIC WORD LIST SWITCH. SWITCH POINTS ARE FOR THE FOLLOWING:
C: CONTINUE, DO, IF, COMMON, CALL, SUBROUTINE, FORMAT, READ, WRITE, PAUSE,
C: STOP, PROC, END, ENDFILE, GO, GO TO, 48 S, IAR S, A LOG, SORT, SIN, CCS,
C: RETURN
GO TO (55, 57, 74, 96, 57, 101, 152
, 152, 152, 150, 55, 164, 145, 152
, 103, 105, 117, 116, 115, 115
, 120, 120, 150), N2
55 CONTINUE
C: CONTINUE OR STOP. ERASE IT.
IF (N1 .NE. NREAD) GO TO 152
DO 56 I= LOOP, N1
L BYTE (I)= KAR SET (50)
CONTINUE
56 IF (NUMENC .EQ. C .OR. N2 .NE. 1) GO TO 165
NREAD= LOOP-1
N4= 13
GO TO 50
CONTINUE
C: DO STATEMENT.
N5= 6
GO TO 46
CONTINUE
IF (I .GT. 5) GO TO 152
N4= 2
GO TO 28
C: STORE THE LABEL.
59 CONTINUE
DO 60 I= LOOP, N1

```

```

00048100
00048200
00048300
00048400
00048500
00048600
00048700
00048800
00048900
00049000
00049100
00049200
00049300
00049400
00049500
00049600
00049700
00049800
00049900
00050000
00050100
00050200
00050300
00050400
00050500
00050600
00050700
00050800
00050900
00051000
00051100
00051200
00051300
00051400
00051500
00051600
00051700
00051800
00051900
00052000
00052100
00052200
00052300
00052400
00052500
00052600
00052700
00052800
00052900
00053000
00053100
00053200
00053300
00053400
00053500
00053600
00053700
00053800
00053900
00054000

```

```

60      LBYTE(I)=KARSET(50)
        CONTINUE
        NJ=DO=NUM DO +1
        LAB DO (NUM DO)=N2
        N5=2
        GO TO 46
C:CHECK DO NAME FOR POSSIBLE CONFLICT.
61      CONTINUE
        IF(I .LT. 10 .OR. I .GT. 35) GO TO 192
        N2=26
        N3=30
        N4=2
        GO TO 16
        CONTINUE
        N5=3
C:CHECK FOR EQUAL SIGN.
62      CONTINUE
        GO TO 46
63      IF(I .NE. 44) GO TO 152
64      CONTINUE
        N5=4
        GO TO 46
65      CONTINUE
        IF(I .GT. 35) GO TO 152
        N4=3
        IF(I .LT. 10) GO TO 28
        N2=26
        N3=30
        GO TO 16
        CONTINUE
        IF(JMRK) 71,67,7C
66      CONTINUE
        N5=5
C:CHECK COMMA IN LABEL LIST.
68      CONTINUE
        GO TO 46
        IF(I .NE. 45) GO TO 152
C:INSERT BY I TO
        LBYTE(LOOP)=KARSET(50)
        IF(JMRK .NE. 0) GO TO 64
        JMRK=LOOP+4
        N2=0
        N3=2
        N4=44
        N5=5
        GO TO 152
69      CONTINUE
        N1=LOOP+8
        GO TO 64
70      CONTINUE
        IF(N1 .EQ. NREAD) GO TO 164
        JMRK=-JMRK
        GO TO 67
71      CONTINUE
        IF(N1 .NE. NREAD) GO TO 192
        I=I+30
        I=I-N1-LOOP
        INDEX=IMLBYT+LOOP-I
        CALL SBYTE
        DO 72 I=LOOP,N1
00054100
00054200
00054300
00054400
00054500
00054600
00054700
00054800
00054900
00055000
00055100
00055200
00055300
00055400
00055500
00055600
00055700
00055800
00055900
00056000
00056100
00056200
00056300
00056400
00056500
00056600
00056700
00056800
00056900
00057000
00057100
00057200
00057300
00057400
00057500
00057600
00057700
00057800
00057900
00058000
00058100
00058200
00058300
00058400
00058500
00058600
00058700
00058800
00058900
00059000
00059100
00059200
00059300
00059400
00059500
00059600
00059700
00059800
00059900
00060000

```

