MACRO CONTROL STRUCTURES FOR STRUCTURED PROGRAMMING IN ALC

THESIS

Presented to the Graduate Council of the
North Texas State University in Partial
Fulfillment of the Requirements

For the Degree of

MASTER OF SCIENCE

By

Kim G. Connally, B.A.
Denton, Texas

December, 1975

Connally, Kim G., *Macro Control Structures for Structured Programming in ALC*. Master of Science (Computer Sciences), December 1975, 187 pp., 4 tables, 27 illustrations, bibliography, 38 titles.

This thesis describes a set of computer program control structures which permits the application of certain structured programming techniques to the IBM/360 assembly language (ALC). The control structures are implemented by programmer-defined instructions known as macros.

A history of computer software is presented, providing a basis for the emergence of structured programming. A survey of the major concepts of structured programming with special attention to control structures and their significance to structured programming follows.

The macros developed in this study include DO, ENDDO, LEAVE, CASE, and ENDCASE. They provide a looping control structure, a loop-escape construct, and a selective control structure. Examples of usage are given.

TCI

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF TABLES

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Continued)

CHAPTER I

INTRODUCTION

The Evolution of Software

During the first generation of the computer industry, the late 1940's through the middle 1950's, the emphasis on the development of computer software was negligible (2, p. 470). At the time technology was focused on the development of functional hardware. The vacuum tube proved so unreliable that extensive maintenance was required; consequently, only a minimum amount of programming could be done. The few computers available were typically one-of-a-kind and short-lived, so the need for standard software packages was not felt. Since the early machines were slow in execution and contained small memories, the programmer was severely restricted by the capabilities of his computer (3, p. 860).

The lack of software development then was a direct result of the primitive and unreliable hardware in existence; moreover, many programmers believed that with the improvement of computer machinery the burden of the programmer would be lifted (3, p. 860). The programmer would no longer have to contend with the limitations of the hardware but would enjoy a new freedom, and programming would no longer be a problem.

As a result of the early experiences with computers, the important role that software and programming would assume in future machines was grossly underestimated (2, p. 469). The second generation of computers, extending from the late-1950's to the mid-1960's, gave the programmer his larger and faster machine, and in so doing, completely altered the role of software in the computer industry.

The programmers of the second generation faced the task of developing software to match the advances in electronic technology. The transistorized machines of the second generation introduced an unforeseen complexity to programming. Programmers had to deal with such problems as I/O interrupts, multilevel stores and multiprogramming. The limitations of computing were shifting from the hardware to the software.

Another problem confronted the programmers of the second generation. With these new and more powerful machines the widespread use of computers in industry became feasible (2, p. 472). There was a tremendous demand for programmers and software by business and industry. Not only was the computer industry lacking in software development, but there was also an inadequate number of experienced programmers available to produce the needed software.

The problems of the software industry continued to grow with time. During the mid-1960's, the beginning of the third generation computers, integrated circuit technology came into use. Again computer hardware increased in speed and complexity. There now existed what has been termed the "software crisis" (2, p. 474). In short this crisis represented the disparity between the sophistication and capabilities of computer machinery and the inadequate and functionally underdeveloped software used by these machines.

In summary, as the technology of computer hardware improved, providing smaller, faster, and more complex machines, programmers were faced with the increasingly difficult task of designing software for these machines and programming the many problems of business and industry which these machines were capable of solving. To quote Dijkstra,

> . . . as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now that we have gigantic computers, programming has an equally gigantic problem (3, p. 861).

A crisis has developed as a result of the inabilities of the programmers and their software to meet this problem.

Today this problem is best reflected in the software produced in industry. Industrial programs are usually very expensive since they typically require many man-hours to code, and quite often they are error-prone because their complexity prevents adequate debugging. Industrial software is usually difficult to understand, hard to modify, requires constant maintenance and cannot be adequately tested for correctness. This is not surprising since many industrial programmers have had no formal training in organized program production.

The typical industrial programmer produces on the order of five to ten lines of code per day over the average life of a project because most of his time is spent debugging (1, p. 58). Programming techniques in present use are not producing either the quantity or quality of software that is in demand.

The computer industry has realized the inadequacies of software and programming techniques for quite some time. Accordingly, within the last seven years a growing number of programmers have expounded various concepts and methodologies which they believe will improve software design and will increase the productivity, reliability, maintainability, and extendability of programs. These various concepts and techniques have been referred to as "structured programming." At this time there is disagreement as to what does or does not constitute structured programming. Some programmers feel that it encompasses a wide variety of techniques while others view it as a single or perhaps limited aspect of programming. It remains to be seen exactly what will eventually be included in a definition of structured programming, but its development has been a direct result of the software crisis.

Chapter II presents a survey of the major concepts associated with structured programming. The work of E. W. Dijkstra is first presented. It includes the concepts of "GOTO-less" programming, hierarchy of software modules, abstract resources and the principle of non-interference. Dijkstra's work is followed by a discussion of the contributions of H. D. Mills, which includes such concepts as the "top-down" approach to programming, the chief programmer team method, the one-entry and one-exit rule for program module design, and the development support library. A number of other important structured programming concepts which are not associated with a particular contributor are also discussed. These concepts include modular programming, program clarity, and open programming.

Of the various concepts involved in the structured programming controversy one of the most commonly implicated is the issue of control structures. A control structure is a program instruction or set of instructions which determines the order in which other program instructions will be executed or the number of times a particular set of instructions are to be executed. Simple control structures include the unconditional branch statement, the branch-on-high and the branch-on-low instructions. More sophisticated control structures include the FORTRAN DO statement, PL/I IF-THEN-ELSE statement, and program interrupts caused by external events.

The controversy centers around the correct use of control structures. What set of control structures should the programmer be allowed to use? Should he have a large set of structures at his disposal or should he be limited to a small but sufficient set? Should the GOTO statement be allowed or should it be removed completely? What discipline should be imposed on interrupt programming methods? These are some of the questions to which programmers are addressing themselves in an effort to determine the proper set and use of control structures in programming.

Purpose of the Study

The purpose of this thesis is to describe the development of a small but sufficient set of control structures to facilitate structured programming in ALC, the macro assembly language of the IBM 360 computer.

An ALC macro call consists of a single programmer-defined macro instruction which may be inserted into an ALC program. At assembly time the macro instruction is replaced by a predetermined set of ALC instructions associated with it. With a set of ALC macro definitions for various higher-level language control structures an assembly language programmer is able to code programs using the concepts of structured programming which require these more sophisticated structures. The programmer uses the macro instructions to implement the logic defined by higher-level language control structures.

Before describing the macros developed in this thesis, a survey of the major concepts of structured programming is presented both to familiarize the reader with structured programming and to set in proper perspective the important role that control structures play in structured programming. The concepts of control structures are more fully expanded in a separate chapter which precedes the discussion of the macros.

Justification

The numerous articles which discuss structured programming and the control structures of structured programming limit themselves almost without exception to a discussion of concepts which apply to higher-level languages. Kessler (4) has published a noteworthy report which did attempt to apply the concepts of structured programming directly to assembly language. Given the present state of software and the prevalent use of assembly languages in industrial programming, it should be evident that there exists a need for a thorough investigation into the application of structured programming techniques to the control structures of assembly language programming.

Assembly language programs by the nature of their instruction sets are difficult to code and debug; thus the need for improving assembly language programming techniques is obviously present. Since assembly language programs usually require many more lines of code than equivalent higher-level language programs, the necessity of improving assembly language programs would seem to be greater than that of higher-level languages. One of the major intentions of structured programming is to make programs more readable; the assembly language program is in most instances more difficult to read than a comparable higher-level language program.

The use of structured programming control structures in higher-level languages facilitates a block-structured or modular program design. If these control structures are applied to assembly languages, then block-structured programming could be more easily implemented in assembly language programs. The control structures or branching mechanisms which are presently available in most assembly languages do not readily lend themselves to modular or block-structured programming.

Higher-level languages usually contain a variety of control structures, some of which may be very powerful.

In contrast, the typical assembly language contains an unconditional branch, unconditional subroutine linkage, and a small set of very simple conditional branches such as a branch-on-equal, branch-on-high, branch-on-negative, etc. With the implementation of a more powerful and problem-oriented set of control structures in assembly languages the methods of structured programming could be applied more directly to assembly language programming.

This research was undertaken in light of the fact that there exists a need to improve the techniques of assembly language programming. If the concepts of control structures of higher-level languages within the scope of structured programming are applied at the assembly language level, perhaps significant improvements in assembly language programming will be achieved.

Before entering a discussion of control structures and their proper use, an overview of the development of structured programming, its major concepts, and its major contributors will be given in Chapter II in order to establish the significance and relationship of control structures to structured programming.

CHAPTER BIBLIOGRAPHY

1. Baker, F. T. and H. D. Mills, "Chief Programmer Teams," Datamation, XIX (December, 1973), 58-61.

2. Bauer, F. L., "Software and Software Engineering," SIAM Review, XV (April, 1973), 469-480.

3. Dijkstra, E. W., "The Humble Programmer," Communications of the ACM (October, 1972), 859-866.

4. Kessler, M. M., *Concepts* Report 4, OS/360 Assembly Language Block Structured Programming Macros, IBM.

5. Rosen, Saul, "Electronic Computers: A Historical Survey," Computing Surveys (March, 1969), 7-36.

CHAPTER II

STRUCTURED PROGRAMMING

The origin of structured programming is usually traced to a letter by E. W. Dijkstra (7), which appeared in the March, 1968, issue of the Communications of the ACM. The letter entitled "GOTO Statement Considered Harmful" warned programmers of the potential problems which the GOTO statement may introduce into programs.

Dijkstra's Work

In his letter Dijkstra pointed out that the unrestricted use of the GOTO statement may unnecessarily complicate the flow of control within a program. Because of the complexity of such a program difficulties may be encountered if debugging or modification is required.

If a program contains many GOTO statements, it is likely that the program will have a nonlinear flow of control. This means that its instructions will not be executed in the same order as they are written. In contrast, when a program contains no GOTO's or other branching constructs, its instructions are executed in a purely sequential manner. The first instruction is executed first, the second next, and so forth until termination occurs. Such a program is said to have a linear flow of control. While few programs can be written in a purely linear fashion, the free use of GOTO's may completely destroy any semblance of linearity and produce unnecessarily complicated program logic.

Figure 1 illustrates the type of program which may result from the blatant use of the GOTO statement. Notice that the program has a very complicated execution sequence. There is no correspondence between the order in which the statements appear and their order of execution. Although the program of Figure 1 is rather small, it would be difficult to debug or prove correct because of its nonlinear flow of control.

Since the time of Dijkstra's letter so much literature has been published on the use of GOTO statements that structured programming and "GOTO-less" programming are sometimes used synonymously. This, however, is an over-simplification. A program which contains GOTO's may be very well structured in the sense that it maintains an almost linear or sequential flow of control. On the other hand the flow of control of a program coded without GOTO's is not inherently linear or necessarily well structured. Dijkstra cautioned the programmer that the GOTO statement lends itself to misuse, i.e., it may disrupt the sequential execution of the program. He was not suggesting that programming without

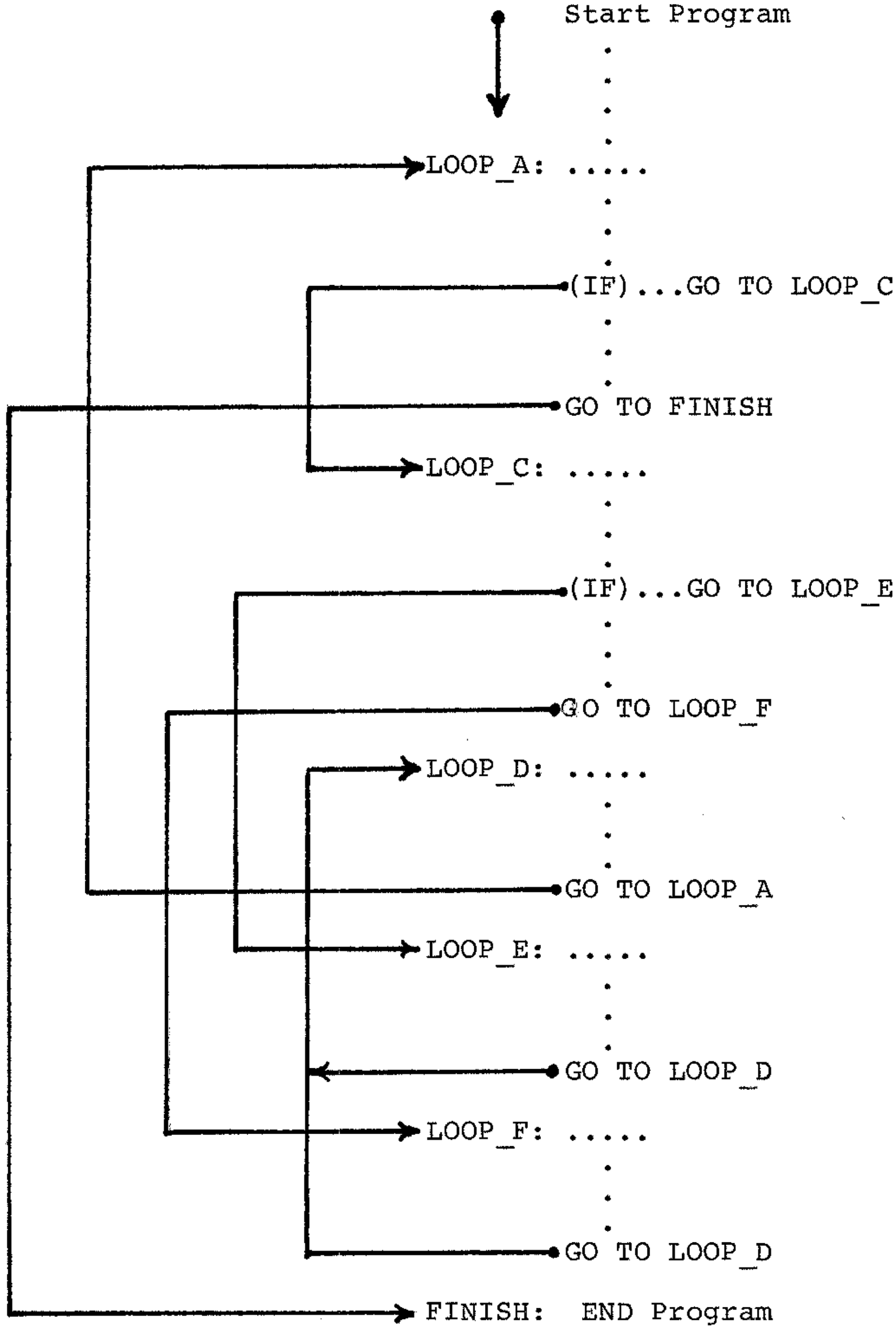

Fig. 1--The excessive use of GOTO statements in a program.

the GOTO statement would in itself produce well-structured programs.

It has been shown in a paper by Bohm and Jacopini (4) that the following three control structures are a sufficient set to define any program logic: a sequence control structure, a repetitive control structure, and a selective control structure. Using these three structures only, the GOTO statement is not needed nor for that matter are any other control structures required.

At this time there is a controversy among programmers as to whether or not the GOTO statement should be completely eliminated since it is not strictly required. Those in favor of its removal argue that it is merely a temptation toward poor programming practices. Those not in favor of its removal argue that circumstances occur when the judicious use of an occasional GOTO statement will not interfere with the program's structure and will, in fact, provide a solution to a logic problem which may otherwise require excessive code.

The concept of programming without the GOTO statement is by no means the only contribution of Dijkstra to structured programming. Concerned with the development of software systems, he published his experience with the "THE" Multiprogramming System which made use of what he calls abstract resources in the design of software systems (6).

Dijkstra views a software system as a hierarchy of software modules or machines. Each level of the hierarchy produces an abstract resource which is supported by the level below it. Each of the resources is available to the level directly above it. Any module then consists of a set of programs which manipulate the abstract resource of the next lower level and produce an abstract resource which can be manipulated by the next higher level.

An example will illustrate Dijkstra's concepts of abstract resources, hierarchy layers, and levels of abstraction.

A software package which will read a file is to be written. The file is considered the highest level of abstraction and will consequently, be the highest layer in the hierarchy. The next level is a record since a file can be considered a collection of records. A record is a collection of words, a word is a collection of bytes and a byte is a collection of bits so that the hierarchy of resources needed to read a single bit is as follows:

Highest Level: FILE
RECORD
WORD
BYTE
Lowest Level: BIT

There are five abstract resources and hence, five levels of abstraction. Each level is a layer in the hierarchy and represents a functional part or machine in the system. A set of programs would be written to implement each level. For instance, at the lowest level a set of programs would be written which operate on bits and thus alter bytes. At the highest level a set of programs would operate on records to manipulate files.

Dijkstra emphasizes that each level has a specific relation to the level immediately above and below it, and that special care should be taken to insure that each level is consistent in that the operations executed at one level will be supported by those below and will, in turn, support those in the level immediately above.

In developing his software system Dijkstra attempted to use a design which would lend itself to thorough testing and which could be proven logically correct. Using his method of hierarchy of levels, Dijkstra was applying what he refers to as the "principle of non-interference" (3, p. 143). This principle suggests that the correctness of the software system can be more readily determined if the system is divided into a set of smaller problems which are logically independent and which can be united into the software package at the functional level. These small problems must be proven correct at every level of integration.

To test the example above using this principle it would be necessary to test each layer. For example, the middle layer would be independently tested to insure that it correctly manipulates bytes to produce words. Likewise, when each layer had been thoroughly and independently tested, the system would be considered correct.

It should be noted at this time that Dijkstra's hierarchy of levels or layered approach to software design appears to be equivalent to a modular approach to programming. In fact, Dijkstra's approach is modular, but he has imposed other restrictions such as the hierarchy concept on the modules formed by the design. The sample given above is modular in that the byte programs are independent of the bit and word programs, however, there is the added restriction that the word programs may manipulate only bytes and have no direct effect or connection to record, file or bit programs. Modular programming,which will be discussed later, does not necessarily place such restrictions on the relationships among various modules. There is no hierarchy or layering of modules in the conventional approach to modular programming.

Dijkstra seems to be most concerned with developing techniques for designing and encoding programs which will lend themselves to testing and proof of correctness. He advocates the avoidance of GOTO statements because they tend to add complexity to programs and hence, increase the difficulty of proving their correctness (7). Dijkstra approaches software design with his hierarchy technique since to verify the software system, it is necessary to prove correct only the independent layers rather than the system as an entity. In designing his multiprogramming system Dijkstra was assisted by a group of programmers who were mostly mathematicians; thus, an indication of the emphasis he places on the necessity of proving program correctness (6).

Mills' Work

Harlen Mills, F. T. Baker, and others at the IBM Corporation have developed some operational procedures for the design and encoding of large reliable programs. The techniques they have developed represent a major contribution to structured programming.

Mills (14) describes what he calls a "top-down" approach to program design and coding. He also elaborates on a number of programming techniques, which he collectively refers to as structured programming. These techniques include the use of a standard set of control structures or branching conventions, a modular or segmenting method of programming, and a restriction on any module that it have only one entry and one exit point.

The top-down approach to programming is a technique whereby the initial problem is repeatedly broken down into a hierarchy of program modules or segments. The highest module might represent a control or supervisory function.

The various routines called by the control module would represent the next lower-level of the hierarchy. Likewise, the second-level routines reference various lower-level routines. In effect, a tree structure containing program modules is formed. Coding begins at the highest level and proceeds downward with "program stubs," dummy names to represent uncoded segments being inserted into the code where references to lower-level segments are made.

The program modules or segments are carefully limited in size so that they may be coded on a single page. A segment defines a function having one entry and one exit point. The function merely transforms data which may or may not represent another segment.

The branching conventions or control structures which may be used in coding a segment include a simple sequencing of code, a selective branching structure such as an IF THEN ELSE statement and a repetitive control structure such as a DO WHILE statement. The GOTO statement is not permitted.

By requiring that each program segment be designed and coded according to these particular structured programming techniques, a certain amount of program uniformity is insured (12, pp. 22-23). Each segment will be reasonably small and will not possess any complicated control structures to interfere with module testing or readability (13, p. 156). Module interfacing is simplified when the one entry and one exit rule is used since there can be only one path of connection between program segments.

A top-down design method permits program testing concurrently with program coding (13, pp. 9-10). Once a particular segment on the hierarchy structure has been coded, it may be tested by having the programmer provide the necessary input to evaluate the correctness of the segment. This input, of course, will eventually be supplied by the lower-level segments which are called by the segment being tested. In this manner the program can be verified as each module is coded; thus the reliability of the entire system is proven without having to rely on exhaustive testing once the system is complete.

Mills' top-down programming technique is similar to Dijkstra's layered approach to program design; however, the two methods seem to depend on slightly different concepts and each emphasizes somewhat different aspects of design. While Dijkstra is chiefly concerned with the separation of abstract resources during his hierarchy development (13, p. 57), Mills is mostly interested in achieving a hierarchy tree structure of one entry and one exit modules. Dijkstra emphasizes a design approach which is intended to be highly testable and provable while Mills, though he is concerned with the ability to test programs seems to emphasize simplicity and clarity in program design (12, p. 57).

Another concept of structured programming developed at IBM and described by Mills (13) is the "chief programmer team," which is an organizational and managerial technique for large program production.

The chief programmer team consists of a chief programmer, a backup programmer, a programming librarian, and other junior programmers required by the particular production. These team members utilize a development support library and apply the principles of top-down design and structured programming described by Mills in the development of large programs.

The chief programmer is a technical manager responsible for designing and coding the most important segments of the program. All other team members receive their responsibilities and coding assignments from the chief programmer. He coordinates all program interfaces and supervises all coding to insure the proper use of top-down structured programming techniques. Quite naturally the chief programmer must be a highly experienced and competent programmer since the success of the team and the project are largely dependent upon his decisions.

The backup programmer is a research assistant to the chief programmer and helps significantly in the design and coding of the major portion of the project. He must be completely familiar with the entire project and be ready to assume the position of chief programmer should the need arise. The backup programmer shares the burden of responsibility, allowing the chief programmer to concentrate on the major problems encountered in the project. The backup programmer may, for example, develop all testing procedures without the assistance of the chief programmer.

The programming librarian is responsible for maintaining all listings and records of the project. This information is kept in both an internal, machine-readable, and an external, human-readable, form. The librarian maintains this information in a development support library.

The development support library contains a set of organized listings which detail the current status and previous development of the project. These listings represent the external project records. Among these records are notebooks which are headed by a directory and contain an alphabetized list of the program modules. A journal is kept to record all changes in updating the directory. All results of testing procedures are also recorded in a journal.

By maintaining a detailed record of the project's development the programmers have an accurate account of all program bugs encountered and tests made at any given time. Since these records are maintained by a programming librarian, the programmers are not burdened with time-consuming clerical work.

Besides maintaining and updating the various records which are kept in the library, the librarian has a significant amount of paperwork to do concerning the documentation of the design and coding phases of the program. Mills (2) emphasizes that the librarian is a key member of the team rather than a part-time assistant to programmers.

The development support library consists of a number of office and machine procedures for maintaining programmer-generated material such as coded program segments, for maintaining files and records of the project in the external records, for processing data in the internal library which is on disk, and for performing all runs during each stage of program development including testing procedures.

During the course of the project programmers make corrections in status notebooks, introduce new or altered coding sheets, and request various runs. It is the responsibility of the programming librarian to invoke the necessary office or machine procedures to accomplish these tasks. He is responsible for preparing and executing all program runs and posting the results in the external and internal files.

Figure 2 shows the relationship of the librarian and programmers within the development support library (2). It is evident in the diagram that the librarian plays a key role in the success of the library.