```

72  LBYTE(I)=KARSET(50)
    CONTINUE
    NREAD=LOOP-2
    C: PUT T HSGDR(40)=IBYT1
        HSGDR(40)=IBYT2
        N1=IABS(JMRK)
        N2=1
        N4=40
        CONTINUE
        N3=2
        GO TO 95
    C: IF STATEMENT
        IDPTH=IDPTH+1
        IF(IDPTH.GT.5) GO TO 152
        KINDST(IDPTH)=1
        CONTINUE
        NPAREN(IDPTH)--1
        GO TO 46
        CONTINUE
    C: LEFT
        IF(IDPTH.EQ.0) GO TO 46
        IF(NPAREN(IDPTH)) 77,152,78
        CONTINUE
        NPAREN(IDPTH)=0
        CONTINUE
        NPAREN(IDPTH)=NPAREN(IDPTH)+1
        GO TO 46
        CONTINUE
    C: RIGHT
        LSTPRN=LOOP
        IF(IDPTH.EQ.0) GO TO 46
        IF(NPAREN(IDPTH).LE.0) GO TO 152
        NPAREN(IDPTH)=NPAREN(IDPTH)-1
        IF(NPAREN(IDPTH).NE.0) GO TO 46
        C: FOUND THE MATCHING CLOSE PAREN.
        IF(KINDST(IDPTH).GE.3) GO TO 121
        N5=10
        IF(N1.LT.NREAD) GO TO 46
        GO TO 192
        CONTINUE
    C: END OF IF CLAUSE
    C: INSERT WHEN CLAUSE.
        N2=0
        N3=2
        N4=46
        N5=3
        GO TO 152
        CONTINUE
        N1=LOOP+4
        LBYTE(N1)=KARSET(50)
        IFST=N1
        IEO=0
        JMRK=KINDL
        KINDL=0
        KINDR=0
        IF(IDPTH.NE.1) GO TO 192
        IDPTH=0

```

```

00060100
00060200
00060300
00060400
00060500
00060600
00060700
00060800
00060900
00061000
00061100
00061200
00061300
00061400
00061500
00061600
00061700
00061800
00061900
00062000
00062100
00062200
00062300
00062400
00062500
00062600
00062700
00062800
00062900
00063000
00063100
00063200
00063300
00063400
00063500
00063600
00063700
00063800
00063900
00064000
00064100
00064200
00064300
00064400
00064500
00064600
00064700
00064800
00064900
00065000
00065100
00065200
00065300
00065400
00065500
00065600
00065700
00065800
00065900
00066000

```

```

00066100
00066200
00066300
00066400
00066500
00066600
00066700
00066800
00066900
00067000
00067100
00067200
00067300
00067400
00067500
00067600
00067700
00067800
00067900
00068000
00068100
00068200
00068300
00068400
00068500
00068600
00068700
00068800
00068900
00069000
00069100
00069200
00069300
00069400
00069500
00069600
00069700
00069800
00069900
00070000
00070100
00070200
00070300
00070400
00070500
00070600
00070700
00070800
00070900
00071000
00071100
00071200
00071300
00071400
00071500
00071600
00071700
00071800
00071900
00072000

```

```

C: RESET POINTERS FOR STATEMENT FOLLOWING IF STATEMENT.
IF (KINDST(I)) .NE. 1) GO TO 45
C: IF ABOVE NOT TRUE, THEN IT IS AN ARITHMETIC IF FORM.
J=0
DO 82 I=1, 5
K= ICHK(I)
IF (MAP(K)) .EQ. 5) GO TO 42
IF (MAP(K)) .GT. 5) GO TO 152
J= 10+J+MAP(K)
CONTINUE
IF (J .EQ. 0) GO TO 152
C: J IS THE NEXT LABEL AFTER THE IF.
ICLK(1)=J
ICLK(4)=0
ICLK(5)=0
ICLK(6)=0
JCNT=4
N5=8
GO TO 46
CONTINUE NUMBERS.
C: COLLECT LABEL NUMBERS.
IF (I .NE. 45) GO TO 44
JCNT=JCNT+1
I=0
CONTINUE
IF (I .GT. 5) GO TO 152
ICLK(JCNT)= 10+ICLK(JCNT)+1
IF (IN I .L. NREAD) GO TO 46
IF (JCNT .NE. 6) GO TO 152
C: LABELS NOW ARE STORED IN ICHK.
DO 85 I=1, N5
LBYTE(I)=KARSET(50)
CONTINUE
NREAD=IFST
C: REMOVE LABELS FROM STATEMENT.
N1=LSTPRN
N4=JMRK+47
C: INSERT COMPARISON WITH ZERO INTO STATEMENT.
86 CONTINUE
N2=0
N3=2
N5=4
JCNT=N1
GO TO 152
CONTINUE
IF (JCNT .NE. LSTPRN) GO TO 48
JMRK=NREAD
C: INSERT GOTO
N1=NREAD+1
GO TO 86
CONTINUE
LSTPRN=LSTPRN+1
IF (IFST .GT. 150) GO TO 152
DO 89 I=1, 150
IF (INE(I)=LBYTE(I))
CONTINUE
JFLG=0
JCNT=3

```

```

90      CONTINUE
      JSH=2
      JCNT=JCNT+1
      IF(JCNT.GT.6) GO TO 7
C:CHECK IF LABEL WITH NEXT LINE LABEL.
      IF(ICHK(3).NE.ICHK(JCNT)) GO TO 91
      JFLG=1
      GO TO 90
91      CONTINUE INTO LBYTE.
C:RESTORE LINE INTO LBYTE.
      DO 92 I=1,150
      LBYTE(I)=IFLIME(I)
      IF(I.LE.5) IFLINE(I)=KARSET(50)
92      CONTINUE
      NREAD=150
C:TEST FOR SIMPLE GOTO DEFAULT CASE.
      IF(JCNT.EQ.6.AND.JFLG.EQ.0) GO TO 93
C:INSERT PROPER RELATIONAL.
      N1=LSTPRN
      N2=N1+1
      N3=2
      N4=JCNT+37
      N5=6
      GO TO 152
93      CONTINUE
      N1=7
      N2=JMRK
      N3=0
      N5=6
      GO TO 152
94      CONTINUE
C:APPEND FINAL LABEL.
      N1=NREAD+1
      N2=0
      N3=1
      N4=ICHK(JCNT)
      JSH=3
      CONTINUE
      N5=7
      GO TO 152
95      CONTINUE
C:COMMON STATEMENT.
      KCOMMON=1
      GO TO 45
96      CONTINUE
C:CALL STATEMENT.CHECK FOR LDBYTE, SBYTE AND MIOP CALLS.
      IF(NTYPE.NE.1) GO TO 164
      N5=12
      CONTINUE
      N2=66
      N3=70
      GO TO 16
97      CONTINUE
      N1=NE
      NFN1(1)=76
      NFN1(2)=71
      NFN1(3)=N2+5
      NFN1(4)=78
      NFN1(5)=0

```

```

00072100
00072200
00072300
00072400
00072500
00072600
00072700
00072800
00072900
00073000
00073100
00073200
00073300
00073400
00073500
00073600
00073700
00073800
00073900
00074000
00074100
00074200
00074300
00074400
00074500
00074600
00074700
00074800
00074900
00075000
00075100
00075200
00075300
00075400
00075500
00075600
00075700
00075800
00075900
00076000
00076100
00076200
00076300
00076400
00076500
00076600
00076700
00076800
00076900
00077000
00077100
00077200
00077300
00077400
00077500
00077600
00077700
00077800
00077900
00078000

```

```

100      JCNT=0
      CJNT INUE
      JCNT=2
      JCNT=JCNT+1
      IF(NFN(JCNT) .EQ. C) GO TO 7
      JSM=5
      NS=NFN(JCNT)
      GO TO 195
      CJNT INUE
      INTRY=0
      IFRTRN=0
      N2=N1
      N1=2
      N3=0
      N5=8
      GO TO 152
102      CJNT INUE
      LBYTE(NR EAD+1)=KARSET(46)
      N1=NR EAD+3
      N2=0
      N4=12
      C: INSERT PROC 73
      GO TO 73
103      CJNT INUE
      C: GO COMMAND, SQUEEZE INTO GOTO.
      I=LBYTE(N1+1)
      IF(MAPI) .NE. 50) GO TO 152
      J=N1+2
      DJ 104 I=J,NREAD
      LBYTE(I-1)=LBYTE(I)
      CJNT INUE
      C: CLOSE THE SPACE BETWEEN GO AND TO AND TRY AGAIN.
      NREAD=NR EAD-1
      N1=LOOP-1
      GO TO 45
105      CJNT INUE
      C: GOTO (VOM PROPERLY SQUEEZED).
      NGOTO=LOOP
      N5=9
      GO TO 46
106      CJNT INUE
      C: S IMPL E GOTO. PREFIX THE LABEL WITH A C.
      IF(I .GT. 9) GO TO 107
      N2=0
      N4=49
      GO TO 73
107      CJNT INUE
      C: COMPUTED GOTO. CHANGE TO SWITCH.
      IF(I .NE. 40) GO TO 152
      N1=NGOTO
      N2=LOOP
      N3=2
      N4=50
      N5=9
      GO TO 152
108      CJNT INUE
      C: MOVE THE ITEM FROM END INTO PARENS.
      NGOTO=NGOTO+12
      JCNT=NR EAD

```

```

00078100
00078200
00078300
00078400
00078500
00078600
00078700
00078800
00078900
00079000
00079100
00079200
00079300
00079400
00079500
00079600
00079700
00079800
00079900
00080000
00080100
00080200
00080300
00080400
00080500
00080600
00080700
00080800
00080900
00081000
00081100
00081200
00081300
00081400
00081500
00081600
00081700
00081800
00081900
00082000
00082100
00082200
00082300
00082400
00082500
00082600
00082700
00082800
00082900
00083000
00083100
00083200
00083300
00083400
00083500
00083600
00083700
00083800
00083900
00084000

```



```

000A4100
000A4200
000A4300
000A4400
000A4500
000A4600
000A4700
000A4800
000A4900
000A5000
000A5100
000A5200
000A5300
000A5400
000A5500
000A5600
000A5700
000A5800
000A5900
000A6000
000A6100
000A6200
000A6300
000A6400
000A6500
000A6600
000A6700
000A6800
000A6900
000A7000
000A7100
000A7200
000A7300
000A7400
000A7500
000A7600
000A7700
000A7800
000A7900
000A8000
000A8100
000A8200
000A8300
000A8400
000A8500
000A8600
000A8700
000A8800
000A8900
000A9000
000A9100
000A9200
000A9300
000A9400
000A9500
000A9600
000A9700
000A9800
000A9900
000A9000

```