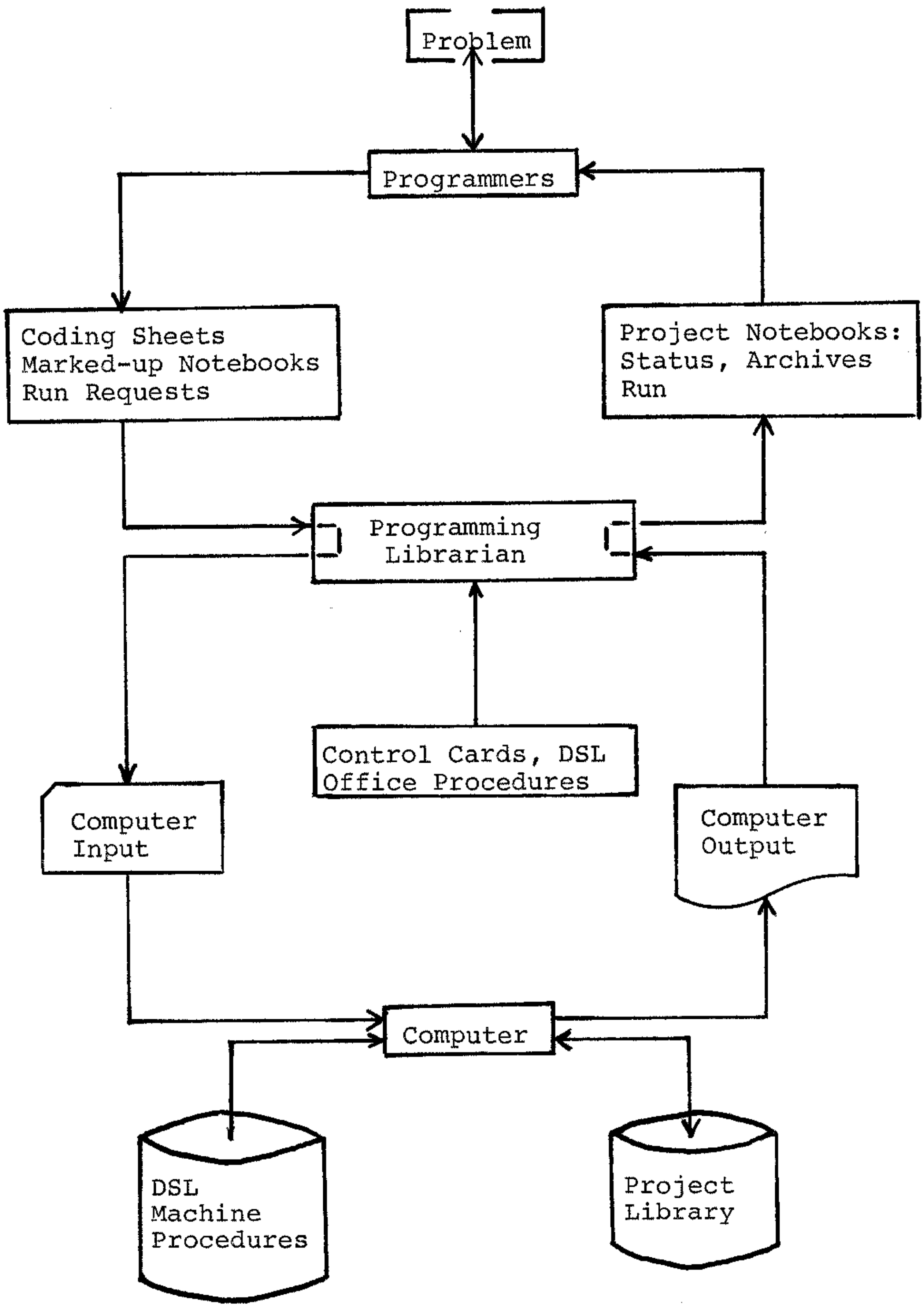

Fig. 2--Development support library

By using the development support library within the chief programmer team method, the team members are working on a common product rather than merely coding independently, separate segments of a large program. Since the members are working together, there is less chance of duplicating code or coding errors. The library provides an in-depth documentation of the development of the project. This documentation reflects the progress of the team.

The chief programmer team concept was developed in order to improve the organization, communication, and productivity of programmers involved in large-scale programming. The top-down method is applied to program design under the chief programmer team method, and the structured programming techniques are applied in code segments within the top-down method.

F. T. Baker (1) described an industrial project which was developed using a chief programmer team. The project required twenty-two man-months to design and code, involving the production of over 83,000 lines of code. Only twenty-one errors were found during formal testing of the completed system.

Baker concluded that the use of the chief programmer team, top-down programming, and the application of structured programming techniques contributed to the success of the project. He suggested that the top-down method may not always be applicable to some types of projects, and that other methods may be more feasible in the design of some programs. For instance, when a program organization, viewed as a tree structure, is narrow and tall, then a strictly top-down approach may require too much time to be practical (1, p. 343). Baker also suggested that the chief programmer should do more code reviewing and allow the other members to do the major portion of coding. In the project described by Baker the chief and backup programmers did most of the functional coding; consequently, there was little time to review code, especially that written by junior programmers.

As a result of the development of top-down programming and the chief programmer team at IBM, the concept of structured walk-through evolved at that corporation (16, p. 31). A structured walk-through is a series of progress meetings which are held at various times in the design and development of a programming project.

A committee of approximately five members discuss the completeness, accuracy, and general quality of a project's development (10, p. 30). Each member of the committee, or reviewer as he is called, presents a brief introduction to his portion of the project and then "walks" the other committee members through the specific function of his area of responsibility. In this way each member becomes familiar with the purpose and progress of all aspects of the project.

One committee member, the recording secretary, records errors and inconsistencies which are discovered during each reviewer's walk-through. When the meeting is completed, each reviewer is given a copy of the secretary's notes. It is the responsibility of each member to insure that any problems within his area are resolved and that he notifies other committee members of any corrective action he takes.

The purpose of a structured walk-through is not to evaluate the ability or effectiveness of its committee members, but rather, through the exchange of information and ideas, determine the progress of the project, and detect the errors existing in the production at its current level of development (16, p. 30). Committee members are encouraged to exchange ideas, offer constructive criticism, and view the meetings as educational experiences.

The use of structured walk-through provides a method for not only measuring a project's development, but also for discovering production errors at the earliest possible time when they are easiest to correct and have the smallest impact on the production (10, p. 35).

Modular Programming

Modularizing a program refers to the technique of isolating sections or functions in the program so they may be designed and coded independently; hence the original problem is reduced to a set of smaller ones. Program modules might be coroutines, subroutines, programmer-defined functions, loops, or merely sequences of instructions which are logically grouped and coded together. The concept of modularity is not new to programming, but it has recently been given new emphasis as a structured programming technique.

The advantages of modular programming include: simplification of program design and coding by reducing a large problem into a set of smaller ones, extendability of coded modules since a functional program module can be inserted into other programs, ease of program modification where only effected modules need to be altered, and clarity of program design since the function of the entire program can be viewed as the interaction of the individual program modules.

Within the realm of structured programming programmers are well aware of the benefits of modular programming. They do not agree, however, on the principles which should govern program modularizing. Two methods which form a basis for program modularization have already been discussed, namely Dijkstra's levels of abstraction which modularizes according to what he calls abstract resources, i.e., the various elements or data types on which a program operates, and Mills' top-down design approach which modularizes according to a stepwise refinement of program segments. Besides these methods there exist two other important techniques which involve program modularity. They are compartmentalization and information hiding.

Compartmentalization is an approach to program coding whereby program modules are formed on the basis of their relationship to a particular design decision. Examples of design decisions include such things as I/O formats, arithmetic precision, and variable declarations. The advantage of this approach to program modularity is its ease of modification when new design specifications must be incorporated. For example, in a compartmentalized program if a new input format were required, only the module which generated formats would need to be changed. There would be no need to search through various program segments to alter each format statement.

To implement compartmentalized modules the most appropriate technique is by means of macros. One macro may expand to produce various format statements while another may, for instance, generate declaration statements. Such macros may provide the basis for program modules rather than other program features such as functions or subroutines.

Parnas (15) has suggested a method of modular programming which attempts to reduce the interface requirements and relationships between modules and thus reduce module connectivity. His method, information hiding, stresses the need to code some modules without utilizing characteristics or features of other interfacing modules. The programmer is supplied with the information needed to code his module, and information relating to the connecting modules is deliberately withheld or "hidden" from him. This technique may require additional coding to facilitate the proper interfacing of modules, but information hiding produces program modules which are inherently nonrestrictive and independent in nature.

It appears that no single modularizing technique is sufficient to always produce the most efficient and well-structured program; rather the technique to use seems to be dictated by the nature of the problem. The current interest in modular programming should yield valuable methods for the decomposition of programs and contribute significantly to structured programming.

Program Clarity

It was once considered sufficient to produce a program which would satisfy the problem at hand. It did not really matter if the program could be interpreted by anyone other than its author. As programs increased in size, however, and as the need for program modification became greater, program managers began to insist on code which could be understood and readily interpreted.

One of the basic goals of structured programming is to establish methods which will increase the readability of programs. This means writing a program in such a way that programmers unfamiliar with it will be able to understand what it is attempting to do.

The use of documentation is very important in providing program clarity. If there are explanatory comments at the beginning of each program module, the purpose of the code will be more apparent. Besides explaining code documentation should specify any restrictions or exceptions to code function such as input format requirements, arithmetic precision of output, or "special cases" not handled properly by the program.

The use of spacing and indentation will also improve the clarity of program documentation. Spacing to separate program modules and indentation to represent loop nesting and extent of control within a loop will help produce more legible code.

Figure 3 illustrates the use of spacing and indentation. Modules A and B in the figure are separated by spacing to show their logical independence. Module B has two levels of indentation to indicate both the range of the IF statement within the outer DO, and the range of the THEN DO and ELSE DO statements within the IF. Notice that spacing within the IF statement clarifies the effect of the THEN DO and the ELSE DO routines.

The use of meaningful labels and variable names will also improve program readability. Acronyms and word abbreviations may be necessary if the language being used

```
                    DO WHILE . . .
                     .
                     .
                     .
Module A  ---->      .
                     .
                     .
                     .
                    END
Spacing   ---->

                    DO I = 1 TO 10
                     .
                     .
                     .
                     .
                       IF . . .
                          THEN DO:
                            .
                            .
                            .
                            .
                            .
                          END
Module B  ---->

                          ELSE DO:
                            .
                            .
                            .
                            .
                          END
                        .
                        .
                        .
                        .
                      END
                     .
                     .
                     .
                     .
                    END
```

Fig. 3--Spacing and indentation

places restrictions on the lengths of labels and variable names.

Open Programming

Programming was once considered an art which only a few people knew how to do, but today it is more and more becoming a professional skill which can be taught and improved upon with proper guidance. Indeed, the basis of structured programming is a set of methods and techniques which assumes that programming can be taught and improved upon with the application of these techniques.

Open programming is a method for teaching and improving programming skills. It involves a group approach to learning. Programmers compare and discuss each other's work making evaluations and suggestions for improvement. An open panel discussion may be used to critique programs. There is frequent exchange of experiences and problems with fellow programmers.

The basic goal of open programming is self-improvement of programming techniques by studying the programming methods of others and having others evaluate one's own work. Weinberg (16) speaks of this practice as "egoless" programming. Each programmer must be willing to accept the constructive criticism of others while, at the same time, share his programming abilities with his critics.

Summary

This chapter has presented the major concepts associated with the term "structured programming." A variety of methods and techniques have been discussed, and it remains to be seen just which of them will be included in a formal definition of structured programming.

Intuitively it appears that any definition of structured programming will have to be of a general nature encompassing many concepts and techniques since the solution of the multitude of problems presented to programmers today require a variety of approaches and methods, especially with regard to program design.

Chapter III will discuss more fully the role of control structures in structured programming. So far it has been suggested that the proper use of control structures is to adapt a small but sufficient set of branching conventions and that the use of the unconditional GOTO branching statement should be avoided. Chapter III will present some of the theory and application of control structures with a more thorough analysis of the GOTO controversy.

CHAPTER BIBLIOGRAPHY

1. Baker, F. T., "System Quality Through Structured Programming," Proc. FJCC, (1972), 339-343.

2. ____________, and H. D. Mills, "Chief Programmer Teams," Datamation, XIX (December, 1973), 58-61.

3. Benson, Jeoffrey, "Structured Programming Techniques," Record of the 1973 IEEE Symposium on Computer Software Reliability, New York, (April, 1973), 143-147.

4. Bohm, C. and G. Jacopini, "Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules," Communications of the ACM, IX (May, 1966), 366-371.

5. Denning, P. J., "Is It Not Time to Define 'Structured Programming'?" SIGPLAN Notices, (February, 1974, 6-7.

6. Dijkstra, E. W., "The Structure of the 'THE' Multiprogramming System," Communications of the ACM, XI (May, 1968), 341-356.

7. ______________, "GOTO Statement Considered Harmful," Communications of the ACM, XI (March, 1968), 147-148.

8. ______________, "Complexity Controlled by Hierarchecal Ordering of Function and Variability," Software Engineering, Report on a conference sponsored by the NATO Science Committee, Garmish, Germany, October, 1968, 181-185.

9. Donaldson, James, "Structured Programming,"Datamation, XIX (December, 1973), 52-54.

10. Improved Programming Technologies Management Overview, IBM Corporation, Data Processing Division, Systems Marketing, Installation Productivity Programs Department (August, 1973).

11. McCracken, Daniel, "Revolution in Programming: An Overview," Datamation, XIX (December, 1973), 50-52.

12. Miller, E. and G. Lindamood, "Structured Programming: Top-down Approach," _Datamation_, XIX (December, 1973), 55-57.

13. Mills, H. D., _Chief Programmer Teams: Principles and Procedures_, Report No. FSC 71-5108, IBM Federal Systems Division, Gaithersburg, Maryland.

14. ___________, "On the Development of Large Reliable Programs," _Record of the 1973 IEEE Symposium on Computer Software Reliability_, New York (May, 1973), 155-159.

15. Parnas, D. L., "Information Distribution Aspects of Design Methology," _Information Processing_, LXXI, 339-344.

16. Weinberg, G., _The Psychology of Computer Programming_, New York, Von Nostrand Reinhold Co., 1971.

CHAPTER III

CONTROL STRUCTURES

Almost every paper on structured programming alludes to the proper use of control structures. This is not surprising since the structures used can affect many aspects of a program. The degree of linearity in program execution is a direct result of the structures used. Program modularity can be maximized with effective control structures. Testing a program and proving its correctness may be feasible only if the program's structures allow the isolation of program segments. The clarity and readability of a program may be significantly hampered if it contains complicated control mechanisms.

Bohn and Jacopini's Work

The importance of control structures has not been overlooked by those interested in improving programming techniques. A paper by Bohm and Jacopini (2) is frequently referenced in structured programming articles. The paper shows mathematically that the control of any flowchartable program can be logically defined using only three control structures. These structures include 1) a sequencing procedure, 2) a selective structure, and 3) a repetitive structure. Each structure is characterized by a single entry and single exit point.

The sequencing structure, shown in Figure 4, represents the normal execution of instructions, i.e., in the order the instructions were written. Figure 4 represents a linear flow of control. A program containing no branching structures would be completely sequential or linear in the strictest sense.

A selective structure, illustrated in Figure 5, causes the execution of a particular block of code or set of instructions depending on the truth of the selective condition. In the diagram the diamond represents the selective mechanism. The IF THEN ELSE structure of PL/I is a selective structure. When the IF statement is true, all instructions within the THEN block are executed. When the IF statement is false, all instructions within the ELSE block are executed.

The IF THEN ELSE structure may be considered linear since each instruction is executed or not executed in the same order as it is coded. The program flow of control does not change directions but continues forward either bypassing instructions or executing them as they are encountered.

The third structure, a repetitive block, is shown in Figures 6 and 7. This structure represents the repeated execution of a block of code. The diamonds in the diagrams

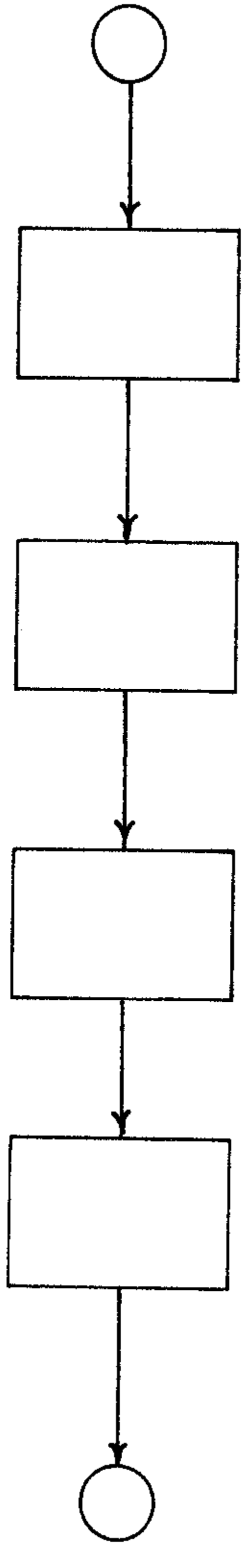

Fig. 4--A sequencing structure

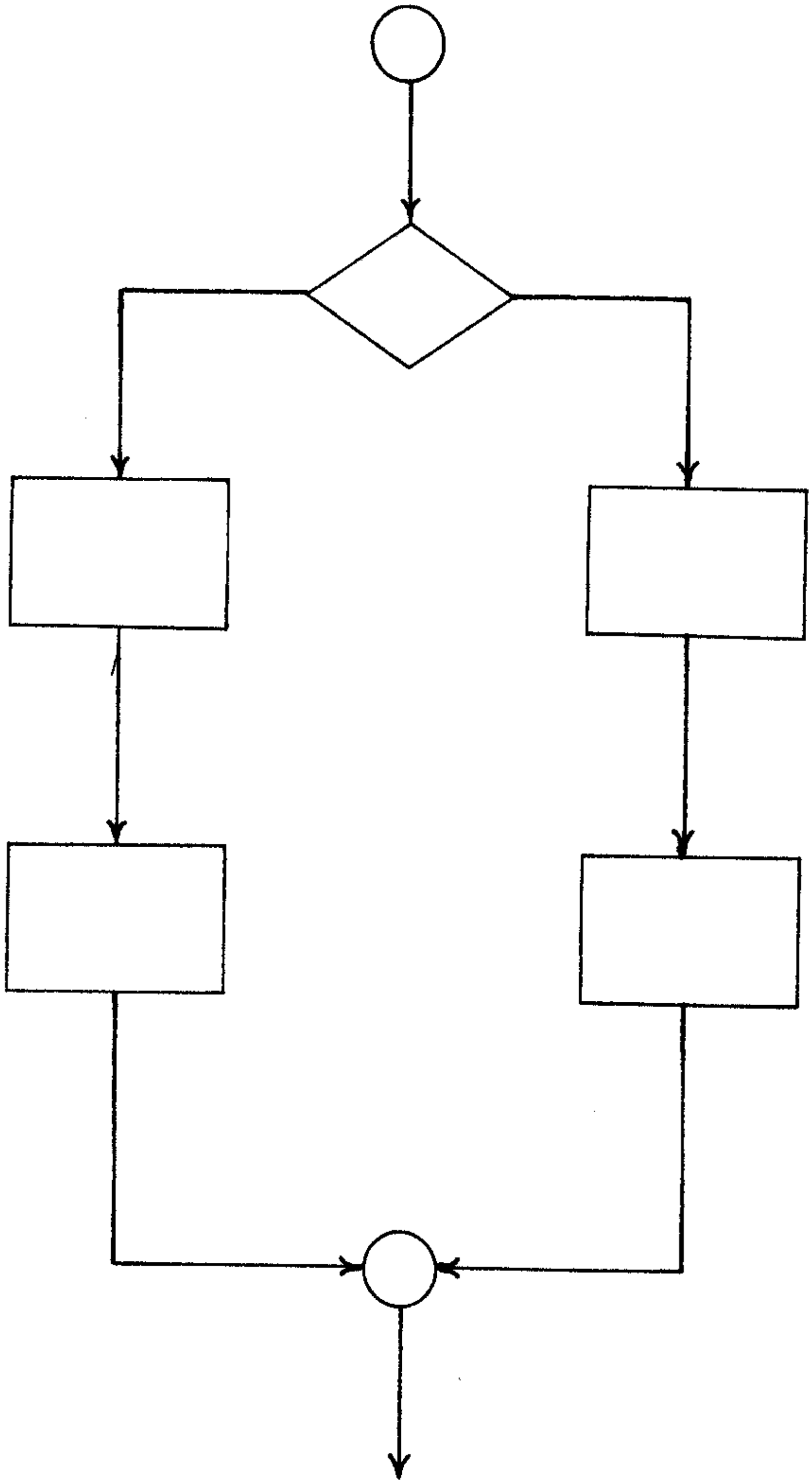

Fig. 5--A selective structure

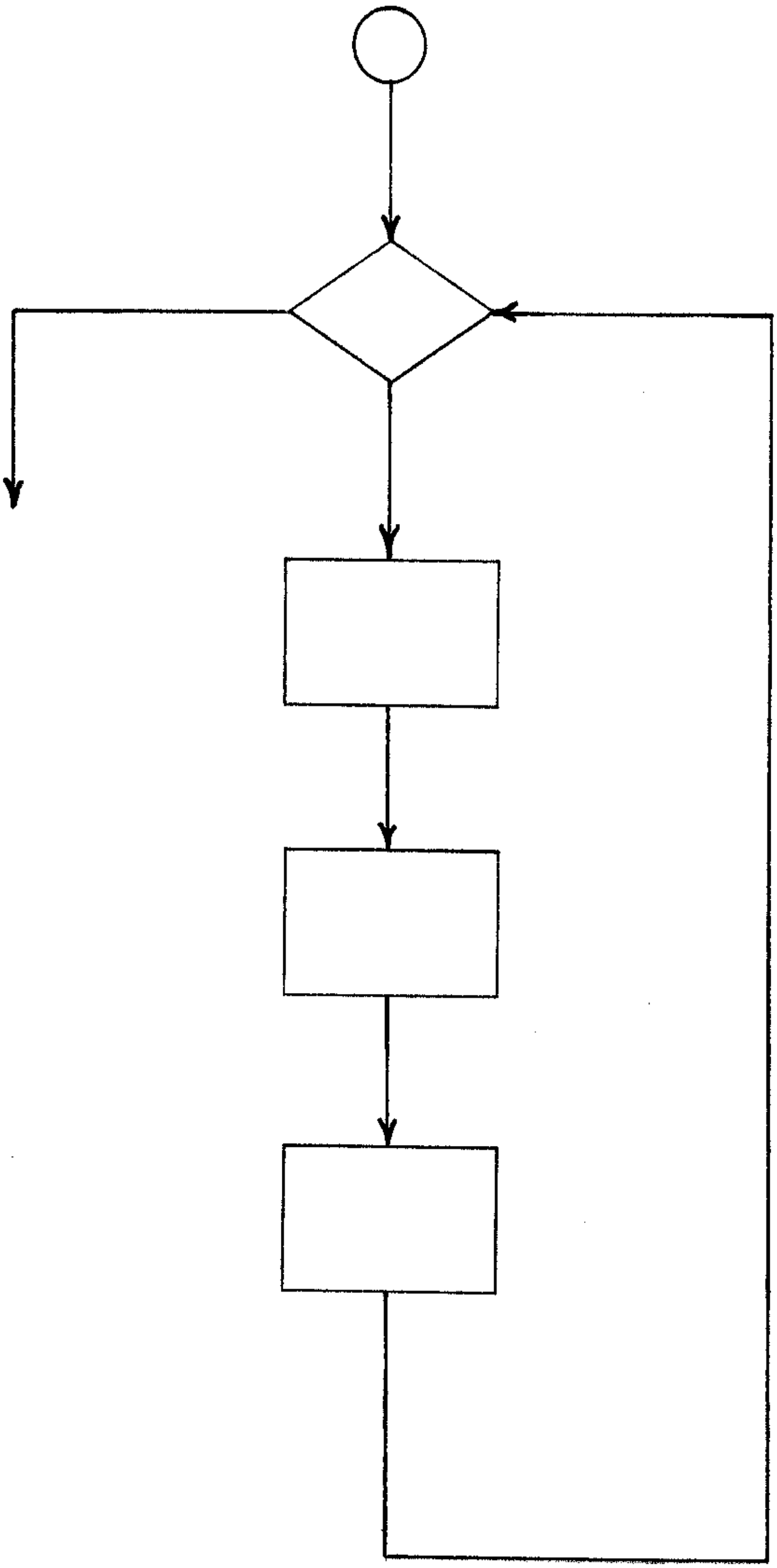

Fig. 6--A DO WHILE structure

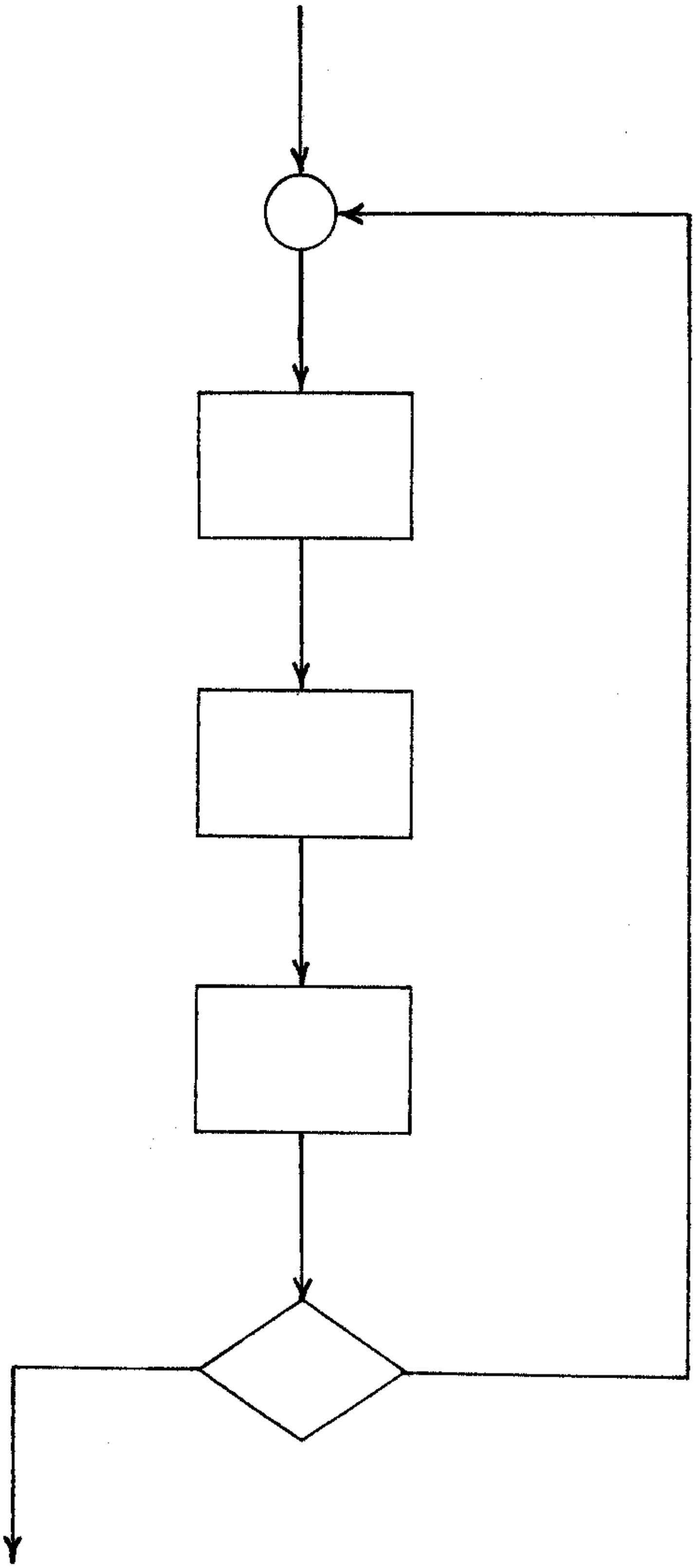

Fig. 7--A DO UNTIL structure

represent the selective mechanisms which determine the number of repetitions. There are two forms of this structure. The first one, Figure 6, tests the looping condition before loop execution. This is implemented in PL/I with the DO WHILE statement. The second form, Figure 7, implemented in ALGOL with the DO UNTIL statement and in FORTRAN with the DO statement, tests the looping condition after each execution of the loop. When the looping condition is satisfied, the repetitive structure passes program control to the instruction immediately following the loop.

Any program logic can be performed by some combination of these structures. They may be sequenced one after another or nested in any combination. As an example, Figure 8 shows the nesting of several structures. A repetitive block is contained within a selective block which is itself nested within another selective block. Notice that the one entry and one exit feature is maintained within the logic structure.

A simple program which uses only the structures suggested by Bohm and Jacopini is diagrammed in Figure 9. The program includes combinations of all three structures and can be decomposed into the various structures used. Each structure maintains a single entry and single exit point; this insures a certain degree of consistency in program design.

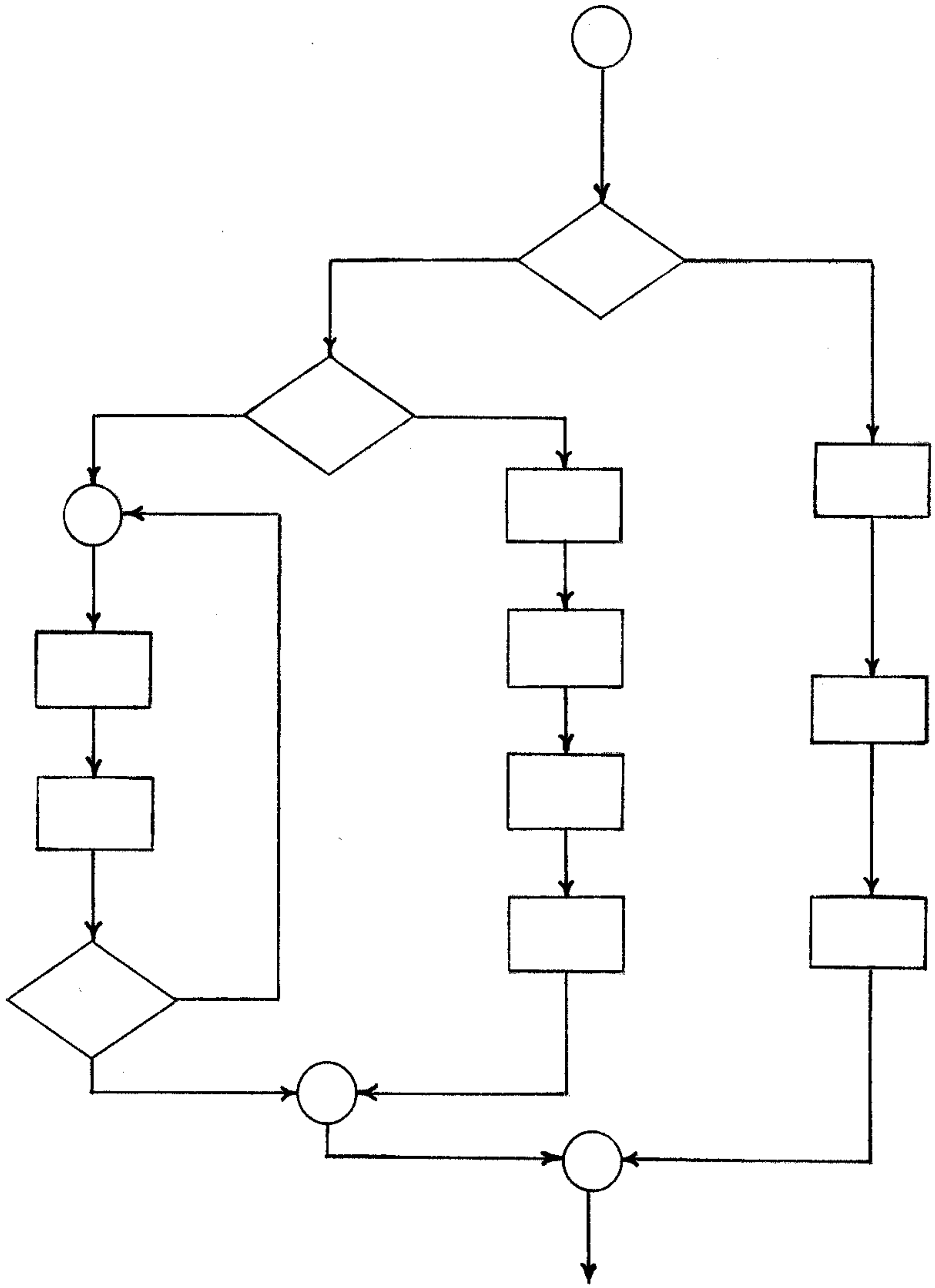

Fig. 8--Nested control structures

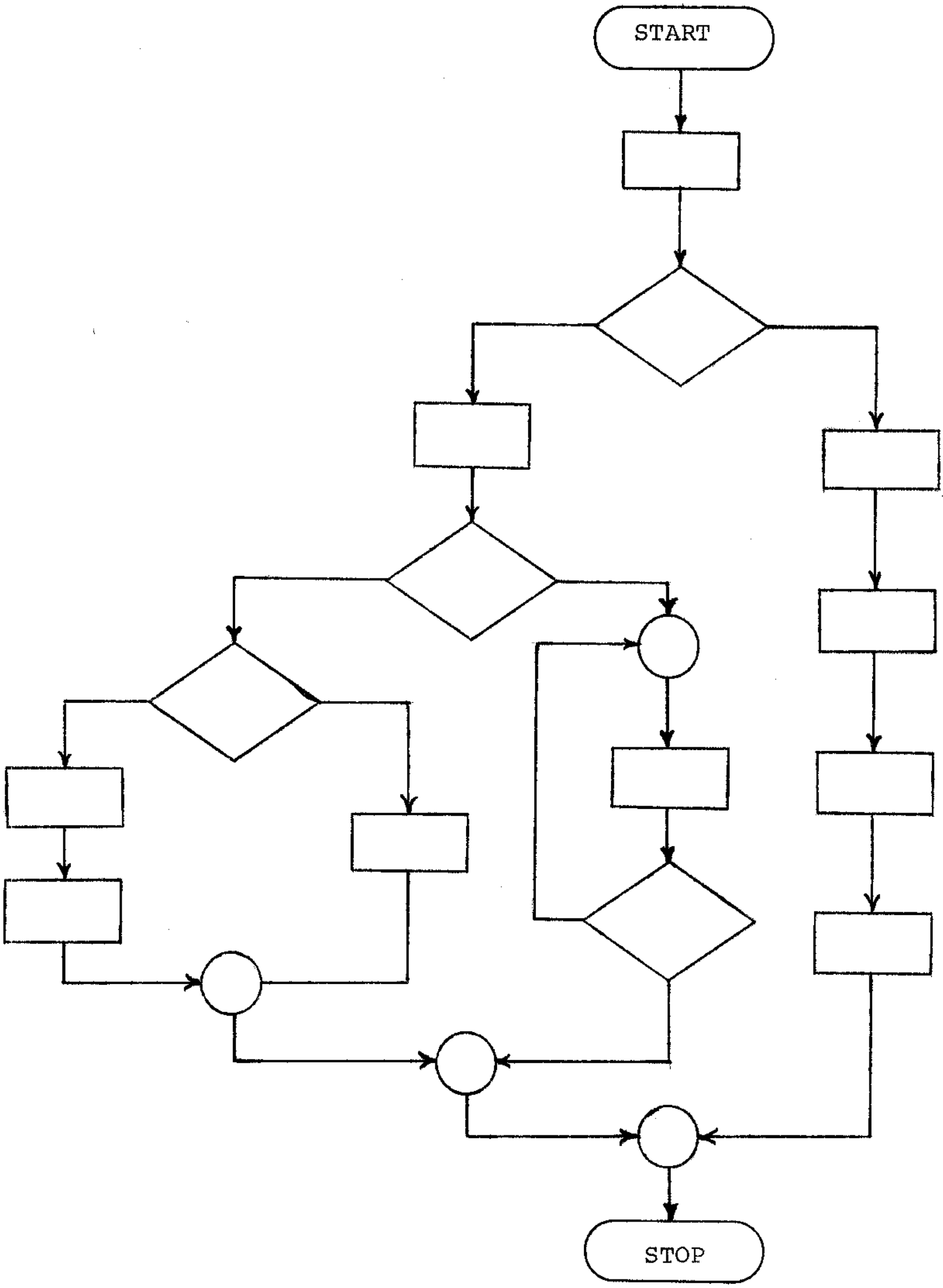

Fig. 9--Decomposition of program structures

Figure 10 represents a program containing control structures other than the three just described. In the figure blocks one through four appear to form a DO UNTIL structure; however, the loop condition may possibly be overridden by decision block two which provides a branch from within the loop. In blocks nine, eleven, and twelve a similar loop situation exists in that there are two decision blocks and two exits from the loop. In the second situation, however, the logic is further complicated by the fact that different paths are taken depending upon the point of departure from the loop.

Figure 10 is a reasonably short program, yet its unusual control structures may prove difficult to debug. Also problems may arise in understanding the purpose of the code because of the rather complicated logic structure. It appears that the programmer has developed control structures to accomplish the program logic, but he has made no attempt to develop a relatively uncomplicated or readable program design. By adhering to the use of the three simple and sufficient control structures, the programmer can define the logic of any program in a manner that will require only simple control structures which may be combined in larger decomposable structures.

Donaldson (4, p. 53), however, describes two occasions where the strict use of only the three mentioned structures will result in inefficiency.

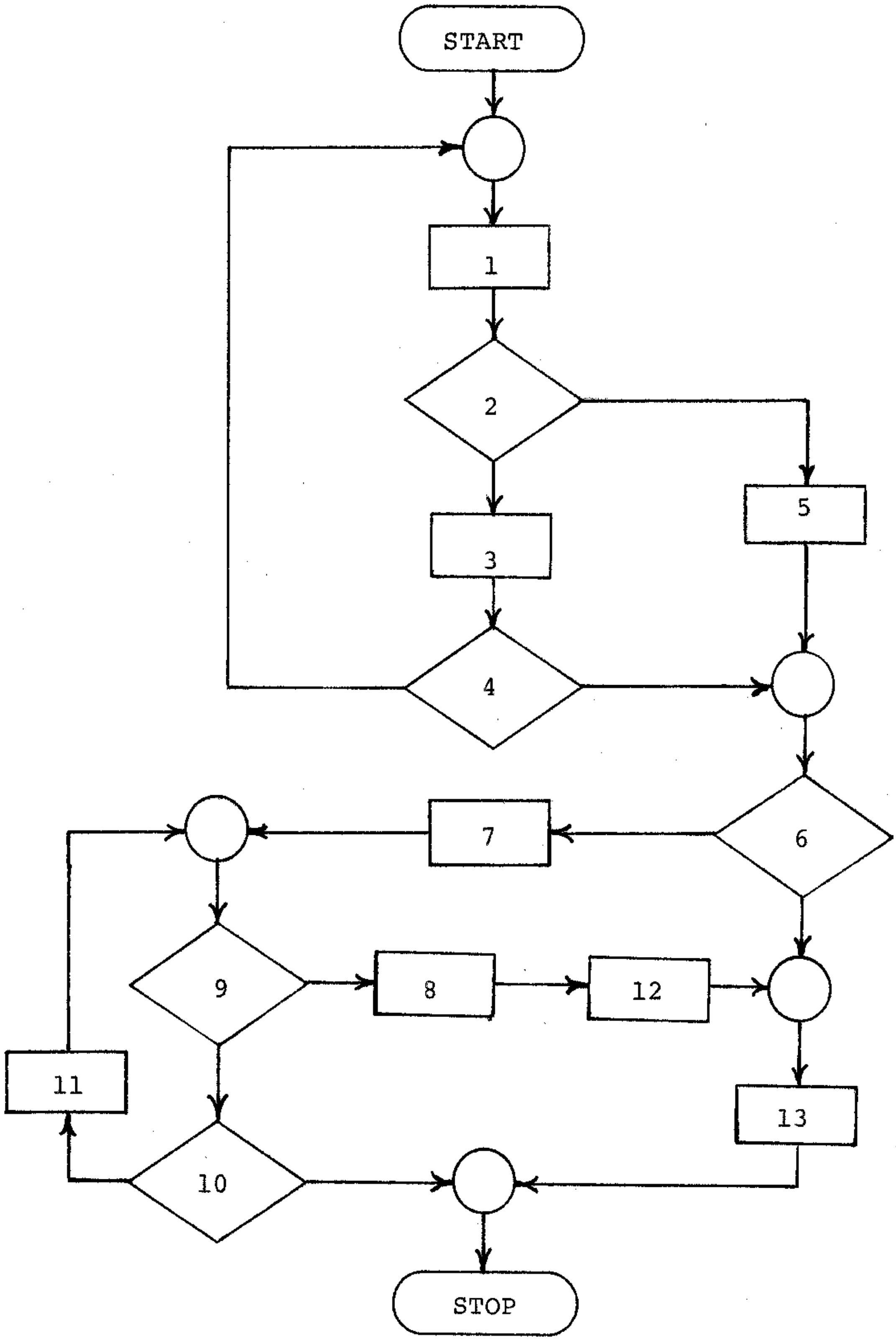

Fig. 10--Improper use of control structures

The first situation occurs when a multivalued selective structure, as shown in Figure 11, is required. Such a situation arises when only one of the three operations is to be performed, depending on whether a variable is less than, equal to, or greater than zero. A control structure implemented for this use is the FORTRAN computed GOTO statement.

To implement a multivalued selective structure using only the three structures of Bohm and Jacopini, a selective structure must be nested within another selective structure, as in Figure 12. Compared to the computed GOTO, this nested form of the selective structure may be grossly inefficient and less readable since it requires an unnecessary ordering of decision statements.

The second instance of inefficiency occurs when the abnormal termination of a repetitive structure is required. This situation is illustrated in Figure 13. Although this structure violates the single entry and single exit rule, Donaldson (4, p. 53) suggests that such a structure may save considerable time and space. Since this structure contains two possible loop-terminating conditions, its use should be properly flagged.

The three basic control structures can usually be approximated in most higher languages. They are directly implemented in PL/I with the IF THEN ELSE and DO constructs, in ALGOL with the IF THEN ELSE and the FOR constructs and

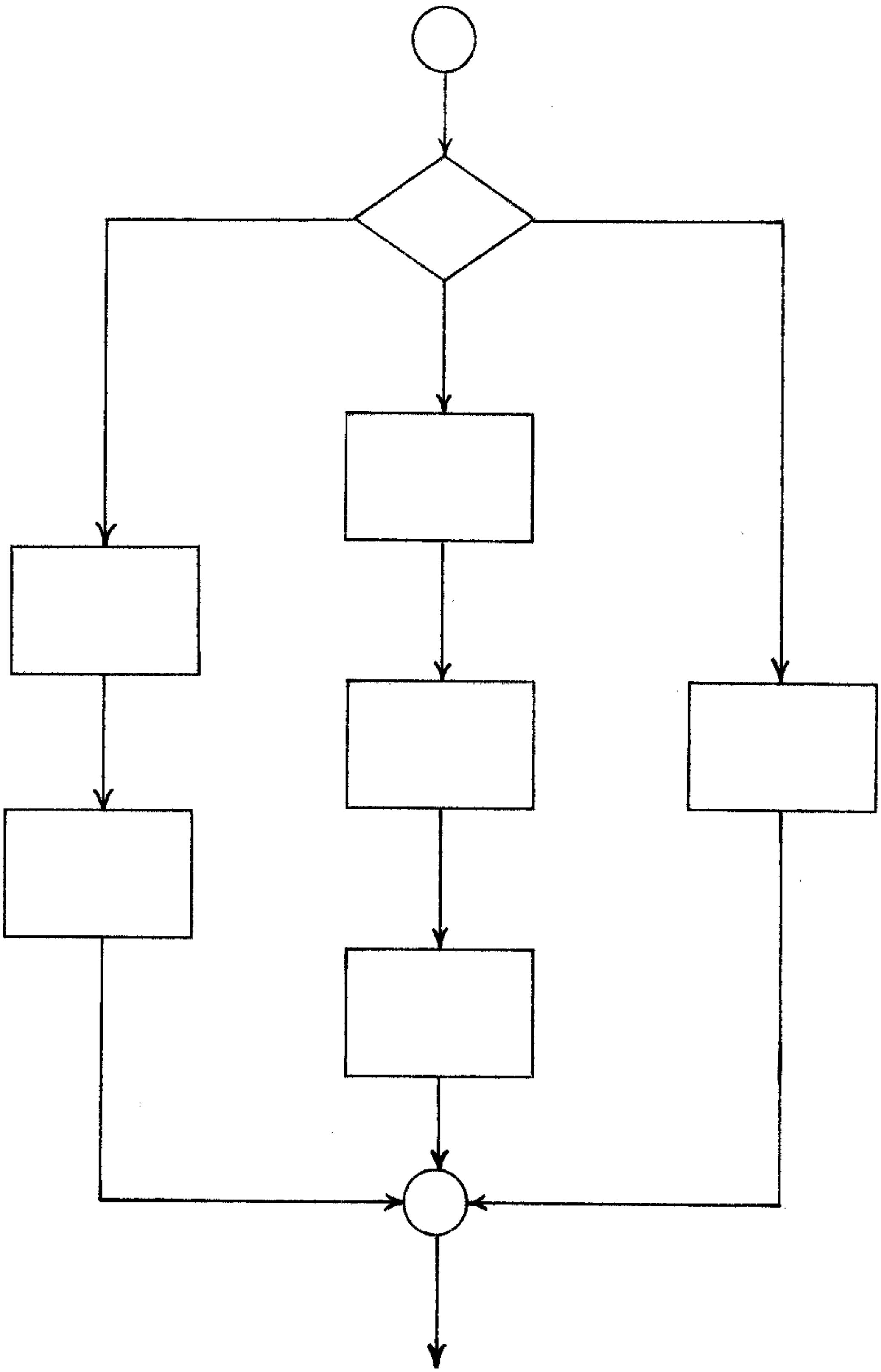

Fig. 11--Multivalued selective structure

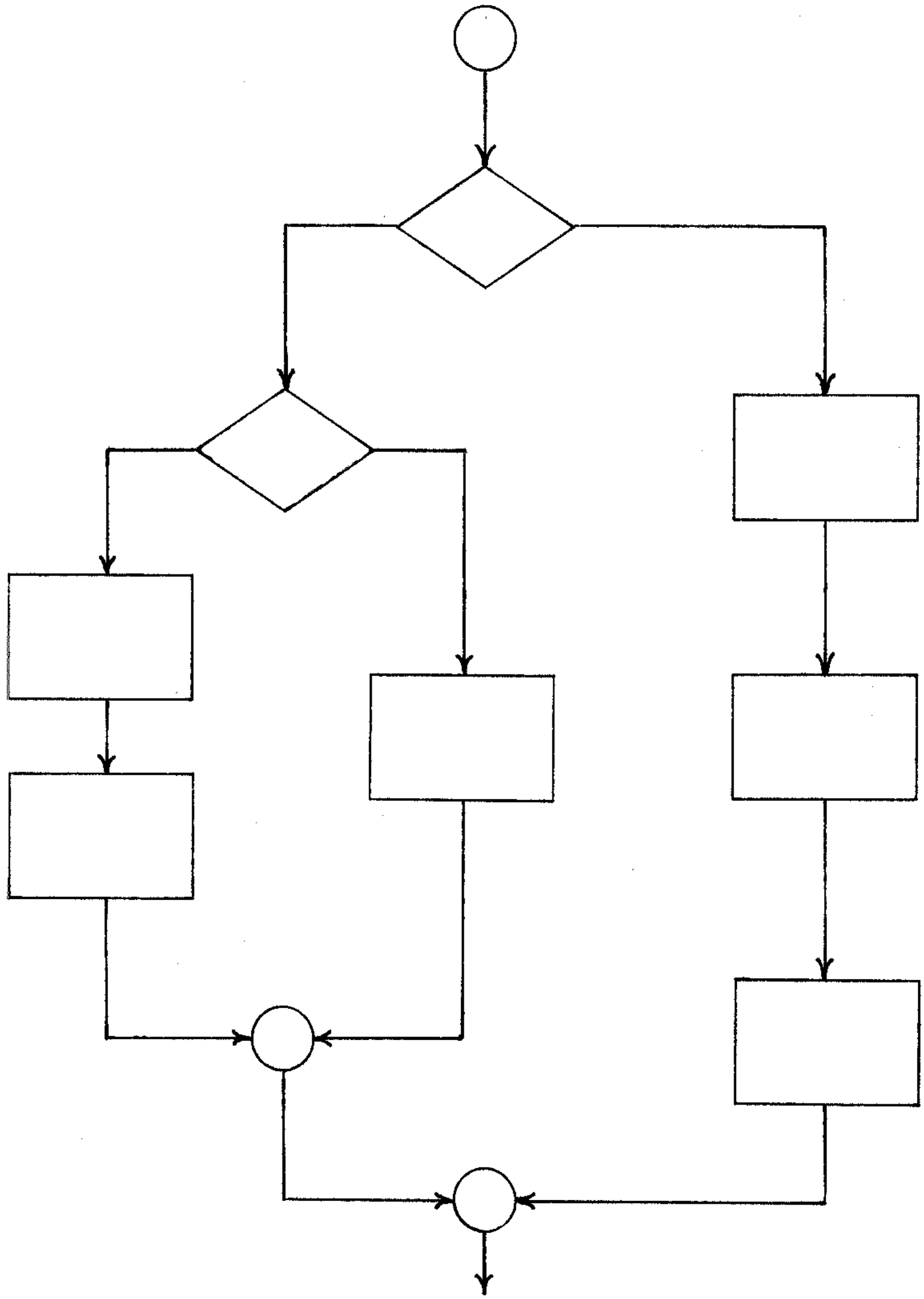

Fig. 12--Multivalued selective structure implemented by nesting simple selective structures.