```

109 JMRK=0
      CJNT INUE
      JCNT=JCNT-1
      IF(JCNT.LE.LOOP) GO TO 192
      I=LBYTE(JCNT)
      I=MAP(I)
      IF(I.EQ.41) GO TO 110
      IF(I.LE.35) GO TO 109
      IF(JMRK.EQ.0) JMRK=JCNT+1
      IF(I.NE.45.AND.I.NE.50) GO TO 192
      LBYTE(JCNT)=KARSET(50)
      THE COMMA.
      C: BLANK GO TO 109
110 CJNT INUE
      IF(JMRK.EQ.0.OR.NREAD-JMRK.GT.5)
        GO TO 152
      LTOP=JMRK
      N2=26
      N3=30
      N4=4
      GO TO 16
      CJNT INUE
      DO 112 I=JMRK,NREAD
      LBYTE(NGO TO)=LBYTE(I)
      LBYTE(I)=KARSET(50)
      NGOTO=NGO TO+1
      CJNT INUE
      NREAD=JCNT
      C: PREFIX ALL LABELS WITH A Q.
113 CJNT INUE
      J=0
114 CJNT INUE
      NGOTO=NGO TO+1
      IF(NGOTO.GE.NREAD) GO TO 164
      I=LBYTE(NGO TO)
      IF(MAP(I).GT.9) GO TO 113
      IF(J.NE.0) GO TO 114
      LABEL TO GET Q PREFIX.
      C: NEXT
115 I=VR EAD
      CJNT INUE
      LBYTE(I+1)=LBYTE(I)
      I=I-1
      IF(I.GE.NGO TO) GO TO 115
      LBYTE(NGO TO)=KARSET(26)
      NREAD=NREAD+1
      J=J+1
      GO TO 114
      CJNT INUE
      C: I ASS BLANK FIRST LETTER.
      LBYTE(LOOP)=KARSET(50)
117 CJNT INUE
      IF(I.EQ.50) GO TO 46
      C: CHECK FOR PRECEDING MINUS SIGN FOR '1' FIX.
      N3=LOOP
      N4=3
      CJNT INUE
      N3=N3-1
      I=LBYTE(N3)
      IF(MAP(I).EQ.50) GO TO 116
      IF(MAP(I).NE.N4) GO TO 46

```

```

00090100
00090200
00090300
00090400
00090500
00090600
00090700
00090800
00090900
00091000
00091100
00091200
00091300
00091400
00091500
00091600
00091700
00091800
00091900
00092000
00092100
00092200
00092300
00092400
00092500
00092600
00092700
00092800
00092900
00093000
00093100
00093200
00093300
00093400
00093500
00093600
00093700
00093800
00093900
00094000
00094100
00094200
00094300
00094400
00094500
00094600
00094700
00094800
00094900
00095000
00095100
00095200
00095300
00095400
00095500
00095600
00095700
00095800
00095900
00096000

```

```

N4=N4+7
IF(N4.EQ.44) GO TO 118
C: PUT IN THE CORRECTION.
J CNT=NREAD-N1
N1=LOOP
N4=N2+67
N2=0
N3=2
GO TO 151
CONTINUE

```

```

119 C:ALOG, SQRT, SIN, COS FUNCTION CALLS.
IDPTH=IDPTH+1
IF(IDPTH.GT.5) GO TO 192
KINDST(IDPTH)=3
NFN1(IDPTH)=LOOP
NFN2(IDPTH)=N1+1
IF(N2.EQ.20) GO TO 75
C: CHANGE ALOG TO LN.

```