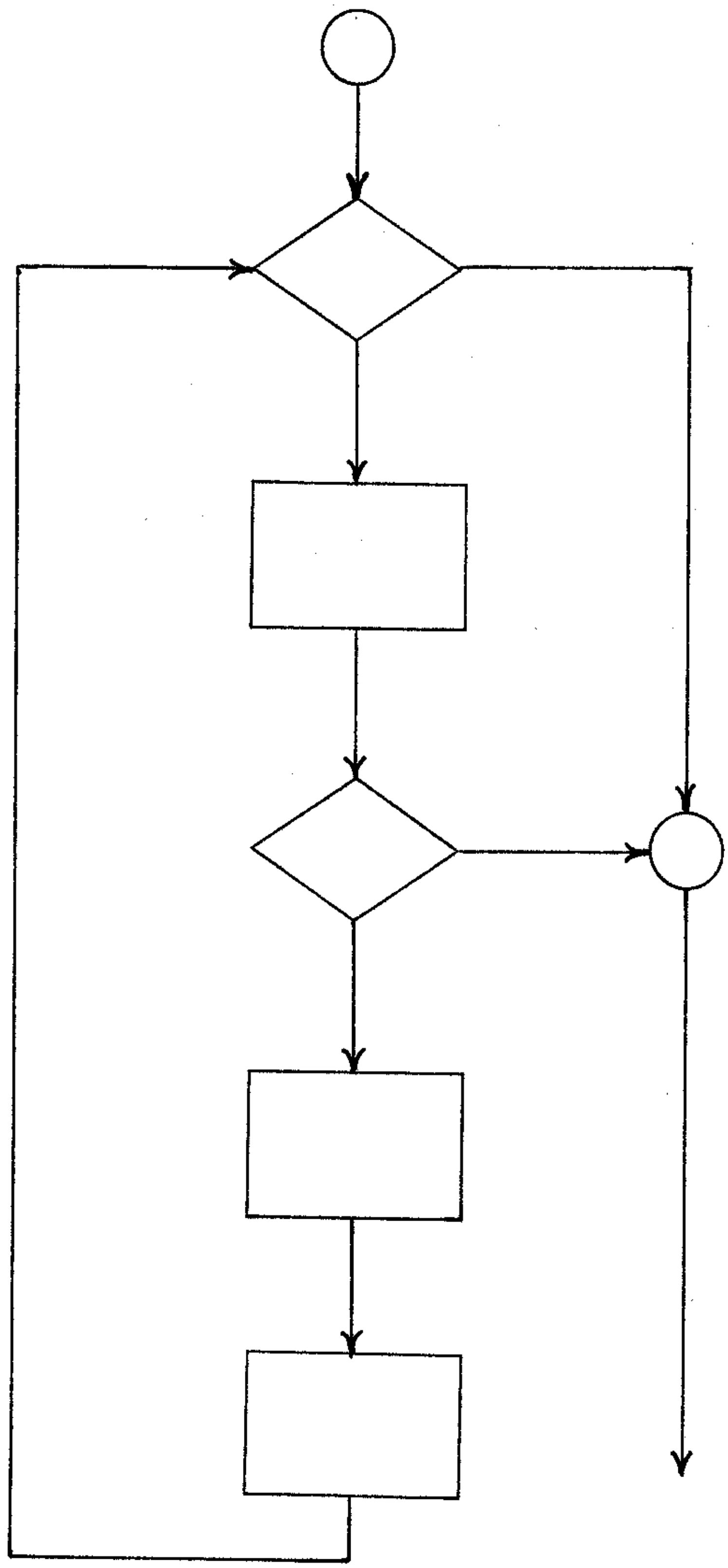

Fig. 13--Abnormal termination of a repetitive structure.

in COBOL with the IF THEN ELSE and the PERFORM constructs. FORTRAN has an IF statement which can approximate a selective structure and a DO construct for repetition (13, p. 111).

Assembly languages do not possess IF statements or DO constructs, but these structures can be approximated if macro processing is available. The use of macros to implement these constructs will make the structure of an assembly language program more visible since macro instructions tend to stand out among other assembly language instructions. Without macros, however, assembly language programming is limited to a few conditional and unconditional branching mechanisms which can only simulate selective and repetitive structures with a great deal of awkwardness, and which tend to obscure program structure.

Significance of Simple Control Structures

The major advantage of limiting the programmer to the three sufficient control structures is that it forces him to design more carefully (1, p. 146). With simplified logic he must take care in deciding how to code iterative and selective procedures and make sure that termination conditions are correct. Forcing the programmer to be careful will increase the likelihood of a correct program.

The programmer need not be concerned with a vast repertoire of program structures if he is limited to three.

He need only decide how to combine these three to satisfy his problem. With such simple logic his programs will more likely be understood. Diagrams, such as Figure 7, can be used to illustrate program logic and program decomposition.

Since each of the three structures maintains a single entry and exit point, the program lends itself to segmentation and modularity. Such programs are more readily verified since testing can be done on program segments rather than on the program as an entity.

If only the three structures are used, program notation can be greatly simplified and easily understood. A notation such as the following:

```
DO
 .
 .
 .
End DO
```

and

```
IF . . . .
      THEN
       .
       .
       .
      END THEN
      ELSE
       .
       .
       .
      END ELSE
```

will suffice to define most program logic. Programs will not contain complicated or vague structuring mechanisms which tend to obstruct readability and clarity. If the single entry and exit rule is maintained while using this notation, the program can be read in a linear fashion making program logic easier to follow.

While these three structures are sufficient to code any programmable logic, it has been pointed out (4) that they may be inefficient in some cases. Limitations on code size or execution time may force the programmer to use other structures to improve code efficiency. When there are not, however, strict limitations on code size and execution time, the use of this simple set of structures should provide overall efficiency, with respect to program readability, accuracy, and maintenance. Although a program may require more time to design and may be less efficient in terms of code usage and execution time, the debugging time will be considerably less and program maintenance considerably easier. A certain amount of assurance in the quality of the program can be implied if the constructs used are simple (1).

The use of a simple but sufficient set of control structures can improve almost every aspect of program design and coding. Program features most likely to be negatively affected by these structures include: time required for program design since the designer is limited by the logic structures available to him, and program length and execution time since the set of structures are sometimes inefficient.

The benefits of using the simple structures seem to outweigh their disadvantages. The author believes that the use of a simple set of structures will improve most programs especially in terms of functional correctness and program readability. The use of additional constructs should be avoided whenever possible, but when they are utilized, care should be taken to insure that their presence has been properly documented. It is much too easy for a program to become unnecessarily complex when a variety of constructs are used indiscriminately by a careless programmer.

The GOTO Controversy

In regard to the use of other control structures in programming besides the three discussed above, much attention has been directed to the use of GOTO statements. Dijkstra (3) and Mills (9) are just two of the many authors who have directly attacked the use of GOTO constructs in programming. Hopkins (6), on the other hand, suggests that while the GOTO is commonly abused in programming today, it is not necessary or even desirable to discontinue its use in computer languages.

In an effort to define a structured programming technique, the elimination of GOTO constructs is sometimes presented as the only issue (8, p. 51). Indeed, Hopkins (6, p. 55) suggests that the GOTO issue has been greatly over-emphasized. Programmers looking for a simple solution to the problems of programming today have used the GOTO as a scapegoat. While Hopkins (6) agrees that the removal of GOTO's will improve the code of most programmers, he does not consider this sufficient grounds for eliminating the structure from programming.

Removal of the GOTO statement is suggested mostly on the arguments of programming clarity and simplicity of logic. Most studies expound methods of improving GOTO programs by the substitution of other control structures; few, however, discuss the cases where these other structures will themselves produce inefficient code or perhaps some other undesirable side effect. Attention has been focused on the elimination of the GOTO statement and, for the most part, has ignored the problems which programming without the GOTO may introduce.

One of the major criticisms of the GOTO structure is its use in multiple exits to different program locations from within a loop. Consider the following FORTRAN code whose structure is diagrammed in Figure 14.

```
       .
       .
       .
   DO 1 I=1,50
   X = X + 1.
   Y = Y + 1.
```

```
      IF (M .EQ. 10) GO TO 3
      M = Y * 2
      IF (M .EQ. 200) GO TO 4
      T = T + 1.
1     CONTINUE
         .
         .
3     ......
         .
         .
4     ......
         .
         .
```

The loop contains three possible exit points, each of which branches to a different location in the program, and consequently, produces a somewhat complicated execution sequence. When the loop has terminated, program control will be at one of three possible locations, and the problem of determining which location may be significant. If the programmer is accustomed to unrestricted use of the GOTO, loops similar to the one just described may be common in his programs.

While it is desirable to avoid this type of logic, eliminating the GOTO will not necessarily prevent multiple exits to locations outside the control of the looping structure. The following PL/I code produces a multiple-exit loop, yet it contains no GOTO structures. Here the programmer has used the RETURN statement just like a GOTO, i.e., to escape the control of the loop and branch to a location outside the range of the loop. Thus, merely

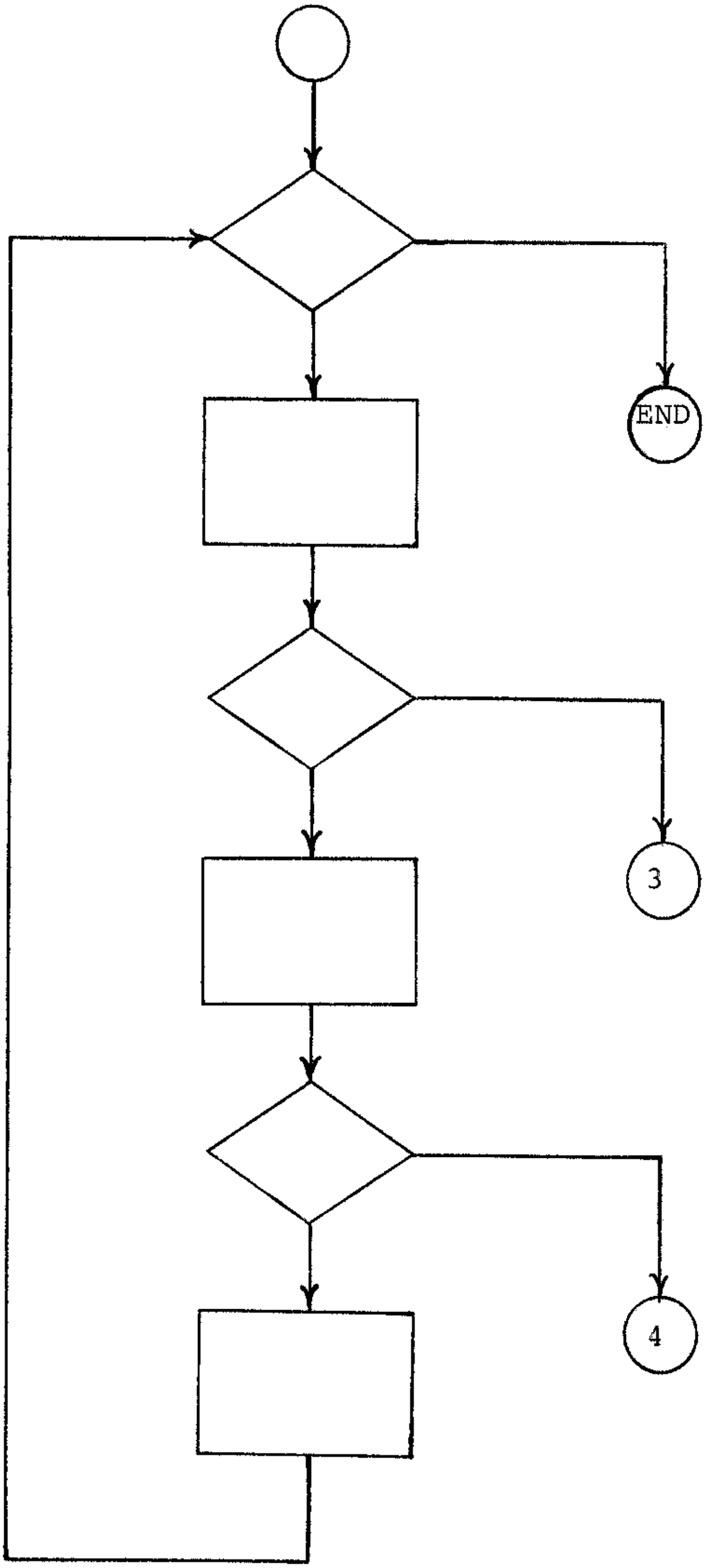

Fig. 14--A DO loop with multiple exits

eliminating GOTO statements will not prevent awkward control paths or complex logic structures.

```
DO I = 1 TO 10;
X = X + 1;
Y = Y + 10;
IF X = 100 THEN RETURN;
M = Y * 2;
IF M = 200 THEN RETURN;
T = T + 1;
END;
```

Multiple loop exits will not necessarily complicate the logic structure. If the exits all branch to the end of the loop, then the block structure which the loop represents is maintained. In effect, the loop has executed or partially executed some number of times and control passes to the location just past the loop.

On occasion a programmer will discover that a large code segment is not functionally correct, and with the addition of a GOTO statement the program can be made to execute correctly. In this case the insertion of a GOTO may save a considerable amount of time, especially if major code alteration is required to correct the mistake without using the GOTO; moreover, if a functional program were required in a short time, the programmer may be forced to use the GOTO since major code revision is usually time-consuming.

If used with discretion then, the GOTO may be an appropriate means of solving an awkward logic problem. Also since some programs are used only once or are used only by the programmer himself, the need for simplicity or clarity in a program may not be felt; the programmer may see no justification for coding without the GOTO. Although every program can be written without the use of GOTO statements, not every program need be so written.

The program which must maintain strict limits with regard to code efficiency or length may be possible only with the use of GOTO statements. While this situation should be avoided whenever possible, the fact that it does occur suggests that, from a practical point of view, the GOTO construct may be quite useful.

Some of the reasons for eliminating the GOTO have already been mentioned. They include: destruction of program linearity when GOTO constructs are freely used; increase in program complexity due to GOTO branches; and the decrease of program modularity, clarity, and readability due to the presents of GOTO constructs.

The strongest argument against the GOTO statement is the fact that there are so many ways to use it incorrectly. The presence of a GOTO is more likely to be the result of poor programming techniques than it is a sound logic structure. Since the GOTO is not needed and since it is so apt to be misused, there is good reason to desire its removal.

It is suggested by Wulf (14, p. 68) that efficiency should not be used as an excuse for GOTO programming. Efficiency of program code should be achieved through a highly optimizing compiler; the programmer should not have to resort to introducing GOTO constructs for the sake of improving run time. Code optimization during compilation is more effective if the programmer has used a block-structured program design and has avoided the use of GOTO structures which unnecessarily disrupt program modularity.

Programming with GOTO's necessitates the use of labels, a factor which adds to program complexity since a label referenced in a program instruction requires a search through the source code to locate the label and interpret the referencing instruction. One of the basic reasons for developing a structured programming style is to achieve a certain degree of program simplicity. Removing the GOTO's will reduce the need for labels, and hopefully, simplify program logic by reducing the number of label references.

The author is of the opinion that programming without the GOTO in most cases will improve the quality of a program; however, to completely eliminate the GOTO seems to be unjustified at this time. It should be remembered that there are circumstances where the GOTO is useful even though it may produce awkward code or undesirable program logic. Although it lends itself to misuse, the GOTO is not unique in this respect. Many features of a language may be subject to improper or inefficient use, yet it would be impractical to remove all of them.

Subroutines

Another control structure which deserves special attention is the subroutine call. Like the GOTO statement the subroutine call causes an unconditional branch. The branch passes control from the main program to an external set of instructions, referred to as the subroutine or subprogram, which are then executed. Once the subroutine has been executed, program control returns to some location in the main program.

The use of the subroutine call does not disrupt the structure of a repetitive, selective, or sequential construct within the main program when the subroutine satisfies two conditions. First, if the subroutine returns control to the statement immediately following the subroutine call, then linearity has been preserved. Three sequential statements will have been executed: these statements are the one preceding the call, the subroutine call, and the statement following the subroutine call. The second condition requires that the subroutine itself consist of some combination of the three simple control structures.

Some subroutines allow multiple entry and exit points in addition to multiple return locations. The legality of their use in terms of structured programming control structures seems to be unclear at this time since most authors, when discussing control structures of structured programming, do not mention subroutines specifically. Program modules is a frequently-used term which may or may not include subroutines. Mills (10), when discussing control structures, uses the term segments. He suggests (10) that program segments should have only one entry and one exit point, but it is unclear whether or not he is applying this rule to subroutines as well.

It seems reasonable to assume that the concepts of control structures discussed in this chapter would apply directly to subroutines. The subroutine is a block of code that is represented in a program by a single instruction, the subroutine call. The subroutine, when inserted in place of the call, should produce a logic structure conforming to the principles of structured programming.

Conclusions

The theory of control structures plays a key role in the development of a structured programming technique;

however, control structures alone will not provide the solution to the present software dilemma. All aspects of programming, including data structures, must be investigated to develop the most effective approaches to program design and constructions.

CHAPTER BIBLIOGRAPHY

1. Benson, Jeoffrey, "Structured Programming Techniques," Record of the 1973 IEEE Symposium on Computer Software Reliability, New York (April, 1973), 143-147.

2. Bohm, C. and G. Jacopini, "Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules," Communications of the ACM, IX (May 1966), 366-371.

3. Dijkstra, E. W., "GOTO Statement Considered Harmful," Communications of the ACM, XI (March, 1968), 147-148.

4. Donaldson, James, "Structured Programming," Datamation, XIX (December, 1973), 53-54.

5. Fisher, David, "A Survey of Control Structures in Programming Languages," SIGPLAN Notices, (November, 1972), 1-13.

6. Hopkins, Martin, "A Case for the GOTO," SIGPLAN Notices, (November, 1972), 59-62.

7. Leavenworth, B. M., "Programming With(out) the GOTO," SIGPLAN Notices, (November, 1972), 54-58.

8. McCracken, Daniel, "Revolution in Programming: An Overview," Datamation, XIX (December, 1973), 50-52.

9. Mills, H. D., How to Write Correct Programs and Know It, FSC 73-5998, IBM, Gaithersburg, Maryland, (December, 1972).

10. ____________, "On the Development of Large Reliable Programs," Record of the 1973 IEEE Symposium on Computer Software Reliability, New York (May, 1973), 155-159.

11. ____________, Mathematical Foundations for Structured Programming, IBM FSD Report FSC 72-6012, IBM Gaithersburg, Maryland, (February, 1972).

12. Peterson, W. W., et al., "On the Capabilities of While, Repeat, and Exit Statements," _Communications of the ACM_, XVI (August, 1973), 503-512.

13. Tenny, Ted, "Structured Programming in FORTRAN," _Datamation_, XX (July, 1974), 110-115.

14. Wulf, William, "A Case Against the GOTO," _SIGPLAN Notices_, (November, 1972), 63-69.

CHAPTER IV

CONTROL STRUCTURES IMPLEMENTED WITH MACROS FOR ASSEMBLY LANGUAGE PROGRAMMING

In IBM 360 assembly language programming (7) there are no instructions which will provide for the documentation and execution of selective and repetitive control structures described in the previous chapters. Like most assembly languages the IBM 360 language contains only a set of conditional and unconditional branching instructions of which the most sophisticated include the BXLE, BSH, and BCT instructions (6, pp. 66-67). With the use of labels and these branching instructions available to him, the programmer may simulate in ALC the selective and repetitive structures he needs. The additional instructions required for implementing these structures, however, may significantly increase the complexity of the code and reduce the clarity of the program. Because there are no adequate control structures available in ALC, the programmer is forced to improvise looping and selective structures each time they are required in a program.

A set of IBM 360 assembly language macros to be used as selective and repetitive control structures were developed in this study. These macros should provide at least two important benefits to the ALC programmer. First, the burden of formulating such control structures when they are required is removed from the programmer; he needs only supply the necessary symbolic parameters for the macro which will generate the desired control structure.

The second important benefit which the macros provide is an increase in program clarity. It is readily apparent to anyone reading the code that the macro prototype statements represent looping and selective structures. The mere presence of the macro names within the code indicates the extent of control of each structure. If the NOGEN print option is used, the program source listing is significantly simplified since macro-generated code is not shown. The NOGEN option causes the suppression of print-out of all macro-expanded code. The source listing includes only ALC instructions and macro instructions as they were coded by the programmer. When the NOGEN print option is omitted and the source listing contains the expanded macro code, the program, of course, looks much like a typical assembly language program, since all of the branching instructions and labels necessary to implement the macros are shown. The Appendices contain examples of programs with unexpanded code and programs with expanded code. The differences will be readily apparent to the reader.

Repetitive Control Structure Macros

To implement a looping or repetitive control structure in ALC two macros, DO and ENDDO, have been developed. The DO macro generates the code for testing the looping condition and the code which either continues the execution of the loop or branches to terminate the loop. The ENDDO macro besides providing a label to which a branch will occur when the loop is terminated, signifies the extent of control of the DO macro.

The DO macro is similar to the DO statement in PL/I. It may be specified with just an increment and range such as DO I = 1 TO 10 BY 3, with just a WHILE condition such as DO WHILE (X LE 2) or with both a looping range and WHILE condition such as DO I = 1 TO 100 BY 3 WHILE (Y GT 30). These options are specified by means of keyword parameters attached to the macro prototype statement.

The looping conditions may be specified by literals when parameters are equated to numeric values, in registers when parameters are equated to register numbers,or by locations when parameters are equated to labels. Numeric values may not be used to specify locations. Once the DO loop parameters are determined, they are maintained in a stack generated by the macro; the programmer need not be concerned with destroying looping variables initially specified in registers or locations. He is free to use registers and locations originally containing looping parameters as he wishes without affecting the state of the DO loop parameters. Table I contains an explanation of each DO macro parameter.

TABLE I

KEYWORD PARAMETERS FOR THE DO MACRO

Keyword Parameter	Meaning	Characteristics of Value
LABEL	Label to identify corresponding ENDDO macro.	Any valid ALC label.
LOWNUM	Value of the lower range of a DO loop.	Integer.
BYNUM	Value of increment of a DO loop range.	Integer.
HGHNUM	Value of the upper range of a DO loop.	Integer.
LOWREG	Register containing the lower range of a DO loop.	Integer from 0 to 15.
BYREG	Register containing the increment of a DO loop range.	Integer from 0 to 15.
HGHREG	Register containing the upper range of a DO loop.	Integer from 0 to 15.
LOWLOC	Location containing the lower range of a DO loop.	Any valid ALC label.
BYLOC	Location containing the incremenet of a DO loop range.	Any valid ALC label.
HGHLOC	Location containing the upper range of a DO loop.	Any valid ALC label.
DOLOOP	Indicates the presence of a DO loop.	Any non-null value
WHILE	Indicates the presence of a DO WHILE condition.	Any non-null value.

TABLE I--Continued

Keyword Parameter	Meaning	Characteristics of Value
WOP	Logical operator for a DO WHILE condition.	LT,LE,EQ,GE, GT, or NE.
WREGA	Register containing the left operand of a DO WHILE condition.	Integer from 0 to 15.
WREGB	Register containing the right operand of a DO WHILE condition.	Integer from 0 to 15.
WLOCA	Location containing the left operand of a DO WHILE condition.	Any valid ALC label.
WLOCB	Location containing the right operand of a DO WHILE condition.	Any valid ALC label.
WLITA	Numerical value representing the left operand of a DO WHILE condition.	Integer.
WLITB	Numerical value representing the right operand of a DO WHILE condition.	Integer.

The ENDDO macro utilizes only one keyword parameter LABEL. This parameter should be equated to the value of the LABEL parameter of the DO macro associated with the ENDDO macro.

Examples of the DO macro are shown in Figure 15. Macro A represents a simple DO loop. The lower range is specified in register three (LOWREG=3), the upper range is a numerical

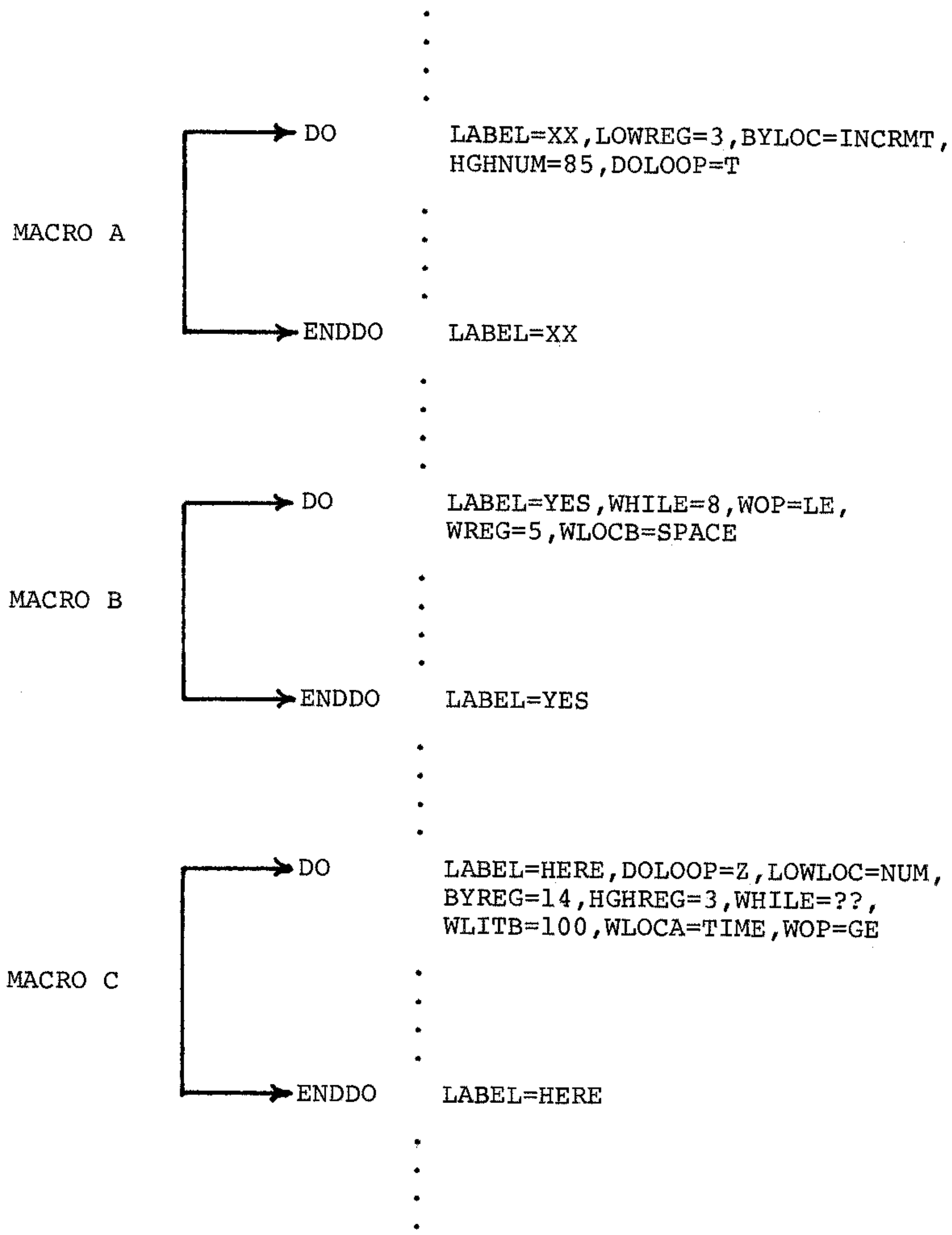

Fig. 15--Examples of DO macro parameters

eighty-five (HGHNUM=85), and the increment for the loop is contained in location INCRMT (BYLOC=INCRMT). Note that the DOLOOP parameter has been made non-null. This is required when a DO loop is specified. Macro B represents a simple DO WHILE loop. The loop will be executed as long as the contents of register five (WREGA=5), is less than or equal to (WOP=LE), the contents of location SPACE (SLOCB=SPACE). The WHILE parameter must be made non-null when a DO WHILE loop is specified. Macro C represents a combination of a DO loop and DO WHILE condition. In this macro both the DOLOOP and WHILE parameters must be made non-null. The lower limit of the DO loop is contained in location NUM, (LOWLOC=NUM), the upper limit is in register three (HGHREG=3), and the increment is in register fourteen (BYREG=14). The WHILE condition is satisfied when the contents of location TIME (WLOCA-TIME), is greater than or equal to (WOP=GE), a numerical one hundred (WLITB=100). When both a DO loop and DO WHILE condition are specified, each is tested before the loop is executed. The basic logic of a DO and ENDDO construct is shown in Figure 16.

Since a stack is used to maintain DO loop parameters, the DO macro permits nesting. A DO macro and its ENDDO macro may be entirely contained within the bounds of a second DO and ENDDO macro. Figure 17 illustrates nesting of DO macros to the second level. Notice that the LABEL parameters of the DO and ENDDO macros must be properly

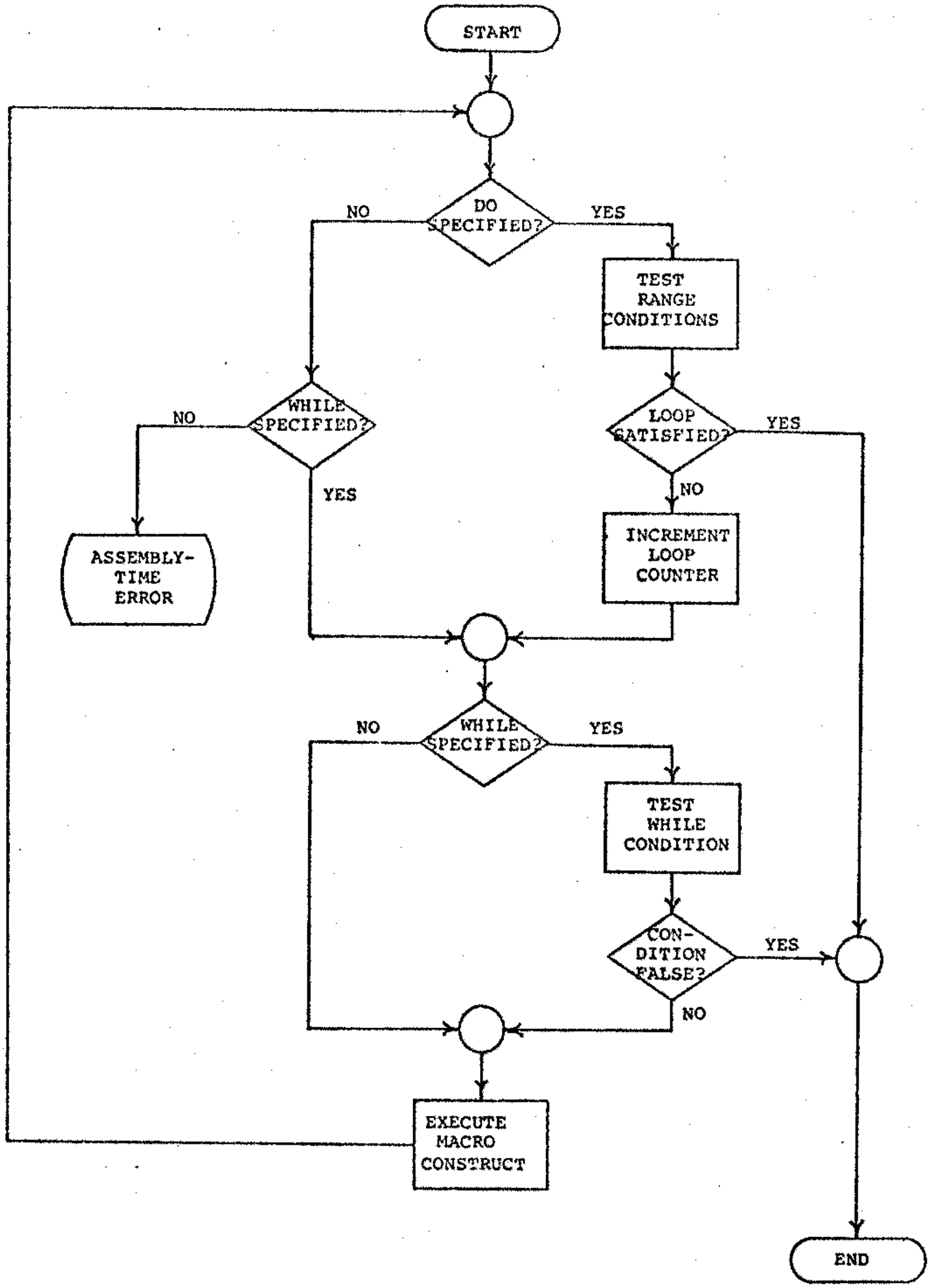

Fig. 16--Flowchart of a DO and ENDDO construct

```
           .
           .
           .
DO         LABEL=LOOPA, . . .
           .
           .
           .
           DO       LABEL=LOOPB, . . .
                    .
                    .
                    .
                    DO       LABEL=LOOPC, . . .
                             .
                             .
                             .
                             .
                    ENDDO LABEL=LOOPC
                    .
                    .
                    .
           ENDDO LABEL=LOOPB
           .
           .
           .
ENDDO LABEL=LOOPA
```

Fig. 17--DO macros nested to the second level

matched, i.e., at any given level of nesting, the LABEL parameters of the DO and ENDDO macros are equated to identical values. (A unique LABEL parameter value should be specified for each DO construct.)

Figure 18 illustrates the use of the stack in executing the nested macros of Figure 17. As each DO macro is encountered its parameters are stacked in a last-in first-out stack. Before executing a particular DO loop, its parameters are unstacked and tested. If the loop is to be executed, then the parameters are restacked; otherwise, the parameters are not restacked, and the loop is bypassed. Thus, each

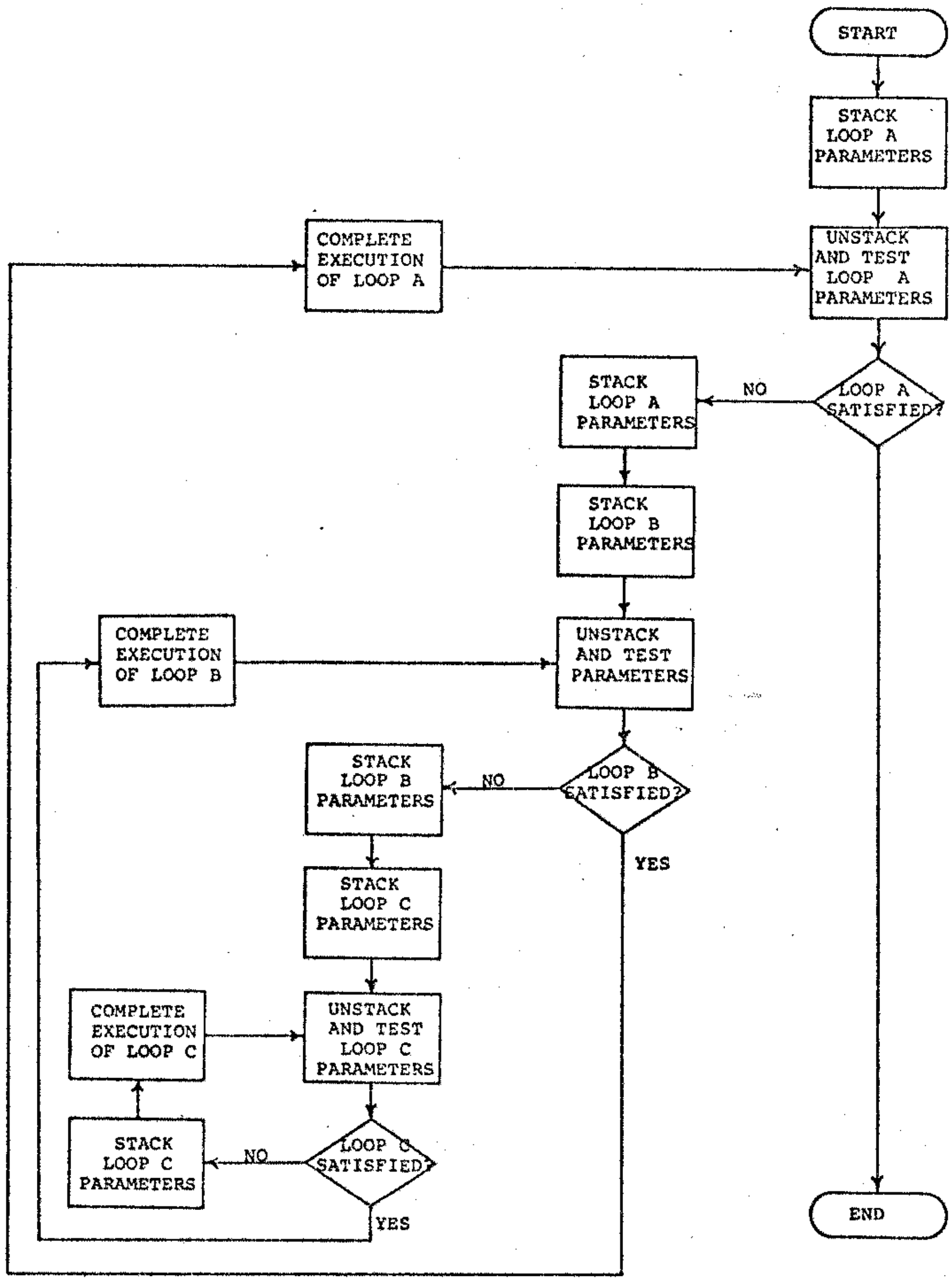

Fig. 18--Flowchart of stack logic for code in Figure 17

time LOOP A is executed the DO macro of LOOP B will be encountered, its parameters stacked, unstacked, and tested. In a similar manner, with each execution of LOOP B, the DO macro of LOOP C will be encountered, and its parameters will be stacked, unstacked, and tested.

Program instructions falling between a DO macro and its corresponding ENDDO macro represent the body of the code affected by the looping construct. This code may include any acceptable ALC instruction including macro instructions; however, the loop should not include branching instructions which transfer program control to a point outside the range of the DO macro. If the programmer wishes to terminate the looping construct abnormally, he should use the LEAVE macro, which is discussed in this chapter.

The DO macro extensively tests parameter specifications at assembly time. When an incorrectly specified parameter is detected, an appropriate error message is printed if the expanded macro code is shown, and the macro is terminated. Before termination of the macro, however, the "work" registers will be restored with their original contents so that program execution may continue. The macro, however, will have essentially been ignored by the program.

In addition to the complete source listings of the DO and ENDDO macros, Appendix A contains an expanded version of each of these macros and sample listings of macro-generated error messages.

This particular form of the repetitive control structure was selected for implementation because it offers the greatest flexibility in defining looping conditions. A loop may be defined with a simple range condition similar to the FORTRAN DO statement, with a simple comparison condition similar to a PL/I DO WHILE or with a combination of both a range condition and comparison condition similar to the PL/I statement, DO I=...TO...WHILE(......). Since conditional tests are always executed prior to loop execution, a DO UNTIL structure as illustrated in Figure 7 cannot be generated with the DO and ENDDO macros. The DO UNTIL statement tests looping conditions after the loop is executed; consequently, the loop is always executed at least once even when a condition fails on initial testing. In order to avoid any ambiguity with regard to initial loop execution, a DO UNTIL construct was not implemented. All repetitive loops will be executed only after their looping conditions are tested. If the conditional test fails initially, the loop will not be executed at all.

The LEAVE Macro

It was pointed out in Chapter III that occasionally it may be expedient to abnormally terminate a looping construct, i.e. branch to the end of the loop from a point within the loop control. Figure 13 is a construct representing an abnormal loop termination.

The DO and ENDDO macros generate instructions that will terminate a loop only when the inclusive code has been completely executed. The programmer might insert a branch instruction within the inclusive code to escape the DO macro, but such a practice is not advised, since the effects of the branch instruction may not be readily understood or perhaps may be overlooked by a programmer unfamiliar with the code.

To provide a loop escape mechanism the LEAVE macro has been developed. It permits a conditional or unconditional branch to the end of the ENDDO macro, and thus, the looping control structure will be abnormally terminated.

Figure 19 illustrates the use of the various LEAVE macro parameters. In the first LEAVE macro LABEL is equated to AAAA in order to associate it with the external DO and ENDDO macros. This LEAVE macro contains a conditional branch since the COND parameter has been made non-null, COND=F. If the number nine (LITA=9) is equal to (OPRATOR=EQ) location Y (LOCB=Y) then the LEAVE condition will be true and a branch to location AAAA, (LABEL=AAAA) will occur. The second LEAVE macro does not specify a condition and will, consequently, produce an unconditional branch to AAAA (LABEL=AAAA) when executed. The third LEAVE macro contains a conditional branch (COND=;), and the branch is executed if the contents of register ten

```
     .
     .
     .
     .
DO         LABEL=AAAA, . . . .
          .
          .
          .
          .
     LEAVE      LABEL=AAAA,COND=F,LITA=0,
                OPRATOR=EQ,LOCB=Y
          .
          .
          .
          .
     LEAVE      LABEL=AAAA
          .
          .
          .
          .
ENDDO      LABEL=AAAA
     .
     .
     .
     .
     .
     .
DO         LABEL=XYZ, . . . .
          .
          .
          .
          .
     LEAVE      REGA=10,LITB=22,OPRATOR=NE,
                LABEL=XYZ,COND=;
          .
          .
          .
          .
ENDDO      LABEL=XYZ
     .
     .
```

Fig. 19--Examples of LEAVE macro parameters

(REGA=10) are not equal (OPRATOR=NE) to a numeric twenty-two (LITB=22). The LABEL parameter (LABEL=XYZ) specifies the destination of the LEAVE branch. Table II contains a summary of LEAVE parameters, and Figure 20 describes the basic logic of the LEAVE macro at execution time.

TABLE II

KEYWORD PARAMETERS FOR THE LEAVE MACRO

Keyword Parameter	Meaning	Characteristics of Value
LABEL	Destination of a LEAVE macro producing a branch.	Any valid ALC label.
COND	Indicates the LEAVE branch is conditional.	Any non-null value.
OPRATOR	Logical operator for a conditional branch.	LT,LE,EQ,GE, GR, or NE.
REGA	Register containing the left operand of LEAVE condition.	Integer from 0 to 15.
REGB	Register containing the right operand of LEAVE condition.	Integer from 0 to 15.
LOCA	Location containing the left operand of LEAVE condition.	Any valid ALC label.
LOCB	Location containing the right operand of LEAVE condition.	Any valid ALC label.
LITA	Numerical value of the left LEAVE condition operand.	Integer.
LITB	Numerical value of the right operand of LEAVE condition.	Integer.

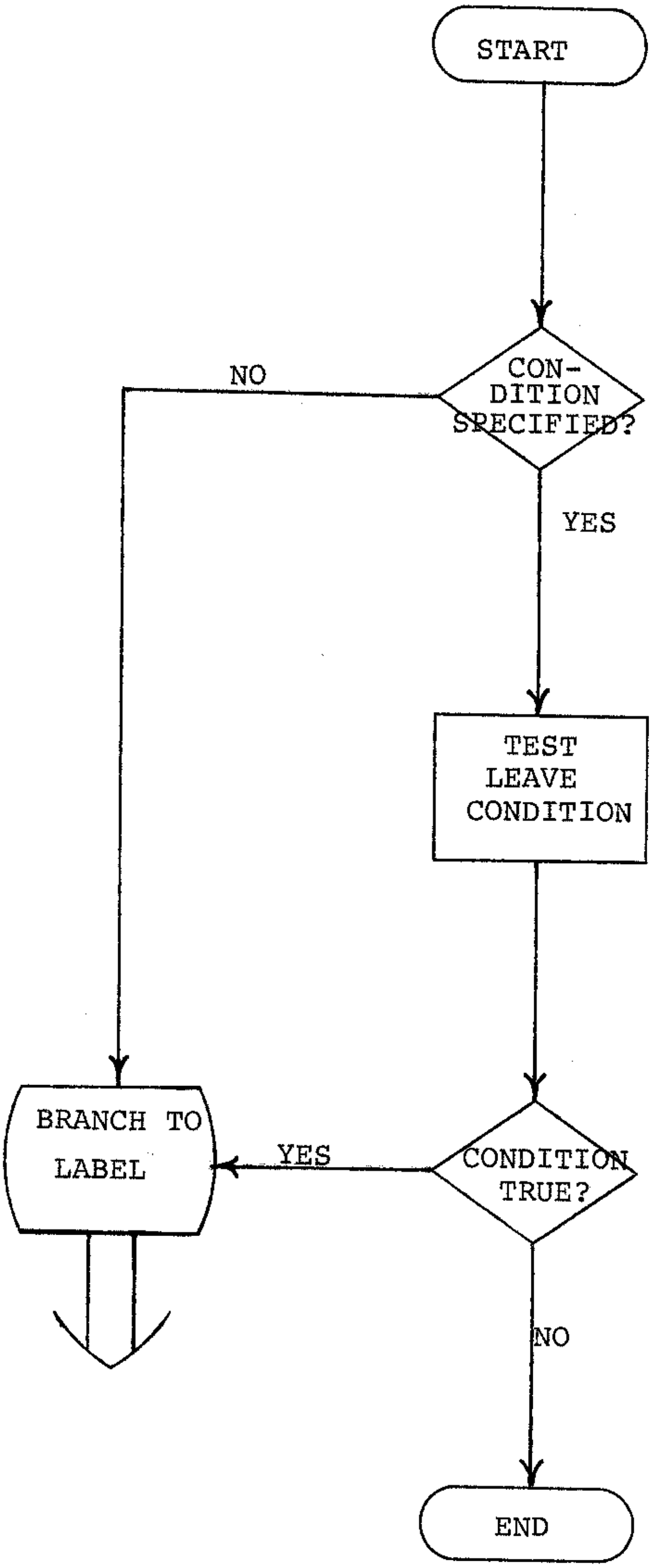

Fig. 20--Flowchart of execution of a LEAVE macro

If the LEAVE macro is used in lieu of a branching instruction, there can be no doubt as to the intentions of the code. The LEAVE statement appearing in the code will readily indicate the presence of a possible abnormal loop exit. Since the LEAVE macro may be specified with condition parameters, the programmer need not include the additional instructions to test his branching condition. He need merely specify the branching conditions by means of the keyword parameters; thus, the development of the LEAVE macro provides the programmer with a convenient and simple method for abnormal loop termination.

While it is possible to specify a LEAVE macro which will branch to a point other than just beyond the control of the looping macro, this practice should be avoided as explained in Chapter III since it may significantly complicate the program's structure.