```

120 CONTINUE
GO TO 75
C: S IN, JCNT=NRREAD-N1
JCNT=NRREAD-N1
N2=0
N3=2
N4=55
GO TO 151
CONTINUE

```

```

121 C: END OF CALLING ARGUMENT FOR ALOG, SORT, SIN, CCS.
J CNT=NREAD-N1
N1=LNSTPRN
N2=0
N3=2
GO TO 151
CONTINUE

```

```

C: ADD ENDING CHARACTERS.
N3=2
N4=52
N5=12
GO TO 152
CONTINUE

```

```

122 N1=NFN2(IDPTH)
N2=0
N3=2
N4=57
C: INSERT (SHORT
N5=13
GO TO 152
CONTINUE

```

```

123 N1=NFN1(IDPTH)
IDPTH=IDPTH-1
N2=0
N3=2
N4=52
C: INSERT LONG(
GO TO 151
CONTINUE

```

```

124 N1=NRREAD-JCNT

```

```

125      GO TO 45
        CONTINUE
C:SCANING RIGHT OF THE FIRST CHARACTER.
        IF I .GT. 9) GO TO 127
C:CHECK THE NUMBER.
126      CONTINUE
        N4=4
        GO TO 28
        CONTINUE
C:CHECK THE NAME.
        IF I .GT. 25) GO TO 128
        N2=15
        GO TO 53
        CONTINUE
        IF I .EQ. 38) GO TO 13C
        IF I .EQ. 39) GO TO 131
        IF I .EQ. 40) GO TO 76
        IF I .EQ. 41) GO TO 75
        IF I .EQ. 44) IEO=LOOP
        IF I .NE. 42) GO TO 46
        IF (LOOP .GT. NREAD-3) GO TO 126
C:TEST FOR RELATIONAL.
        N2=31
        N3=38
        GO TO 16
        CONTINUE
C:RELATIONAL FOUND. DROP PERIODS.
        N1=N1+1
        IF (MAP(I) .NE. 42) GO TO 152
        LBYTE(N1)=KARSET(50)
        LBYTE(LOOP)=KARSET(50)
        IF (N2 .LE. 36) GO TO 45
        LBYTE(N1)=KARSET(40)
        LBYTE(LOOP)=KARSET(41)
        GO TO 45
        CONTINUE
C:ASTERISK. CHECK FOR DOUBLE.
        IF (LBYTE(N1-1)
        IF (MAP(I) .NE. 3E) GO TO 46
        N5=83
        GO TO 195
        CONTINUE
C:AND. KINDL .EQ. 0 .AND. KINDL .EQ. C .OR. IEO .NE. 0
        IF (IEO .AND. KINDR .EQ. 0) GO TO 46
C:ADD TRUNC FUNCTION TO FIX DIVIDE SCALING PROBLEM.
        JCNT=NREAD-N1
        J4PK=N1
        N3=0
        N4=0
        N5=1
        CONTINUE
        J4PK=J4PK-1
        IF (J4PK .LE. 18GN) GO TO 152
        I=LBYTE(J4PK)
        I=LBYTE(I)
        IF (I=N5 .EQ. 50) GO TO 132
        TRAILING BLANKS.
        N5=0

```

```

00096100
00096200
00096300
00096400
00096500
00096600
00096700
00096800
00096900
00097000
00097100
00097200
00097300
00097400
00097500
00097600
00097700
00097800
00097900
00098000
00098100
00098200
00098300
00098400
00098500
00098600
00098700
00098800
00098900
00099000
00099100
00099200
00099300
00099400
00099500
00099600
00099700
00099800
00099900
00100000
00100100
00100200
00100300
00100400
00100500
00100600
00100700
00100800
00100900
00101000
00101100
00101200
00101300
00101400
00101500
00101600
00101700
00101800
00101900

```

```

C:OPEN      IF(I .NE. 40) GO TO 134
            PAR EN.
            N3=N3-1
            IF(N3) 137, 133, 132
            CONTINUE
C:CHECK     N4=JMRK
            N5=1
            GO TO 132
            CONTINUE
            IF(I .NE. 41) GO TO 135
C:CLOSE     PAR EN.
            IF(N5 .LT. 0) GO TO 137
            N3=N3+1
            GO TO 132
            CONTINUE
            IF(N3 .NE. 0 .OR. I .EQ. 42) GO TO 132
C:CONTINUE INJE FOR PERIOD OR WITHIN PARENS.
            CONTINUE
            IF(I .GT. 35) GO TO 137
C:INTJ      A NAME OR NUMBER.
            IF(N4 .NE. 0) N4=JMRK
            N5=1
            GO TO 132
            CONTINUE
            BEGINNING OF TRUNC FIELD BY JMRK.
            JMRK=JMRK+1
            IF(N4 .NE. 0) JMRK=N4
            N2=0
            N3=2
            N4=81
            N5=11
            GO TO 152
            CONTINUE
            N1=JMRK
            N2=0
            N3=2
            N4=82
            GO TO 151
            CONTINUE
C:JNDECLAR D ITEM NAME ERROR.
            N5=63
            NUMERR=NUMERR+1
            CONTINUE
            JSW=4
            GO TO 156
            CONTINUE
            JSW=2
            GO TO 45
            CONTINUE
            DECLARATION.
            JYRK=LOOP
            JCNT=N1
            IF(N1 .GE. NREAD) GO TO 146
            N5=11
            GO TO 46
            CONTINUE
            IF(I .EQ. 40) GO TO 144
            N1=JCNT
            GO TO 146
133
C:CHECK     N4=JMRK
            N5=1
            GO TO 132
            CONTINUE
            IF(I .NE. 41) GO TO 135
C:CLOSE     PAR EN.
            IF(N5 .LT. 0) GO TO 137
            N3=N3+1
            GO TO 132
            CONTINUE
            IF(N3 .NE. 0 .OR. I .EQ. 42) GO TO 132
C:CONTINUE INJE FOR PERIOD OR WITHIN PARENS.
            CONTINUE
            IF(I .GT. 35) GO TO 137
C:INTJ      A NAME OR NUMBER.
            IF(N4 .NE. 0) N4=JMRK
            N5=1
            GO TO 132
            CONTINUE
            BEGINNING OF TRUNC FIELD BY JMRK.
            JMRK=JMRK+1
            IF(N4 .NE. 0) JMRK=N4
            N2=0
            N3=2
            N4=81
            N5=11
            GO TO 152
            CONTINUE
            N1=JMRK
            N2=0
            N3=2
            N4=82
            GO TO 151
            CONTINUE
C:JNDECLAR D ITEM NAME ERROR.
            N5=63
            NUMERR=NUMERR+1
            CONTINUE
            JSW=4
            GO TO 156
            CONTINUE
            JSW=2
            GO TO 45
            CONTINUE
            DECLARATION.
            JYRK=LOOP
            JCNT=N1
            IF(N1 .GE. NREAD) GO TO 146
            N5=11
            GO TO 46
            CONTINUE
            IF(I .EQ. 40) GO TO 144
            N1=JCNT
            GO TO 146
138
139
C:JNDECLAR D ITEM NAME ERROR.
            N5=63
            NUMERR=NUMERR+1
            CONTINUE
            JSW=4
            GO TO 156
            CONTINUE
            JSW=2
            GO TO 45
            CONTINUE
            DECLARATION.
            JYRK=LOOP
            JCNT=N1
            IF(N1 .GE. NREAD) GO TO 146
            N5=11
            GO TO 46
            CONTINUE
            IF(I .EQ. 40) GO TO 144
            N1=JCNT
            GO TO 146
140
141
142
C:ITEM
            N5=11
            GO TO 46
            CONTINUE
            IF(I .EQ. 40) GO TO 144
            N1=JCNT
            GO TO 146
143
            N5=11
            GO TO 46
            CONTINUE
            IF(I .EQ. 40) GO TO 144
            N1=JCNT
            GO TO 146

```

```

00 102100
00 102200
00 102300
00 102400
00 102500
00 102600
00 102700
00 102800
00 102900
00 103000
00 103100
00 103200
00 103300
00 103400
00 103500
00 103600
00 103700
00 103800
00 103900
00 104000
00 104100
00 104200
00 104300
00 104400
00 104500
00 104600
00 104700
00 104800
00 104900
00 105000
00 105100
00 105200
00 105300
00 105400
00 105500
00 105600
00 105700
00 105800
00 105900
00 106000
00 106100
00 106200
00 106300
00 106400
00 106500
00 106600
00 106700
00 106800
00 106900
00 107000
00 107100
00 107200
00 107300
00 107400
00 107500
00 107600
00 107700
00 107800
00 107900
00 108000

```

```

00 10A100
00 10A200
00 10A300
00 10A400
00 10A500
00 10A600
00 10A700
00 10A800
00 10A900
00 109100
00 109200
00 109300
00 109400
00 109500
00 109600
00 109700
00 109800
00 109900
00 110000
00 110100
00 110200
00 110300
00 110400
00 110500
00 110600
00 110700
00 110800
00 110900
00 111000
00 111100
00 111200
00 111300
00 111400
00 111500
00 111600
00 111700
00 111800
00 111900
00 112000
00 112100
00 112200
00 112300
00 112400
00 112500
00 112600
00 112700
00 112800
00 112900
00 113000
00 113100
00 113200
00 113300
00 113400
00 113500
00 113600
00 113700
00 113800
00 113900
00 114000

```

```

144 CJNT INUE
C: DIMENSION EC ITEM
IF (COMMON .EQ. 0) GO TO 135
N5=12
GO TO 46
CONT INUE
IF (I .LE. 5) GO TO 46
IF (I .NE. 4) GO TO 152
C: CLOSE PAREN OF DIMENSIONED ITEM.
JCNT=N1
IF (INTYPE .LT. 2) GO TO 45
C: OUTPUT THE DCL STATEMENT.
IF (I=1) GO TO 147
ICMPRE(I)=KARSET(5C)
CONT INUE
N3=59
IF (N2 .EQ. 0) N3=6C
C: IF N2 IS ZERO, THE ITEM IS FLOATING. OTHERWISE, IT IS INTEGER.
IBYT1=MSGOR1(N2)
IBYT2=MSGOR2(N2)
INDEX=INCOMP+7
CALL LDBYTE
N3=12
CONT INUE
ICMPRE(N3)=LBYTE(JMRK)
N3=N3+1
JMRK=JMRK+1
IF (JMRK .LE. JCNT) GO TO 148
N5=0
GO TO 140
CONT INUE
IF (N1 .NE. NREAD) GO TO 152
IF (IFRTRN .EQ. 0) GO TO 164
C: END STATEMENT. CHANGE TO QREND. END.
N2=N1
N1=LJOP
N4=61
GO TO 73
CONT INUE
C: RETRY STATEMENT. CHANGE TO GO TO QREND.
IFRTRN=1
JCNT=NREAD-N1
N2=N1
N1=LJOP
N3=2
N4=62
CONT INUE
N5=14
CONT INUE
C: REPLACE CHARACTERS IN LBYTE ARRAY.
C: N1---FIRST REPLACEMENT POSITION.
C: N2---SECOND REPLACEMENT POSITION.
C: N3---DELETE I NUMBER REPLACEMENT.
C: N4---NUMBER FOR MESSAGE NUMBER, Q LABEL IF MINUS.
C: N5---RETURN CODE.
IF (N2 .LT. N1) N2=N1-1
IF (NREAD .LT. N2) NREAD=N2
IF (N3 .NE. 2) GO TO 153

```

```

00114100
00114200
00114300
00114400
00114500
00114600
00114700
00114800
00114900
00115000
00115100
00115200
00115300
00115400
00115500
00115600
00115700
00115800
00115900
00116000
00116100
00116200
00116300
00116400
00116500
00116600
00116700
00116800
00116900
00117000
00117100
00117200
00117300
00117400
00117500
00117600
00117700
00117800
00117900
00118000
00118100
00118200
00118300
00118400
00118500
00118600
00118700
00118800
00118900
00119000
00119100
00119200
00119300
00119400
00119500
00119600
00119700
00119800
00119900
00120000

```