Figures 21 and 22 illustrate the proper and improper use of the LEAVE macro. Notice that when coded properly the DO and ENDDO LABEL parameters are identical to each other and to any inclusive LEAVE macro LABEL parameters.

The LEAVE macro also tests parameter values at assembly time, and when improperly specified, the LEAVE macro will be ignored at execution time. Like the DO macro not all parameters of the LEAVE macro can be verified at assembly time; macro specification errors may, consequently, cause system-generated errors at execution time.

```
        .
        .
        .
        DO        LABEL=LOOP, . . .
                  .
                  .
                  .
                  LEAVE     LABEL=LOOP, . . .
                  .
                  .
                  LEAVE     LABEL=LOOP, . . .
                  .
                  .
        ENDDO     LABEL=LOOP
        .
        .
        .
```

Fig. 21--Properly coded LEAVE macros

```
HERE    .
        .
        .
        DO        LABEL=LOOP, . . .
                  .
                  .
                  .
                  LEAVE     LABEL=HERE, . . .
                  .
                  .
                  LEAVE     LABEL=THERE, . . .
                  .
                  .
        ENDDO     LABEL=LOOP
        .
        .
THERE   .
```

Fig. 22--Improperly coded LEAVE macros

Appendix B contains a source listing of the LEAVE macro, an example of the expanded macro code, and samples of macro-generated error messages.

Selective Control Structure Macros

A selective control structure, the CASE macro, for ALC has been developed in this study. The CASE macro provides a multiple-selective structure for the programmer so that alternative sets of program instructions may be executed depending on the truth of the conditional parameters specified with each CASE macro.

Each CASE macro must be followed by an ENDCASE macro. All program instructions placed between the CASE and ENDCASE macros represent the code which will be executed if the CASE condition is true.

The CASE macro generates code to test the macro condition which is specified with keyword parameters similar to those which are used with the LEAVE macro. If the condition is true, instructions immediately following the CASE macro are then executed. If the condition is false, a forward branch to the paired ENDCASE macro will occur. The basic logic of a CASE macro is flowcharted in Figure 23.

The ENDCASE macro may be specified in one of two ways. The first way, OPTION=1, will cause a branch around any successive CASE macros if the preceding CASE condition were

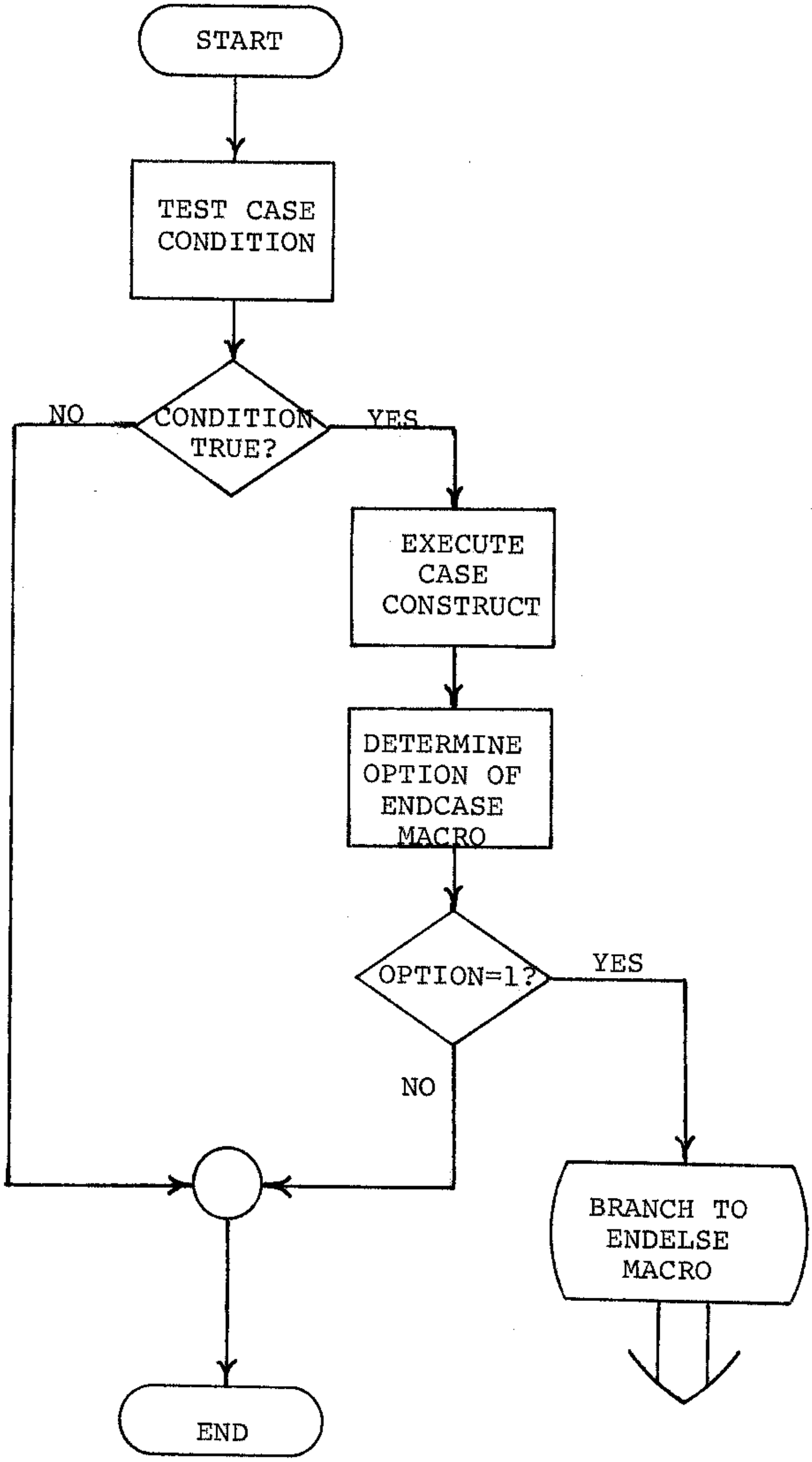

Fig. 23--Flowchart of the execution of a CASE and ENDCASE construct.

true. In other words, when several successive CASE and ENDCASE macros are coded with OPTION=1, the first encountered CASE construct whose condition is true will be executed. All subsequent CASE macros will be bypassed. If OPTION=2 is specified with successive CASE macros, each of the macro constructs with true conditions will be executed.

In order to provide an alternative set of instructions to be executed when all of the immediately preceding CASE conditions are false, the ELSE and ENDELSE macros have been developed.

The ELSE macro is merely a prototype statement to signify the beginning of the alternative code. The ENDELSE macro provides a label for branching purposes when the instructions included between the ELSE and ENDELSE macros are not to be executed. The ENDELSE prototype statement indicates the termination of both the selective CASE macros and the ELSE macro.

An example of several CASE and ENDCASE macros using OPTION=1 is shown in Figure 24. The conditional parameters for a CASE macro are identical to those of the LEAVE macro with the exception of the COND parameter. The CASE macro is always conditional so a COND parameter is not required. The ENDCASE macros contain three parameters. One of the parameters is the OPTION keyword which in all cases has been equated to one. This indicates that as soon as one of the

```
          .
          .
          .
          .
  ┌──► CASE        LABEL=ONE,LITA=7,REGB=6,OPRATOR=GT
  │                .
  │                .
  │                .
  │                .
  │                .
  └──► ENDCASE     OPTION=1,LABEL=ONE,LAB=TAG
  ┌──► CASE        LOCA=HERE,LABEL=TWO,OPRATOR=GE,LITB=4
  │                .
  │                .
  │                .
  │                .
  │                .
  └──► ENDCASE     OPTION=1,LABEL=TWO,LAB=TAG
  ┌──► CASE        LABEL=THREE,LOCB=NAME,LITA=2,OPRATOR=LE
  │                .
  │                .
  │                .
  │                .
  │                .
  └──► ENDCASE     LAB=TAG,OPTION=1,LABEL=THREE
  ┌──► CASE        LABEL=FOUR,REGA=5,REGB=8,OPRATOR=EQ
  │                .
  │                .
  │                .
  │                .
  │                .
  └──► ENDCASE     LABEL=FOUR,LAB=TAG,OPTION=1
  ┌──► ELSE
  │                .
  │                .
  │                .
  │                .
  │                .
  └──► ENDELSE     LAB=TAG
          .
          .
          .
```

Fig 24.--Example of CASE macros using OPTION=1

CASE conditions is true and its inclusive code is executed, the remaining CASE constructs along with the ELSE construct will be bypassed. In other words at most only one of the four constructs will be executed, namely the first one encountered whose condition is true. If all of the CASE conditions are untrue, the ELSE construct will be executed. The LABEL parameters associate each ENDCASE macro with its corresponding CASE macro. The LAB parameters specify the location of the ENDELSE macro associated with the CASE macros. Notice that each ENDCASE macro has its LAB parameter equated to TAG. This is necessary since a branch to the same ENDELSE macro will occur if any one of the CASE constructs is executed.

Figure 25 is a set of CASE and ENDCASE macros similar to those in Figure 24 except for the specification of the OPTION parameters. In Figure 25 each CASE construct has OPTION=2 specified. Since the second option has been specified, each CASE construct will be tested and executed if its condition is true. There is no ELSE construct included since forward branching does not occur when the second option is used.

The programmer is at liberty to use any combination of OPTION=1 and OPTION=2 CASE constructs that he wishes. Since the CASE and ENDCASE macros never produce backward branching, program linearity is not destroyed when the two options are used in successive CASE constructs. It is

```
          .
          .
          .
          .
┌──> CASE        LABEL=ONE,LITA=7,REGB=6,OPRATOR=GT
│                .
│                .
│                .
│                .
│                .
└──> ENDCASE     OPTION=2,LABEL=ONE
┌──> CASE        LOCA=HERE,LABEL=TWO,OPRATOR=GE,LITB=4
│                .
│                .
│                .
│                .
│                .
└──> ENDCASE     OPTION=2,LABEL=TWO
┌──> CASE        LABEL=THREE,LOCB=NAME,LITA=2,OPRATOR=LE
│                .
│                .
│                .
│                .
│                .
└──> ENDCASE     LABEL=THREE,OPTION=2
┌──> CASE        LABEL=FOUR,REGA=5,REGB=8,OPRATOR=EQ
│                .
│                .
│                .
│                .
│                .
└──> ENDCASE     LABEL=FOUR,OPTION=2
          .
          .
          .
          .
```

Fig. 25--Example of CASE macros using OPTION=2

important to remember that when OPTION=1 is specified, an ELSE construct must also be included. Tables III and IV summarize the various parameters which may be specified with the CASE and ENDCASE macros.

Both the CASE and ENDCASE macros contain a number of error detection instructions. For the most part, macro keyword parameters are checked to insure that they have been coded correctly. If a CASE macro has been incorrectly specified, its condition is set to false. When ENDCASE macros have been incorrectly specified, the OPTION parameter is set to one and the macro is accordingly expanded.

The source listings for the CASE, ENDCASE, ELSE, and ENDELSE macros are in Appendix C, which also contains an expanded version of each of the macros and samples of macro-generated error messages.

The CASE macro as a selective control structure was implemented because of its versatility with respect to path selection. Any number of CASE constructs may be successively defined; thus, there is no limit to the number of possible paths from which the programmer may choose. To the contrary the PL/I IF THEN ELSE statement, which is a form of the selective control structure, limits the programmer to one of two possible paths unless nesting is used. The CASE macro also allows the programmer to execute more than one of the tested paths if he so desires. This is possible when OPTION=2 is specified. The CASE macro is

TABLE III

KEYWORD PARAMETERS FOR THE CASE MACRO

Keyword Parameter	Meaning	Characteristics of Value
LABEL	Label to identify corresponding ENDCASE macro.	Any valid ALC label.
OPRATOR	Logical operator for the CASE condition	LT,LE,EQ,GE, GT,or NE.
REGA	Register containing the left operand of the CASE condition.	Integer from 0 to 15.
LOCA	Location containing the left operand of the CASE condition.	Any valid ALC label.
LOCB	Location containing the right operand of the CASE condition.	Any valid ALC label.
LITA	Numerical value of the left operand of the CASE condition.	Integer.
LITB	Numerical value of the right operand of the CASE condition.	Integer.

TABLE IV

KEYWORD PARAMETERS FOR THE ENDCASE MACRO

Keyword Parameter	Meaning	Characteristics of Value
OPTION	Numerical value specifying type of CASE construct.	Integer value of 1 or 2.
LABEL	Label linking ENDCASE macro with corresponding CASE macro	Any valid ALC label.
LAB	Label linking ENDCASE macro with corresponding ENDELSE macro.	Any valid ALC label.

especially appropriate for structured programming since it produces no backward branches. Each CASE construct is tested sequentially as it is encountered within the code producing a linear flow of control.

Appendix D contains a simple program which illustrates the combined use of the CASE and nested DO macros. A flowchart of the program is also included to illustrate the control structure of the program.

The DO, LEAVE, CASE, and ENDCASE macros contain a number of assembly-time error messages. All pertain to the specifications of macro parameters. Some of these messages indicate illegal parameter values while others identify missing parameters or duplicate parameters. It is possible to detect these error conditions at assembly-time since macro parameter values must be specified before program assembly begins. Other error conditions, relating to particular ALC instructions which are included within a macro definition, will produce system-generated messages at assembly time.

The macros described in this chapter do not contain any execution-time error tests since any error condition occurring during execution will produce a system-generated error message. At execution time each macro has already been expanded into a set of ALC instructions, and these instructions with other program instructions are subject to all of the error-detecting capabilities of the particular system being used.

CHAPTER BIBLIOGRAPHY

1. Chapin, Ned, _360/370 Programming in Assembly Language_, New York, McGraw-Hill Book Co., 1968.

2. Dijkstra, E. W., "Structured Programming," _Software Engineering_, Report on a Conference Sponsored by the NATO Science Committee, Rome, Italy, October, 1969, pp. 84-88.

3. Hannula, Reino, _Computers and Programming: A System/360-370 Assembler Language Approach_, Boston, Ma., Houghton Mifflin Co., 1974.

4. Kent, William, "Assembler-Language Macroprogramming: A Tutorial Oriented Toward the IBM 360," _Computing Surveys_, (December, 1969), 183-196.

5. _IBM System/360 Operating System Assembler Language_, IBM Form No. GC28-6514-8.

6. _IBM System/360 Operating System Principles of Operation_, IBM Form No. GA22-6821-8.

CHAPTER V

SUMMARY, CRITIQUE, AND CONCLUSIONS

Summary

Within the last seven years a growing number of programmers have expounded various concepts and methodologies which they believe will improve software design and will increase the productivity, reliability, maintainability, and extendability of programs. These various concepts and techniques have been collectively, and in part, referred to as "structured programming."

The purpose of this study was to present a survey of the major concepts, contributors and techniques of structured programming, and to develop a set of IBM 360 assembly language macros which would facilitate the application of structured programming control structures to assembly language programming.

The first significant contribution to structured programming was made by E. W. Dijkstra when he suggested (3) that the GOTO statement may prove harmful to a program's structure if the construct is used indiscriminately by the programmer. Dijkstra suggested that avoiding the use of the GOTO structure may significantly improve the logic of most programs.

Since the time of Dijkstra's paper (3) a controversy has developed concerning the use of the GOTO construct. The question today is whether or not the GOTO statement should be eliminated from programming (7). "GOTO-less" programming is sometimes used synonymously with structured programming, although the GOTO issue is by no means the only concept involved in the development of a structured programming method. Other concepts being discussed include such issues as the proper methods for subroutine entry and exit (7), and the conditions permitting abnormal program termination (interrupts).

A second significant contribution to structured programming made by Dijkstra is his development of the 'THE' Multiprogramming System (2) which illustrates a technique for system software design. This technique includes the use of what Dijkstra termed "abstract resources" in system design. These resources divided the system into layers or levels of abstraction (both control and data structures) which then were applicable to the definition of various programmable modules.

Dijkstra's work has had a profound impact on the development of structured programming, and most of the literature on structured programming, at least in part, reflects his philosophies.

Another important contributor to structured programming is Harlan Mills of the IBM Corporation. Mills (5) and his associates at IBM have developed the Chief Programmer Team method for the production of large programs. This method incorporates both a managerial and organizational approach to program design.

The Chief Programmer Team defines in a very specific manner the role of a chief programmer, a backup programmer, and a programming librarian in a programming team. The method also employs a well-defined system of program documentation during program development.

Mills (5) describes his "top-down" approach to program design. This approach includes the use of modular programming, specific programming control structures, and strict rules governing the entry and exit points of program modules. Data structures in the top-down approach are important only to the extent that they affect module interfacing.

The concepts of the Chief Programmer Team and top-down programming have contributed a possible solution to the problem of defining a structured programming method for the development of large programs.

An article by Bohm and Jacopini (1) is the basis for structured programming control structure theory. They showed that the logic of any program can be defined with the use of only three types of control structures. These structures include a sequential, a repetitive, and a selective construct. The use of only these three control structures eliminates not only the necessity of GOTO statements, but also any other control structures which may complicate program logic.

Other programming techniques which are associated with structured programming include: modular programming, open programming, compartmentalization, information hiding, extensive program documentation, the use of indentation and spacing in program code, and the use of meaningful labels and variable names in program coding.

At this time there is no generally accepted definition of structured programming. The literature suggests various approaches to improving programming techniques, and it remains to be seen just which concepts and techniques will be included in a formal definition of structured programming.

In order to apply some of the structured programming principles expounded in the literature, the author developed a set of IBM 360 assembly language macros, described in this thesis, to provide the assembly language programmer the necessary and sufficient control structures defined by Bohm and Jacopini (1).

The IBM 360 assembly language macros implemented include a DO macro for repetitive constructs and a CASE macro for selective control structures. A LEAVE macro was also developed to provide an escape mechanism for abnormal termination of a repetitive control structure. No macro was developed for sequential program control since IBM 360 assembly language instructions are executed sequentially when there are no instructions to the contrary.

These macros may be used to approximate structured programming constructs in assembly language. It should be remembered, however, that assembly language programs are inherently more tedious to code and interpret than are higher languages; consequently, structured programming techniques may not be as easily applied to assembly languages as they are to higher languages.

The presence of these macros within a program does not imply that the program is well structured. Only when the programmer takes care to apply the principles of structured programming throughout his code and uses the macros in the prescribed fashion will the control structures of his program reflect the type of structuring which is characteristic of structured programming.

In addition, a completely well-structured program would presumably observe certain principles of structuring restricting the access of program modules to data. It is not clear that the data hierarchy concepts of Dijkstra (2) will survive the test of time; in any case, the macros developed in this thesis have not considered the problems of data structures.

Critique

This investigation uncovered only one other attempt (4) to implement similar control structures using programmer-defined ALC macros. Kessler's (4) macros included a DO and CASE macro but no LEAVE statement.

The most significant difference in the author's DO macro and Kessler's is the specification of looping conditions through keyword parameters. Kessler's (4) DO macro is somewhat more flexible in that an UNTIL option is available, but it is also more restrictive since some conditional parameters must be specified in prescribed registers. The parameters Kessler (4) uses are fewer in number but substantially more complicated than those of this author.

Kessler's (4) CASE macro has a completely different implementation than the CASE macro of the author. Referring to Figure 26, an example of Kessler's CASE macro, if register R_x conatins a 5, 6, or 7, then Code A is executed. If register R_x contains a 3, then Code B is executed. If R_x does not contain a 3, 5, 6, or 7, then neither Code A or Code B will be executed. Kessler's (4) CASE macro is significantly simplier than that in this thesis, but because it lacks conditional execution parameters, the programmer will be required to include necessary conditional testing procedures in his program. For instance, using the example

```
CASENTRY  Rx
    CASE 5,6,7
         Code A
    CASE 3
         Code B
ENDCASE
```

Fig. 26--An example of Kessler's CASE macro

of Figure 26, the programmer would test a condition within his program code. Depending upon the truth of the test, he would load a certain value into register R_x which would cause the appropriate CASE to be executed. Since the programmer is required to encode the testing procedure, Kessler's (4) CASE macro is not limited to the simple testing conditions of the CASE macro implemented in this thesis. It appears then that the CASE macro of Kessler is more flexible in terms of defining test conditions, but Kessler's macro will require more programmer-generated code than the CASE macro described in this thesis.

Conclusions

Besides developing an adequate set of structured program control structures at the assembly language level, there appears to be a need to impose some new discipline of data structuring and access, especially at the assembly language level of programming. These are only weakly enforced by most input-output system calls. Multics (6) is the outstanding exception with regard to file security, but Multics per se does not enforce a modularity of user program structure.

It is hoped that after further investigation a set of principles will eventually be developed which will form the basis of a structured programming method. Not only would such a method improve the design and production of computer software and programs, but it would also initiate some semblance of uniformity in programming techniques and program structure. If all programs were written using the same basic set of programming principles, perhaps the interpretation of program logic would become standardized.

At the present time it appears that any structured programming method, if it is to be totally accepted, will have to be of a very general nature, encompassing only a few flexible principles, since the software and programs of today involve such a wide variety of programming problems and applications.

CHAPTER BIBLIOGRAPHY

1. Bohm, C. and G. Jacopini, "Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules," Communications of the ACM, IX (May, 1966), 366-371.

2. Dijkstra, E. W., "The Structure of the 'THE' Multiprogramming System," Communications of the ACM, XI (May, 1968), 341-356.

3. _______________, "GOTO Statement Considered Harmful," Communications of the ACM, XI (March, 1968), 147-148.

4. Kessler, M. M., *CONCEPT* Report 4, OS/360 Assembly Language Block Structured Programming Macros, IBM.

5. Mills, H. D., Chief Programmer Teams: Principles and Procedures, Report No. FSC 71-5108, IBM Federal Systems Division, Gaithersburg, Maryland.

6. Organick, Elliot I., The Multics System: An Examination of Its Structure, Cambridge, Ma., The MIT Press, 1972.

7. Special Interest Group on Programming Languages, Special Issue on Control Structures in Programming Languages, New York, New York, Association for Computing Machinery. Vol. VII (November, 1972).