```

C:REPLACEMENT STRING.
IBYT1=MSGDR1(N4)
IBYT2=MSGDR2(N4)
INDEX=ICMP
CALL LDBYTE
N3=1
N4=IBYT2-IBYT1+1
GO TO 157
CNT INUE
IF(N3.EQ.0) GO TO 155
C:REPLACEMENT NUMBER.
N3=MAX IN
J=V4
N4=ABS(N4)
CNT INUE
IBYT1=N4/10
IBYT2=N4-10*IBYT1
N4=IBYT1
ICMPRE(N3)=KARSET(IBYT2)
IF(N4.EQ.0) GO TO 156
N3=N3-1
GO TO 154
CNT INUE
N4=-1
GO TO 157
CNT INUE
N4=MAX IN
IF(J.GE.0) GO TO 157
N3=N3-1
ICMPRE(N3)=KARSET(26)
C:PREFIX THE 0 BEFORE THE LABEL.
157 CNT INUE
I=N1-N2-N3*N4
IBYT1=NREAD
IF(I) 158, 161, 159
C:REDUCE SPACE.
N2=N2+1
IF(N2.GT.NREAD) GO TO 160
J=N2+1
LBYTE(J)=LBYTE(N2)
GO TO 158
CNT INUE
C:INCREASE SPACE.
J=IBYT1+1
LBYTE(J)=LBYTE(IBYT1)
IF(1) 158, 161, 159
IF(1) 158, 161, 159
CNT INUE
160 SPACE NOW CORRECT.
NREAD=NREAD+1
161 CNT INUE
IF(N3.EQ.0) GO TO 163
162 CNT INUE
LBYTE(N1)=ICMPRE(N3)
N1=N1+1
N3=N3+1
IF(N3.LE.N4) GO TO 162
CNT INUE
IF(1) .GE.0) GO TO (45,43,81,87

```

00 120100
 00 120200
 00 120300
 00 120400
 00 120500
 00 120600
 00 120700
 00 120800
 00 120900
 00 121000
 00 121100
 00 121200
 00 121300
 00 121400
 00 121500
 00 121600
 00 121700
 00 121800
 00 121900
 00 122000
 00 122100
 00 122200
 00 122300
 00 122400
 00 122500
 00 122600
 00 122700
 00 122800
 00 122900
 00 123000
 00 123100
 00 123200
 00 123300
 00 123400
 00 123500
 00 123600
 00 123700
 00 123800
 00 123900
 00 124000
 00 124100
 00 124200
 00 124300
 00 124400
 00 124500
 00 124600
 00 124700
 00 124800
 00 124900
 00 125000
 00 125100
 00 125200
 00 125300
 00 125400
 00 125500
 00 125600
 00 125700
 00 125800
 00 125900
 00 126000

```

1 69, 94, 164, 106, 108, 109, 47
2 J=VR EAD-1
  LBYTE(J)=KARSET(50)
  I=I+1
  GO TO 162
CONTINUE
READ NR EAD+1
LBYTE(NR EAD)=KARSET(47)
CONTINUE
IF(NTYPE .GE. 3) GO TO 185
CONTINUE
I=LBYTE(I)
IF(MAP(I) .EQ. 50) GO TO 168
NREAD=NR EAD+1
J=VR EAD
CONTINUE
J=J-1
LBYTE(J+1)=LBYTE(J)
IF(J .GT. 1) GO TO 167
LBYTE(1)=KARSET(50)
IF(MAP(1) .GT. 5) GO TO 168
C: ADD THE PREFIX
LBYTE(1)=KARSET(26)
GO TO 166
CONTINUE
IF(MAP(1) .EQ. 35 .OR. NREAD .LE. 6) GO TO 193
I=3
J=3
CONTINUE
I=I+1
CONTINUE
J=J+1
IF(J-NR EAD) 171, 172, 173
CONTINUE
IF(LBYTE(J) .EQ. KARSET(50) .AND. LBYTE(J+1) .EQ. KARSET(50)) GO TO 170
CONTINUE
IF(I .EQ. J) GO TO 165
LBYTE(I)=LBYTE(J)
LBYTE(J)=KARSET(50)
GO TO 169
CONTINUE
NREAD=I-1
CONTINUE
IF(NREAD .LE. 70) GO TO 193
K=70
CONTINUE
J=LBYTE(K-1)
J=MAP(J)
IF(J .GE. 36 .AND. J .LE. 41 .OR. J .GE. 45 .OR. J .EQ. 50) GO TO 176
K=K-1
IF(K .EQ. 0) GO TO 192
GO TO 175
CONTINUE
DJ=177
IF(DJ=1, 72)
ICMPRE(I)=KARSET(50)
CONTINUE

```