APPENDIX A

THE DO AND ENDDO MACROS

The Source Listing for the DO Macro

```
SOURCE STATEMENT

*                     THE FOLLOWING MACRO EXPANDS TO PRODUCE AN ITERATIVE
*                     DO STATEMENT AND/OR AN ITERATIVE DO WHILE CONDITION.
*                     A DO LOOP WILL BE GENERATED WHEN THE PARAMETER '&DO-
*                     LOOP' IS NON-NULL, AND WHEN THE LOOP PARAMETERS HAVE
*                     BEEN SPECIFIED.  THE LOOP PARAMETERS INCLUDE AN IN-
*                     CREMENT, A LOWER LOOP LIMIT AND AN UPPER LOOP LIMIT.
*                     A DO WHILE CONDITION WILL BE GENERATED WHEN THE PA-
*                     RAMETER '&WHILE' IS NON-NULL, AND WHEN THE CONDITION
*                     PARAMETERS HAVE BEEN SPECIFIED.  THE WHILE CONDITION
*                     PARAMETERS INCLUDE A LEFT OPERAND, A LOGICAL OPERA-
*                     TOR, AND A RIGHT OPERAND.
*
*
           MACRO
           DO     &LABEL=,&LOWNUM=,&BYNUM=,&HGHNUM=,&LOWREG=,&DOLOOP=,      X
                  &BYREG=,&HGHREG=,&LOWLOC=,&BYLOC=,&HGHLOC=,&WHILE=,      X
                  &WOP=,&WREGA=,&WREGB=,&WLOCA=,&WLOCB=,&WLITA=,&WLITB=
           LCLA         &LAB
&LAB       SETA         &SYSNDX
           AGO          .SKIP1
*
*
*                     INITIALLY THE 'DOLOOP' PARAMETER IS TESTED TO DETER-
*                     MINE THE PRESENCE OF A DO LOOP  CONDITION.  IF IT IS
*                     NULL, A BRANCH TO THE 'WHILE' LABEL WILL TEST FOR
*                     THE PRESENCE OF A DO WHILE CONDITION.  IF THE 'DO-
*                     LOOP' PARAMETER IS NOT NULL, THEN THE CONTENTS OF
*                     REGISTERS 8 THROUGH 11 ARE IMMEDIATELY SAVED; THESE
*                     REGISTERS ARE USED AS 'WORK  REGISTERS' BY THE MACRO
*                     GENERATING THE DO LOOP CODE.
*
*
.SKIP1     AIF          ('&DOLOOP' EQ '').WHILE
           ST           8,SAVE8
           ST           9,SAVE9
           ST           10,SAVE10
           ST           11,SAVE11
           AGO          .SKIP2
*
*
*                     IF A DO LOOP HAS BEEN SPECIFIED, THE FOLLOWING SEC-
*                     TION OF CODE SELECTS THE LOWER LIMIT OF THE DO LOOP
*                     RANGE.  THIS LIMIT MAY BE SPECIFIED BY A LITERAL
*                     (LOWNUM), BY THE CONTENTS OF A REGISTER (LOWREG) OR
*                     BY THE CONTENTS OF A LOCATION (LOWLOC).  A NUMBER OF
*                     COMPARISONS ARE MADE TO VERIFY THAT THE SPECIFIED
*                     PARAMETER HAS BEEN CODED WITH AN ACCEPTABLE VALUE.
*                     IF THE PARAMETER IS VALID, THE LOWER LIMIT IS LOADED
*                     INTO REGISTER 10; OTHERWISE, AN ERROR MESSAGE IS
```

```
*                           PRINTED AND THE MACRO TERMINATES.
*
*
.SKIP2    AIF               ('&LOWNUM' EQ '').A
          AIF               ('&HGHNUM' EQ '').A1
          AIF               ('&BYNUM' EQ '').A1
          AIF               ('&LOWNUM' LT '&HGHNUM').POS
          AIF               ('&LOWNUM' EQ '&HGHNUM').ERR16
          AGO               .NEG
.POS      AIF               ('&BYNUM'(1,1) EQ '-').ERR17
          AGO               .A1
.NEG      AIF               ('&BYNUM' GT '0').ERR18
.A1       LA                10,0
          A                 10,=F'&LOWNUM'
          AGO               .C
.A        AIF               ('&LOWREG' EQ '').B
          AIF               (T'&LOWREG NE 'N').ERR24
          AIF               ('&LOWREG' GT '15').ERR19
          LR                10,&LOWREG
          AGO               .C1
.B        AIF               ('&LOWLOC' EQ '').ERR1
          L                 10,&LOWLOC
          AGO               .C2
.C        AIF               ('&LOWREG' NE '').ERR9
.C1       AIF               ('&LOWLOC' NE '').ERR9
          AGO               .C2
*
*
*                           THE FOLLOWING SECTION OF CODE SELECTS THE UPPER LIM-
*                           IT OF THE DO LOOP RANGE.  IT MAY BE SPECIFIED IN THE
*                           SAME FASHION AS THE LOWER LIMIT; THE CORRESPONDING
*                           PARAMETERS ARE HGHNUM, HGHREG AND HGHLOC.  THE CODED
*                           PARAMETER IS ALSO CHECKED FOR VALIDITY, AND THE UP-
*                           PER LIMIT IS LOADED INTO REGISTER 11.  AN APPROPRI-
*                           ATE ERROR MESSAGE WILL BE GENERATED FOR ILLEGAL PA-
*                           RAMETER VALUES AND THE MACRO WILL TERMINATE.
*
*
.C2       AIF               ('&HGHNUM' EQ '').D
          LA                11,0
          A                 11,=F'&HGHNUM'
          AGO               .F
.D        AIF               ('&HGHREG' EQ '').E
          AIF               (T'&HGHREG NE 'N').ERR25
          AIF               ('&HGHREG' GT '15').ERR20
          LR                11,&HGHREG
          AGO               .F1
.E        AIF               ('&HGHLOC' EQ '').ERR2
          L                 11,&HGHLOC
          AGO               .F2
.F        AIF               ('&HGHREG' NE '').ERR10
.F1       AIF               ('&HGHLOC' NE '').ERR10
          AGO               .F2
*
*
*                           THIS NEXT SECTION OF CODE SELECTS THE DO LOOP IN-
*                           CREMENT FOR THE SPECIFIED RANGE.  THE INCREMENT MAY
*                           BE SPECIFIED BY A LITERAL (BYNUM), BY REGISTER
*                           (BYREG) OR BY LOCATION (BYLOC).  THE INCREMENT IS
*                           LOADED INTO REGISTER 9.  ILLEGAL PARAMETER VALUES
*                           WILL TERMINATE THE MACRO.
*
*
.F2       AIF               ('&BYNUM' EQ '').G
          AIF               ('&BYNUM' EQ '0').ERR12
          LA                9,0
          A                 9,=F'&BYNUM'
```

```
         AGO     .J
.G       AIF     ('&BYREG' EQ '').H
         AIF     (T'&BYREG NE 'N').ERR26
         AIF     ('&BYREG' GT '15').ERR23
         LR      9,&BYREG
         AGO     .J1
.H       AIF     ('&BYLOC' EQ '').ERR3
         L       9,&BYLOC
         AGO     .J2
.J       AIF     ('&BYREG' NE '').ERR11
.J1      AIF     ('&BYLOC' NE '').ERR11
         AGO     .J2
*
*
*                HAVING LOADED THE INCREMENT, THE LOWER RANGE LIMIT
*                AND THE UPPER RANGE LIMIT INTO REGISTERS 9,10 AND 11
*                RESPECTIVELY, THESE REGISTERS ARE STORED IN A STACK.
*                REGISTERS 8 THROUGH 11 ARE THEN LOADED WITH THEIR
*                ORIGINAL CONTENTS.
*
*
.J2      L       8,POINTER
         ST      9,STACK(8)
         LA      8,4(8)
         ST      10,STACK(8)
         LA      8,4(8)
         ST      11,STACK(8)
         LA      8,4(8)
         ST      8,POINTER
         L       8,SAVE8
         L       9,SAVE9
         L       10,SAVE10
         L       11,SAVE11
         AGO     .SKIP3
*
*
*                THE FOLLOWING STATEMENTS REPRESENT THE BEGINNING OF
*                THE ACTUAL GENERATED DO LOOP.  WHEN THE LOOP HAS
*                BEEN EXECUTED, A BRANCH BACK TO '&LABEL' WILL INIT-
*                IATE RETESTING OF THE LOOP CONDITIONS AND POSSIBLE
*                REPEATED EXECUTION OF THE LOOP.  TO TEST THE LOOP
*                THE WORKING REGISTERS (8 THROUGH 11) ARE STORED, AND
*                THE TEST PARAMETERS ARE LOADED.
*
*
.SKIP3   ANOP
&LABEL   ST      8,SAVE8
         ST      9,SAVE9
         ST      10,SAVE10
         ST      11,SAVE11
         L       8,POINTER
         S       8,=F'4'
         L       11,STACK(8)
         S       8,=F'4'
         L       10,STACK(8)
         S       8,=F'4'
         L       9,STACK(8)
         ST      8,POINTER
         AGO     .SKIP4
*
*
*                THE CONTENTS OF REGISTER 9 ARE COMPARED AGAINST ZERO
*                TO DETERMINE WHETHER INCREMENTING IS IN THE POSITIVE
*                OR NEGATIVE DIRECTION; THIS WILL DIRECTLY AFFECT THE
*                COMPARISION OF REGISTERS 10 AND 11 WHICH ARE USED TO
*                DETERMINE WHEN THE LOOP IS SATISFIED.  REGISTER 10
```

```
181 *                       IS COMPARED TO REGISTER 11 AND IF IT IS GREATER THAN
182 *                       REGISTER 11 FOR A POSITIVE INCREMENT OR LESS THAN
183 *                       REGISTER 11 FOR A NEGATIVE INCREMENT, THE PROGRAM
184 *                       BRANCHES OUT OF THE DO LOOP.  IF THE LOOP IS NOT
185 *                       SATISFIED, CONTROL PASSES TO THE NEXT SECTION OF
186 *                       CODE.
187 *
188 *
189 .SKIP4    C             9,ZERO
190           BL            X&LAB
191           CR            10,11
192           BH            Y&LAB
193           B             Z&LAB
194 X&LAB     CR            10,11
195           BL            Y&LAB
196           AGO           .SKIP5
197 *
198 *
199 *                       WHEN THE DO LOOP IS NOT SATISFIED, THE FOLLOWING
200 *                       SECTION OF CODE ADDS THE INCREMENT TO REGISTER 10
201 *                       (WHERE THE LOWER RANGE LIMIT WAS INITIALLY LOADED)
202 *                       AND STORES THE INCREMENT AND RANGE VALUES BACK INTO
203 *                       THE STACK.
204 *
205 *
206 .SKIP5    ANOP
207 Z&LAB     AR            10,9
208           L             8,POINTER
209           ST            9,STACK(8)
210           LA            8,4(8)
211           ST            10,STACK(8)
212           LA            8,4(8)
213           ST            11,STACK(8)
214           LA            8,4(8)
215           ST            8,POINTER
216           AGO           .SKIP6
217 *
218 *
219 *                       THE ORIGINAL CONTENTS OF THE WORKING REGISTERS ARE
220 *                       RESTORED AND THE PROGRAM BRANCHES TO THE BEGINNING
221 *                       OF THE DO WHILE CODE.
222 *
223 *
224 .SKIP6    L             8,SAVE8
225           L             9,SAVE9
226           L             10,SAVE10
227           L             11,SAVE11
228           B             W&LAB
229           AGO           .SKIP7
230 *
231 *
232 *                       WHEN THE DO LOOP IS SATISFIED, A BRANCH FROM EITHER
233 *                       STATEMENT 204 OR 207 TO THE FOLLOWING CODE WILL THEN
234 *                       BE MADE.  THE WORKING REGISTERS WILL BE LOADED WITH
235 *                       THEIR ORIGINAL VALUES (THE  DO LOOP PARAMETERS WILL
236 *                       NOT BE STACKED) AND THE BRANCH TO 'A&LABEL' WILL
237 *                       TERMINATE THE LOOP.
238 *
239 *
240 .SKIP7    ANOP
241 Y&LAB     L             8,SAVE8
242           L             9,SAVE9
243           L             10,SAVE10
244           L             11,SAVE11
245           B             A&LABEL
246 W&LAB     EQU           *
```

```
             AGO      .SKIP8
*
*
*                     THE FOLLOWING CODE TESTS FOR THE PRESENCE OF A DO
*                     WHILE CONDITION.  IF A DO LOOP WAS SPECIFIED, BUT A
*                     DO WHILE CONDITION WAS NOT, A BRANCH TO THE END OF
*                     THE MACRO OCCURS.  IF A DO WHILE CONDITION HAS BEEN
*                     SPECIFIED, THEN THE MACRO PROCEEDS TO GENERATE CODE
*                     TO IMPLEMENT IT.  IF NEITHER A DO LOOP NOR DO WHILE
*                     CONDITION HAS BEEN SPECIFIED, AN ERROR MESSAGE IS
*                     PRINTED AND THE MACRO TERMINATES.
*
*
.SKIP8       AIF      ('&WHILE' EQ '').END
             AGO      .R
.WHILE       AIF      ('&WHILE' EQ '').ERR4
&LABEL       EQU      *
             AGO      .R
*
*
*                     TO IMPLEMENT THE DO WHILE CONDITION REGISTERS 10 AND
*                     11 ARE USED AS 'WORK REGISTERS';  THEREFORE, THEIR
*                     CONTENTS ARE INITIALLY SAVED.
*
*
.R           ST       10,SAVE10
             ST       11,SAVE11
             AGO      .SKIP9
*
*
*                     THE FOLLOWING INSTRUCTIONS LOAD THE LEFT OPERAND OF
*                     THE DO WHILE CONDITION INTO REGISTER 10.  THIS OPER-
*                     AND MAY BE SPECIFIED IN A REGISTER (WREGA), BY A
*                     LOCATION (WLOCA) OR BY A LITERAL (WLITA).  IF THE
*                     SPECIFIED OPERAND IS NOT A VALID VALUE (SELF-
*                     DEFINING NUMERIC) WHEN A LITERAL IS CODED, THEN AN
*                     ERROR MESSAGE WILL BE PRINTED AND THE MACRO WILL
*                     TERMINATE.
*
*
.SKIP9       AIF      ('&WREGA' EQ '').K
             AIF      (T'&WREGA NE 'N').ERR27
             AIF      ('&WREGA' GT '15').ERR21
             LR       10,&WREGA
             AGO      .M
.K           AIF      ('&WLOCA' EQ '').L
             L        10,&WLOCA
             AGO      .M1
.L           AIF      ('&WLITA' EQ '').ERR5
             AIF      ('&WLITB' NE '').ERR15
             LA       10,0
             A        10,=F'&WLITA'
             AGO      .M2
.M           AIF      ('&WLOCA' NE '').ERR13
.M1          AIF      ('&WLITA' NE '').ERR13
             AGO      .M2
*
*
*                     THE FOLLOWING STATEMENTS LOAD THE RIGHT OPERAND OF
*                     THE DO WHILE CONDITION INTO REGISTER 11 AND COMPARE
*                     IT TO REGISTER 10.  THIS OPERAND MAY ALSO BE SPECI-
*                     FIED IN A REGISTER (WREGB), BY A LOCATION (WLOCB)
*                     OR BY A LITERAL (WLITB).  IF THE SPECIFIED OPERAND
*                     IS AN INVALID VALUE WHEN A LITERAL IS CODED, AN
*                     ERROR MESSAGE WILL BE PRINTED AND THE MACRO EXPAN-
*                     SION WILL BE TERMINATED.
```

```
*
*
.M2       AIF       ('&WREGB' EQ '').N
          AIF       (T'&WREGB NE 'N').ERR28
          AIF       ('&WREGB' GT '15').ERR22
          LR        11,&WREGB
          AGO       .P
.N        AIF       ('&WLOCB' EQ '').O
          L         11,&WLOCB
          AGO       .P1
.O        AIF       ('&WLITB' EQ '').ERR6
          LA        11,0
          A         11,=F'&WLITB'
          AGO       .P2
.P        AIF       ('&WLOCB' NE '').ERR14
.P1       AIF       ('&WLITB' NE '').ERR14
.P2       CR        10,11
          AGO       .SKIP10
*
*
*                   WHEN BOTH OPERANDS OF THE DO WHILE CONDITION HAVE
*                   BEEN LOADED, THIS NEXT CODE SEGMENT SELECTS THE
*                   LOGICAL OPERATOR USED IN THE COMPARSION.  IF AN IL-
*                   LEGAL OPERATOR HAS BEEN SPECIFIED OR IF NO OPERATOR
*                   HAS BEEN SPECIFIED, THE MACRO WILL TERMINATE WITH AN
*                   ERROR MESSAGE; OTHERWISE, THE APPROPRIATE COMPARISON
*                   WILL BE MADE, AND IF THE DO WHILE CONDITION IS SAT-
*                   ISFIED, A BRANCH TO TERMINATE THE LOOP WILL OCCUR.
*
*
.SKIP10   AIF       ('&WOP' EQ '').ERR7
          AIF       ('&WOP' EQ 'EQ').EQUAL
          AIF       ('&WOP' EQ 'LT').LESS
          AIF       ('&WOP' EQ 'LE').LESSEQ
          AIF       ('&WOP' EQ 'GE').GRTEQ
          AIF       ('&WOP' EQ 'GT').GRT
          AIF       ('&WOP' EQ 'NE').NOTEQ
          AGO       .ERR8
.EQUAL    BNE       T&LAB
          AGO       .Q
.LESS     BNL       T&LAB
          AGO       .Q
.LESSEQ   BH        T&LAB
          AGO       .Q
.GRTEQ    BL        T&LAB
          AGO       .Q
.GRT      BNH       T&LAB
          AGO       .Q
.NOTEQ    BE        T&LAB
          AGO       .Q
*
*
*                   ONCE THE DO WHILE CONDITION HAS BEEN TESTED, THE
*                   FOLLOWING CODE SEGMENT RESTORES THE WORKING REGIS-
*                   TERS, 10 & 11, AND IF THE DO WHILE CONDITION IS NOT
*                   SATISFIED, BRANCHES TO THE END OF THE MACRO WHERE
*                   THE LOOP WILL BE EXECUTED AGAIN.  IF THE CONDITION
*                   HAS BEEN SATISFIED, A BRANCH TO THE 'ENDDO' MACRO
*                   WILL OCCUR AND THE LOOP WILL BE TERMINATED.
*
*
.Q        L         10,SAVE10
          L         11,SAVE11
          B         V&LAB
T&LAB     L         10,SAVE10
          L         11,SAVE11
```

```
         B        A&LABEL
V&LAB    EQU      *
         AGO      .END
*
*
*                 THIS FINAL SECTION OF MACRO CODE REPRESENTS THE ER-
*                 ROR MESSAGES GENERATED DURING MACRO EXPANSION.  ER-
*                 RORS INDICATE THAT THE MACRO CAN NOT BE EXPANDED TO
*                 PRODUCE EXECUTEABLE CODE: CONSEQUENTLY, WHEN ONE
*                 IS DETECTED, A MESSAGE IS PRINTED OUT AND A BRANCH
*                 TO THE 'ENDDO' MACRO OCCURS.  THE WORKING REGISTERS
*                 WILL BE RESTORED AND THE 'DO' MACRO WILL NOT BE FUR-
*                 THER EXPANDED.
*
*
.ERR1    MNOTE    *,'LOWER DO LOOP RANGE NOT SPECIFIED'
         B        C&LABEL
         AGO      .END
.ERR2    MNOTE    *,'UPPER DO LOOP RANGE NOT SPECIFIED'
         B        C&LABEL
         AGO      .END
.ERR3    MNOTE    *,'DO LOOP INCREMENT NOT SPECIFIED'
         B        C&LABEL
         AGO      .END
.ERR4    MNOTE    *,'NO DO OR DO WHILE CONDITION SET'
         B        A&LABEL
         AGO      .END
.ERR5    MNOTE    *,'NO LEFT OPERAND SPECIFIED ON DO WHILE CONDITION'
         B        B&LABEL
         AGO      .END
.ERR6    MNOTE    *,'NO RIGHT OPERAND SPECIFIED ON DO WHILE CONDITION'
         B        B&LABEL
         AGO      .END
.ERR7    MNOTE    *,'DO WHILE OPERATOR NOT SPECIFIED'
         B        B&LABEL
         AGO      .END
.ERR8    MNOTE    *,'INVALID DO WHILE OPERATOR SPECIFIED'
         B        B&LABEL
         AGO      .END
.ERR9    MNOTE    *,'MORE THAN ONE LOWER DO LOOP RANGE SPECIFIED'
         B        C&LABEL
         AGO      .END
.ERR10   MNOTE    *,'MORE THAN ONE UPPER DO LOOP RANGE SPECIFIED'
         B        C&LABEL
         AGO      .END
.ERR11   MNOTE    *,'MORE THAN ONE DO LOOP INCREMENT SPECIFIED'
         B        C&LABEL
         AGO      .END
.ERR12   MNOTE    *,'ZERO IS INVALID DO LOOP INCREMENT'
         B        C&LABEL
         AGO      .END
.ERR13   MNOTE    *,'MORE THAN ONE LEFT DO WHILE OPERAND SPECIFIED'
         B        B&LABEL
         AGO      .END
.ERR14   MNOTE    *,'MORE THAN ONE RIGHT DO WHILE OPERAND SPECIFIED'
         B        B&LABEL
         AGO      .END
.ERR15   MNOTE    *,'DO WHILE CONDITION CONTAINS TWO LITERAL OPERANDS'
         B        B&LABEL
         AGO      .END
.ERR16   MNOTE    *,'DO LOOP HAS SPECIFIED RANGE OF ZERO'
         AGO      .A1
.ERR17   MNOTE    *,'NEGATIVE INCREMENT INVALID FOR SPECIFIED DO LOOP'
         B        C&LABEL
         AGO      .END
.ERR18   MNOTE    *,'POSITIVE INCREMENT INVALID FOR SPECIFIED DO LOOP'
```

```
             B        C&LABEL
             AGO      .END
.ERR19       MNOTE    *,'INVALID NUMBER FOR ''LOWREG'' SPECIFIED IN DO'
             MNOTE    *,'LOOP'
             B        C&LABEL
             AGO      .END
.ERR20       MNOTE    *,'INVALID NUMBER FOR ''HGHREG'' SPECIFIED IN DO'
             MNOTE    *,'LOOP'
             B        C&LABEL
             AGO      .END
.ERR21       MNOTE    *,'INVALID NUMBER FOR ''WREGA'' SPECIFIED IN DO'
             MNOTE    *,'WHILE'
             B        B&LABEL
             AGO      .END
.ERR22       MNOTE    *,'INVALID NUMBER FOR ''WREGB'' SPECIFIED IN DO'
             MNOTE    *,'WHILE'
             B        B&LABEL
             AGO      .END
.ERR23       MNOTE    *,'INVALID NUMBER FOR ''BYREG'' SPECIFIED IN DO'
             MNOTE    *,'LOOP'
             B        C&LABEL
             AGO      .END
.ERR24       MNOTE    *,'''LOWREG'' SPECIFIED IN DO LOOP IS NOT A SELF-'
             MNOTE    *,'DEFINING NUMERIC'
             B        C&LABEL
             AGO      .END
.ERR25       MNOTE    *,'''HGHREG'' SPECIFIED IN DO LOOP IS NOT A SELF-'
             MNOTE    *,'DEFINING NUMERIC'
             B        C&LABEL
             AGO      .END
.ERR26       MNOTE    *,'''BYREG'' SPECIFIED IN DO LOOP IS NOT A SELF-'
             MNOTE    *,'DEFINING NUMERIC'
             B        C&LABEL
             AGO      .END
.ERR27       MNOTE    *,'''WREGA'' SPECIFIED IN DO WHILE IS NOT A SELF-'
             MNOTE    *,'DEFINING NUMERIC'
             B        B&LABEL
             AGO      .END
.ERR28       MNOTE    *,'''WREGB'' SPECIFIED IN DO WHILE IS NOT A SELF-'
             MNOTE    *,'DEFINING NUMERIC'
             B        B&LABEL
             AGO      .END
.END         MEND
```

The Source Listing for the ENDDO Macro

```
STMT   SOURCE STATEMENT

*                      THE ENDDO MACRO PROVIDES THE RETURN BRANCH FOR AN
*                      ITERATIVE DO LOOP OR DO WHILE CONDITION.  IT IS RE-
*                      QUIRED WHENEVER A DO MACRO IS USED, AND THE NAME AS-
*                      SIGNED TO THE 'LABEL' PARAMETER SHOULD BE THE SAME
*                      AS THAT GIVEN TO THE 'LABEL' PARAMETER IN ITS COR-
*                      RESPONDING DO MACRO.  IF A SPECIFIED DO LOOP IS NOT
*                      EXECUTED BECAUSE OF AN ERROR CONDITION, A BRANCH TO
*                      THE ENDDO MACRO WILL OCCUR.  THE WORKING REGISTERS
*                      WILL BE RESTORED AND THE MACRO TERMINATED.  WHEN A
*                      DO MACRO HAS BEEN EXECUTED PROPERLY, IT WILL RESTORE
*                      THE WORKING REGISTERS AND, UPON COMPLETION, BRANCH
*                      TO THE END OF THE ENDDO MACRO WHERE THE PROGRAM WILL
*                      CONTINUE WITH SEQUENTIAL CODE EXECUTION.
*
*
```

```
 504            MACRO
 505            ENDDO      &LABEL=
 506            B          &LABEL
 507 C&LABEL    L          8,SAVE8
 508            L          9,SAVE9
 509 B&LABEL    L          10,SAVE10
 510            L          11,SAVE11
 511 A&LABEL    EQU        *
 512            MEND
```