```

00 126100
00 126200
00 126300
00 126400
00 126500
00 126600
00 126700
00 126800
00 126900
00 127000
00 127100
00 127200
00 127300
00 127400
00 127500
00 127600
00 127700
00 127800
00 127900
00 128000
00 128100
00 128200
00 128300
00 128400
00 128500
00 128600
00 128700
00 128800
00 128900
00 129000
00 129100
00 129200
00 129300
00 129400
00 129500
00 129600
00 129700
00 129800
00 129900
00 130000
00 130100
00 130200
00 130300
00 130400
00 130500
00 130600
00 130700
00 130800
00 130900
00 131000
00 131100
00 131200
00 131300
00 131400
00 131500
00 131600
00 131700
00 131800
00 131900
00 132000

```

```

178 J=7 I=K,NREAD
    DJ 178 I=K,NREAD
    ICMPRE(J)=LBYTE(I)
    LBYTE(I)=KARSET(50)
    J=J+1
    CONTINUE
    NREAC=NREAD-K+7
    CONTINUE
C:CHECK ON AND DON'T OUTPUT A BLANK LINE.
179 DJ 180 I=1,72
    IF(LBYTE(I).NE.KARSET(50).AND.
    LBYTE(I).NE.LSTCHK) GO TO 181
    CONTINUE
    GO TO 183
181 CONTINUE
C:ATTACH LINE NUMBER.
    LINE=LINE+1
    LBYTE(79)=KARDUM
    LBYTE(80)=KARDUM
    I=78
    N=LINE
    CONTINUE
    J=N/10
    K=N-10*J
    N=J
    LBYTE(1)=KARSET(K)
    IF(I-1-GE.72) GO TO 182
    I=I-1
    Ibuff=IBUFF+75
    INDEX=IMLBYT
    CALL SBYTE
    GO TO 3
183 CONTINUE
    IF(IERR.NE.0) GO TO 187
    IF(NREAD.EQ.0) GO TO 185
    J=NREAD
    IF(J.LT.72) J=72
    DJ 184 I=1,J
    LBYTE(I)=ICMPRE(I)
    CONTINUE
    LSTCHK=KARSET(50)
    GO TO 174
185 CONTINUE
    IERR=0
    IF(JSW.EQ.2) GO TO 7
    NREAD=IPEAD MAXLIN
    DJ 186 I=1,ICMPRE(I)
    LBYTE(I)=ICMPRE(I)
    CONTINUE
    GO TO(192,192,50,141,100,1),JSW
187 CONTINUE
C:ERROR LINE. PUT OUT LINE OF //
    NJMERR=NUMERR+1
    DJ 188 I=1,72
    LBYTE(I)=KARSET(5)
    CONTINUE
    IERR=0
    GO TO 193
189 CONTINUE

```



```

C: END.          IF(NUMERR .EQ. C) GO TO 154
                PUT OUT SUMMARY.
190             DO 190 I=1,8C
                LBYTE(I)=KARSET(5C)
                CJNT INUE
                IBYT1=MSGDR1(65)
                IBYT2=MSGDR2(65)
                INDEX=IWLBYT
                CALL LDBYTE
                K=50
                CONT INUE
191             I=NUMERR/10
                J=NUMERR-10*I
                NJMERR=I
                LBYTE(K)=KARSET(J)
                K=K-1
                IF(I .NE. 0) GO TO 151
                GO TO 193
192             CJNT INUE
                C: CANNOT CRACK. PUT OUT ORIGINAL LINE AND //
                JSW=2
                IF(NTYPE .GE. 3) GO TO 7
                IBYT1=IBUFF
                IBYT2=IBUFF+71
                INDEX=IWLBYT
                CALL LDBYTE
                IERR=1
                CJNT INUE
193             NREAD=0
                GO TO 175
194             CJNT INUE
                IF(NTYPE .GE. 2) GO TO 1
                JSW=6
                N5=80
195             CJNT INUE
                IF(NTYPE .GE. 2) GO TO 7
                C: OUTPUT LINE IN ICMPRE IF N5 EQ ZERO OR IN MSGDR(N5)
                C: IF N5 NOT ZERO. DO NOT CHANGE LBYTE.
196             CJNT INUE
                DO 197 I=1,MAXL IN
                J=LBYTE(I)
                ICMPRE(I)=J
                IF(N5 .GT. 0) LBYTE(I)=KARSET(50)
197             CJNT INUE
                IREAL=NREAD
                IF(N5 .EQ. 0) GO TO 153
                IBYT1=MSGDR1(N5)
                IBYT2=MSGDR2(N5)
                INDEX=IWLBYT+2
                IF(N5 .EQ. 80) INDEX=IWLBYT
                CALL LDBYTE
                GO TO 193
198             CJNT INUE
                STOP
                END

```

```

00132100
00132200
00132300
00132400
00132500
00132600
00132700
00132800
00132900
00133000
00133100
00133200
00133300
00133400
00133500
00133600
00133700
00133800
00133900
00134000
00134100
00134200
00134300
00134400
00134500
00134600
00134700
00134800
00134900
00135000
00135100
00135200
00135300
00135400
00135500
00135600
00135700
00135800
00135900
00136000
00136100
00136200
00136300
00136400
00136500
00136600
00136700
00136800
00136900
00137000
00137100
00137200
00137300
00137400
00137500
00137600

```