Sample Program A Illustrating the DO and ENDDO Macros

```
 STMT   SOURCE STATEMENT

  962            PRINT      NOGEN
  963 FRISBEE    ENTER      12,SAVEAREA
  979 START      OPEN       (DUMPAREA,OUTPUT)
  985            LA         2,0
  986            LA         3,0
  987            LA         4,0
  988            LA         5,0
  989            LA         6,0
  990            LA         7,0
  991            LA         8,0
  992            LA         9,0
  993            LA         10,0
  994            LA         11,0
  995            LA         14,0
  996            LA         15,0
  997 *
  998 *
  999            SNAP       ID=1,DCB=DUMPAREA,PDATA=REGS
 1011 *
 1012 *
 1013            DO         WHILE=R,WREGA=4,WOP=LT,WLITB=10,LABEL=ONE
 1029                       A      3,=F'1'
 1030                       A      4,=F'1'
 1031                       A      5,=F'1'
 1032                       A      6,=F'1'
 1033                       A      7,=F'1'
 1034                       A      8,=F'1'
 1035                       A      9,=F'1'
 1036                       A      10,=F'1'
 1037            ENDDO      LABEL=ONE
 1044 *
 1045 *
 1046            SNAP       ID=2,DCB=DUMPAREA,PDATA=REGS
 1058            LA         3,0
 1059            LA         4,0
 1060            LA         5,0
 1061            LA         6,0
 1062            LA         7,0
 1063            LA         8,0
 1064            LA         9,0
 1065            LA         10,0
 1066 *
 1067 *
 1068            DO         LOWNUM=1,HGHNUM=30,BYNUM=4,DOLOOP=S,LABEL=TWO
 1130                       A      3,=F'2'
 1131                       A      4,=F'2'
 1132                       A      5,=F'2'
 1133                       A      6,=F'2'
```

```
1134                   A     7,=F'2'
1135                   A     8,=F'2'
1136                   A     9,=F'2'
1137                   A     10,=F'2'
1138          ENDDO    LABEL=TWO
1145 *
1146 *
1147          SNAP     ID=3,DCB=DUMPAREA,PDATA=REGS
1159          LA       3,0
1160          LA       4,0
1161          LA       5,0
1162          LA       6,0
1163          LA       7,0
1164          LA       8,0
1165          LA       9,0
1166          LA       10,0
1167 *
1168 *
1169          DO       LOWNUM=3,BYLOC=Y,HGHREG=4,DOLOOP=7,WHILE=NN,        X
                     LABEL=THREE,WOP=GT,WLITA=7,WREGB=15
1243                   A     3,=F'3'
1244                   A     4,=F'3'
1245                   A     5,=F'3'
1246                   A     6,=F'3'
1247                   A     7,=F'3'
1248                   A     8,=F'3'
1249                   A     9,=F'3'
1250                   A     10,=F'3'
1251          ENDDO    LABEL=THREE
1258 *
1259 *
1260          SNAP     ID=4,DCB=DUMPAREA,PDATA=REGS
1272 *
1273 *
1274          EXIT
1279          CLOSE    DUMPAREA

1285 SAVEAREA DC       18A(0)
1286          DS       0F
1287 ZERO     DC       1F'0'
1288 SAVE8    DS       1F
1289 SAVE9    DS       1F
1290 SAVE10   DS       1F
1291 SAVE11   DS       1F
1292 POINTER  DC       1F'0'
1293 STACK    DS       100F
1294 Y        DC       1F'1'
1295 DUMPAREA DCB      DDNAME=TEAM,DSORG=PS,RECFM=VBA,                     X
                     MACRF=W,BLKSIZE=882,LRECL=125
1346          END
1347                 =F'10'
1348                 =F'1'
1349                 =F'30'
1350                 =F'4'
1351                 =F'2'
1352                 =F'3'
1353                 =F'7'


REGS AT ENTRY TO SNAP                                  ID = 001


REGS 0-7             00000030  9001B388  00000000  00000000
                     00000000  00000000  00000000  00000000
```

REGS 8-15	00000000	00000000	00000000	00000000
	50018820	00018C00	00000000	00000000

END OF SNAP

REGS AT ENTRY TO SNAP ID = 002

REGS 0-7	000002A0	8001B904	00000000	0000000A
	0000000A	0000000A	0000000A	0000000A
REGS 8-15	0000000A	0000000A	0000000A	00000000
	5001B820	00018C00	00000000	00000000

END OF SNAP

REGS AT ENTRY TO SNAP ID = 003

REGS 0-7	000002A0	A001BA58	00000000	00000010
	00000010	00000010	00000010	00000010
REGS 8-15	00000010	00000010	00000010	00000000
	5001B820	0001BC00	00000000	00000000

END OF SNAP

REGS AT ENTRY TO SNAP ID = 004

REGS 0-7	000002A0	A001B804	00000000	00000000
	00000000	00000000	00000000	00000000
REGS 8-15	00000000	00000000	00000000	00000000
	5001B820	0001BC00	00000000	00000000

END OF SNAP

Program A with Expanded Macros

```
 STMT   SOURCE STATEMENT

  962 FRISBEE  ENTER     12,SAVEAREA
  963+FRISBEE  DS    0H
  964+         ENTRY FRISBEE DECLARE NAME ENTRY
  965+         USING *,12 DECLARE BASE ADDRESSIBILITY.
  966+         BALR  15,0 (INITIAL ADDRESSIBILITY).
  967+         B     12(,15) BRANCH AROUND ID FIELD
  968+         DC    AL1(7),CL7'FRISBEE' ID LENGTH AND ID
  969+         BCTR  15,0 (RESET INITIAL ADDRESSIBILITY
  970+         BCTR  15,0 ABSOLUTE ENTRY POINT).
  971+         STM   14,12,12(13) SAVE REGISTERS
  972+         LR    12,15 SETUP BASE REGISTER.
  973+         ST    13,SAVEAREA+4 CHAIN BACK
  974+         LA    0,SAVEAREA CHAIN FORWARD
  975+         ST    0,8(0,13)
  976+         LR    13,0 SET UP SAVE AREA POINTER
  977+         USING SAVEAREA,13 AND ADDRESSABILITY
  978 START    OPEN      (DUMPAREA,OUTPUT)
  979+         CNOP  0,4 ALIGN LIST TO FULLWORD
  980+START    BAL   1,*+8 LOAD REG1 W/LIST ADDR.
  981+         DC    AL1(143) OPTION BYTE
  982+         DC    AL3(DUMPAREA) DCB ADDRESS
  983+         SVC   19 ISSUE OPEN SVC
  984          LA        2,0
  985          LA        3,0
  986          LA        4,0
  987          LA        5,0
  988          LA        6,0
  989          LA        7,0
  990          LA        8,0
  991          LA        9,0
  992          LA        10,0
  993          LA        11,0
  994          LA        14,0
  995          LA        15,0
  996 *
  997 *
  998          SNAP      ID=1,DCB=DUMPAREA,PDATA=REGS
  999+         CNOP  0,4
 1000+         BAL   1,IHB0003 BRANCH AROUND PARAM LIST
 1001+         DC    AL1(1) ID NUMBER
 1002+         DC    AL1(0)
 1003+         DC    AL1(130) OPTION FLAGS
 1004+         DC    AL1(32) OPTION FLAGS
 1005+         DC    A(DUMPAREA) DCB ADDRESS
 1006+         DC    A(0) TCB ADDRESS
 1007+         DC    A(0) ADDRESS OF SNAP-SHOT LIST
 1008+IHB0003  DS    0H
 1009+         SVC   51
 1010 *
 1011 *
 1012          DO        WHILE=R,WREGA=4,WOP=LT,WLITB=10,LABEL=ONE
 1013+ONE      EQU   *
 1014+         ST    10,SAVE10
 1015+         ST    11,SAVE11
 1016+         LR    10,4
 1017+         LA    11,0
 1018+         A     11,=F'10'
 1019+         CR    10,11
 1020+         BNL   T4
 1021+         L     10,SAVE10
 1022+         L     11,SAVE11
```

```
+         B      V4
+T4       L      10,SAVE10
+         L      11,SAVE11
+         B      AONE
+V4       EQU    *
                        A      3,=F'1'
                        A      4,=F'1'
                        A      5,=F'1'
                        A      6,=F'1'
                        A      7,=F'1'
                        A      8,=F'1'
                        A      9,=F'1'
                        A      10,=F'1'
          ENDDO     LABEL=ONE
+         B      ONE
+CONE     L      8,SAVE8
+         L      9,SAVE9
+BONE     L      10,SAVE10
+         L      11,SAVE11
+AONE     EQU    *
 *
 *
          SNAP      ID=2,DCB=DUMPAREA,PDATA=REGS
+         CNOP   0,4
+         BAL    1,IHB0006 BRANCH AROUND PARAM LIST
+         DC     AL1(2) ID NUMBER
+         DC     AL1(0)
+         DC     AL1(130) OPTION FLAGS
+         DC     AL1(32) OPTION FLAGS
+         DC     A(DUMPAREA) DCB ADDRESS
+         DC     A(0) TCB ADDRESS
+         DC     A(0) ADDRESS OF SNAP-SHOT LIST
+IHB0006  DS     0H
+         SVC    51
          LA        3,0
          LA        4,0
          LA        5,0
          LA        6,0
          LA        7,0
          LA        8,0
          LA        9,0
          LA        10,0
 *
 *
          DO        LOWNUM=1,HGHNUM=30,BYNUM=4,DOLOOP=S,LABEL=TWO
+         ST     8,SAVE8
+         ST     9,SAVE9
+         ST     10,SAVE10
+         ST     11,SAVE11
+         LA     10,0
+         A      10,=F'1'
+         LA     11,0
+         A      11,=F'30'
+         LA     9,0
+         A      9,=F'4'
+         L      8,POINTER
+         ST     9,STACK(8)
+         LA     8,4(8)
+         ST     10,STACK(8)
+         LA     8,4(8)
+         ST     11,STACK(8)
+         LA     8,4(8)
+         ST     8,POINTER
+         L      8,SAVE8
+         L      9,SAVE9
+         L      10,SAVE10
```

```
+          L        11,SAVE11
+TWO       ST       8,SAVE8
+          ST       9,SAVE9
+          ST       10,SAVE10
+          ST       11,SAVE11
+          L        8,POINTER
+          S        8,=F'4'
+          L        11,STACK(8)
+          S        8,=F'4'
+          L        10,STACK(8)
+          S        8,=F'4'
+          L        9,STACK(8)
+          ST       8,POINTER
+          C        9,ZERO
+          BL       X7
+          CR       10,11
+          BH       Y7
+          B        Z7
+X7        CR       10,11
+          BL       Y7
+Z7        AR       10,9
+          L        8,POINTER
+          ST       9,STACK(8)
+          LA       8,4(8)
+          ST       10,STACK(8)
+          LA       8,4(8)
+          ST       11,STACK(8)
+          LA       8,4(8)
+          ST       8,POINTER
+          L        8,SAVE8
+          L        9,SAVE9
+          L        10,SAVE10
+          L        11,SAVE11
+          B        W7
+Y7        L        8,SAVE8
+          L        9,SAVE9
+          L        10,SAVE10
+          L        11,SAVE11
+          B        ATWO
+W7        EQU      *
                        A       3,=F'2'
                        A       4,=F'2'
                        A       5,=F'2'
                        A       6,=F'2'
                        A       7,=F'2'
                        A       8,=F'2'
                        A       9,=F'2'
                        A       10,=F'2'
           ENDDO        LABEL=TWO
+          B        TWO
+CTWO      L        8,SAVE8
+          L        9,SAVE9
+BTWO      L        10,SAVE10
+          L        11,SAVE11
+ATWO      EQU      *
 *
 *
           SNAP         ID=3,DCB=DUMPAREA,PDATA=REGS
+          CNOP     0,4
+          BAL      1,IHB0009 BRANCH AROUND PARAM LIST
+          DC       AL1(3) ID NUMBER
+          DC       AL1(0)
+          DC       AL1(130) OPTION FLAGS
+          DC       AL1(32) OPTION FLAGS
+          DC       A(DUMPAREA) DCB ADDRESS
+          DC       A(0) TCB ADDRESS
```

```
+           DC     A(0) ADDRESS OF SNAP-SHOT LIST
+IHB0009    DS     0H
+           SVC    51
            LA        3,0
            LA        4,0
            LA        5,0
            LA        6,0
            LA        7,0
            LA        8,0
            LA        9,0
            LA        10,0
 *
 *
            DO          LOWNUM=3,BYLOC=Y,HGHREG=4,DOLOOP=7,WHILE=NN,          X
                   LABEL=THREE,WOP=GT,WLITA=7,WREGB=15
+           ST     8,SAVE8
+           ST     9,SAVE9
+           ST     10,SAVE10
+           ST     11,SAVE11
+           LA     10,0
+           A      10,=F'3'
+           LR     11,4
+           L      9,Y
+           L      8,POINTER
+           ST     9,STACK(8)
+           LA     8,4(8)
+           ST     10,STACK(8)
+           LA     8,4(8)
+           ST     11,STACK(8)
+           LA     8,4(8)
+           ST     8,POINTER
+           L      8,SAVE8
+           L      9,SAVE9
+           L      10,SAVE10
+           L      11,SAVE11
+THREE      ST     8,SAVE8
+           ST     9,SAVE9
+           ST     10,SAVE10
+           ST     11,SAVE11
+           L      8,POINTER
+           S      8,=F'4'
+           L      11,STACK(8)
+           S      8,=F'4'
+           L      10,STACK(8)
+           S      8,=F'4'
+           L      9,STACK(8)
+           ST     8,POINTER
+           C      9,ZERO
+           BL     X10
+           CR     10,11
+           BH     Y10
+           B      Z10
+X10        CR     10,11
+           BL     Y10
+Z10        AR     10,9
+           L      8,POINTER
+           ST     9,STACK(8)
+           LA     8,4(8)
+           ST     10,STACK(8)
+           LA     8,4(8)
+           ST     11,STACK(8)
+           LA     8,4(8)
+           ST     8,POINTER
+           L      8,SAVE8
+           L      9,SAVE9
+           L      10,SAVE10
```

```
+               L      11,SAVE11
+               B      W10
+Y10            L      8,SAVE8
+               L      9,SAVE9
+               L      10,SAVE10
+               L      11,SAVE11
+               B      ATHREE
+W10            EQU    *
+               ST     10,SAVE10
+               ST     11,SAVE11
+               LA     10,0
+               A      10,=F'7'
+               LR     11,15
+               CR     10,11
+               BNH    T10
+               L      10,SAVE10
+               L      11,SAVE11
+               B      V10
+T10            L      10,SAVE10
+               L      11,SAVE11
+               B      ATHREE
+V10            EQU    *
                              A      3,=F'3'
                              A      4,=F'3'
                              A      5,=F'3'
                              A      6,=F'3'
                              A      7,=F'3'
                              A      8,=F'3'
                              A      9,=F'3'
                              A      10,=F'3'
                EVDDO       LABEL=THREE
+               B      THREE
+CTHREE         L      8,SAVE8
+               L      9,SAVE9
+BTHREE         L      10,SAVE10
+               L      11,SAVE11
+ATHREE         EQU    *
 *
 *
                SVAP        ID=4,DCB=DUMPAREA,PDATA=REGS
+               CVOP   0,4
+               BAL    1,IHB0012 BRANCH AROUND PARAM LIST
+               DC     AL1(4) ID NUMBER
+               DC     AL1(0)
+               DC     AL1(130) OPTION FLAGS
+               DC     AL1(32) OPTION FLAGS
+               DC     A(DUMPAREA) DCB ADDRESS
+               DC     A(0) TCB ADDRESS
+               DC     A(0) ADDRESS OF SNAP-SHOT LIST
+IHB0012        DS     0H
+               SVC    51
 *
 *
                EXIT
+               L      13,4(,13) POP UP SAVE AREA
+               LM     14,12,12(13) RESTORE REGISTERS
+               MVI    12(13),X'FF' FLAG EXIT
+               BR     14 RETURN
                CLOSE     DUMPAREA
+               CVOP   0,4 ALIGN LIST TO FULLWORD
+               BAL    1,*+8 LOAD PEG1 W/LIST ADDR
+               DC     AL1(128) OPTION BYTE
+               DC     AL3(DUMPAREA) DCB ADDRESS
+               SVC    20 ISSUE CLOSE SVC

 SAVEAREA DC          18A(0)
```

```
1285               DS        0F
1286 ZERO          DC        1F'0'
1287 SAVE8         DS        1F
1288 SAVE9         DS        1F
1289 SAVE10        DS        1F
1290 SAVE11        DS        1F
1291 POINTER       DC        1F'0'
1292 STACK         DS        100F
1293 Y             DC        1F'1'
1294 DUMPAREA DCB            DDNAME=TEAM,DSORG=PS,RECFM=VBA,                 X
                          MACRF=W,BLKSIZE=882,LRECL=125

1296+*                            DATA CONTROL BLOCK
1297+*
1298+DUMPAREA DC          0F'0' ORIGIN ON WORD BOUNDARY

1300+*                            DIRECT ACCESS DEVICE INTERFACE

1302+              DC     BL16'0' FDAD,DVTBL
1303+              DC     A(0) KEYLE,DEVT,TRBAL

1305+*                            COMMON ACCESS METHOD INTERFACE

1307+              DC     AL1(0) BUFNO
1308+              DC     AL3(1) BUFCB
1309+              DC     AL2(0) BUFL
1310+              DC     BL2'0100000000000000' DSORG
1311+              DC     A(1) IOBAD

1313+*                            FOUNDATION EXTENSION

1315+              DC     BL1'00000000' BFTEK,BFLN,HIARCHY
1316+              DC     AL3(1) EODAD
1317+              DC     BL1'01010100' RECFM
1318+              DC     AL3(0) EXLST

1320+*                            FOUNDATION BLOCK

1322+              DC     CL8'TEAM' DDNAME
1323+              DC     BL1'00000010' OFLGS
1324+              DC     BL1'00000000' IFLG
1325+              DC     BL2'0000000000100000' MACR

1327+*                            BSAM-BPAM-QSAM INTERFACE

1329+              DC     BL1'00000000' RER1
1330+              DC     AL3(1) CHECK, GERR, PERR
1331+              DC     A(1) SYNAD
1332+              DC     H'0' CIND1, CIND2
1333+              DC     AL2(882) BLKSIZE
1334+              DC     F'0' WCPO, WCPL, OFFSR, OFFSW
1335+              DC     A(1) IOBA
1336+              DC     AL1(0) NCP
1337+              DC     AL3(1) EOBR, EOBAD

1339+*                               BSAM-BPAM INTERFACE

1341+              DC     A(1) EOBW
1342+              DC     H'0' DIRCT
1343+              DC     AL2(125) LRECL
1344+              DC     A(1) CNTRL, NOTE, POINT
1345               END
1346                      =F'10'
1347                      =F'1'
1348                      =F'30'
```

```
1349              =F'4'
1350              =F'2'
1351              =F'3'
1352              =F'7'
```

Sample Program B Illustrating a DO Macro Coded Incorrectly

```
STMT   SOURCE STATEMENT

 962 FRISBEE   ENTER     12,SAVEAREA
 963+FRISBEE   DS    0H
 964+          ENTRY FRISBEE DECLARE NAME ENTRY
 965+          USING *,12 DECLARE BASE ADDRESSIBILITY.
 966+          BALR  15,0 (INITIAL ADDRESSIBILITY).
 967+          B     12(,15) BRANCH AROUND ID FIELD
 968+          DC    AL1(7),CL7'FRISBEE' ID LENGTH AND ID
 969+          BCTR  15,0 (RESET INITIAL ADDRESSIBILITY
 970+          BCTR  15,0 ABSOLUTE ENTRY POINT).
 971+          STM   14,12,12(13) SAVE REGISTERS
 972+          LR    12,15 SETUP BASE REGISTER.
 973+          ST    13,SAVEAREA+4 CHAIN BACK
 974+          LA    0,SAVEAREA CHAIN FORWARD
 975+          ST    0,8(0,13)
 976+          LR    13,0 SET UP SAVE AREA POINTER
 977+          USING SAVEAREA,13 AND ADDRESSABILITY
 978 START     OPEN      (DUMPAREA,OUTPUT)
 979+          CNOP  0,4 ALIGN LIST TO FULLWORD
 980+START     BAL   1,*+8 LOAD REG1 W/LIST ADDR.
 981+          DC    AL1(143) OPTION BYTE
 982+          DC    AL3(DUMPAREA) DCB ADDRESS
 983+          SVC   19 ISSUE OPEN SVC
 984           LA        2,0
 985           LA        3,0
 986           LA        4,0
 987           LA        5,0
 988           LA        6,0
 989           LA        7,0
 990           LA        8,0
 991           LA        9,0
 992           LA        10,0
 993           LA        11,0
 994           LA        14,0
 995           LA        15,0
 996 *
 997 *
 998           SNAP      ID=1,DCB=DUMPAREA,PDATA=REGS
 999+          CNOP  0,4
1000+          BAL   1,IHB0003 BRANCH AROUND PARAM LIST
1001+          DC    AL1(1) ID NUMBER
1002+          DC    AL1(0)
1003+          DC    AL1(130) OPTION FLAGS
1004+          DC    AL1(32) OPTION FLAGS
1005+          DC    A(DUMPAREA) DCB ADDRESS
1006+          DC    A(0) TCB ADDRESS
1007+          DC    A(0) ADDRESS OF SNAP-SHOT LIST
1008+IHB0003   DS    0H
1009+          SVC   51
1010 *
1011 *
1012           DO        WHILE=R,WREGA=4,WOP=LT,LABEL=ONE
1013+ONE       EQU   *
```

```
1014+           ST     10,SAVE10
1015+           ST     11,SAVE11
1016+           LR     10,4
1017                   *,NO RIGHT OPERAND SPECIFIED ON DO WHILE CONDITION
1018+           B      BONE
1019                          A     3,=F'1'
1020                          A     4,=F'1'
1021                          A     5,=F'1'
1022                          A     6,=F'1'
1023                          A     7,=F'1'
1024                          A     8,=F'1'
1025                          A     9,=F'1'
1026                          A     10,=F'1'
1027            ENDDO         LABEL=ONE
1028+           B      ONE
1029+CONE       L      8,SAVE8
1030+           L      9,SAVE9
1031+BONE       L      10,SAVE10
1032+           L      11,SAVE11
1033+AONE       EQU    *
1034 *
1035 *
1036                 SNAP       ID=2,DCB=DUMPAREA,PDATA=REGS
1037+           CNOP   0,4
1038+           BAL    1,IHB0006 BRANCH AROUND PARAM LIST
1039+           DC     AL1(2) ID NUMBER
1040+           DC     AL1(0)
1041+           DC     AL1(130) OPTION FLAGS
1042+           DC     AL1(32) OPTION FLAGS
1043+           DC     A(DUMPAREA) DCB ADDRESS
1044+           DC     A(0) TCB ADDRESS
1045+           DC     A(0) ADDRESS OF SNAP-SHOT LIST
1046+IHB0006    DS     0H
1047+           SVC    51
1048 *
1049 *
1050            EXIT
1051+           L      13,4(,13) POP UP SAVE AREA
1052+           LM     14,12,12(13) RESTORE REGISTERS
1053+           MVI    12(13),X'FF' FLAG EXIT
1054+           BR     14 RETURN
1055            CLOSE      DUMPAREA
1056+           CNOP   0,4 ALIGN LIST TO FULLWORD
1057+           BAL    1,*+8 LOAD REG1 W/LIST ADDR
1058+           DC     AL1(128) OPTION BYTE
1059+           DC     AL3(DUMPAREA) DCB ADDRESS
1060+           SVC    20 ISSUE CLOSE SVC

1061 SAVEAREA DC          18A(0)
1062          DS          0F
1063 ZERO     DC          1F'0'
1064 SAVE8    DS          1F
1065 SAVE9    DS          1F
1066 SAVE10   DS          1F
1067 SAVE11   DS          1F
1068 POINTER  DC          1F'0'
1069 STACK    DS          100F
1070 DUMPAREA DCB         DDNAME=TEAM,DSORG=PS,RECFM=VBA,                     X
                       MACRF=W,BLKSIZE=882,LRECL=125


1072+*                           DATA CONTROL BLOCK
1073+*
1074+DUMPAREA DC       0F'0' ORIGIN ON WORD BOUNDARY

1076+*                           DIRECT ACCESS DEVICE INTERFACE
```

```
1078+          DC     BL16'0' FDAD,DVTBL
1079+          DC     A(0) KEYLE,DEVT,TRBAL

1081+*                          COMMON ACCESS METHOD INTERFACE

1083+          DC     AL1(0) BUFNO
1084+          DC     AL3(1) BUFCB
1085+          DC     AL2(0) BUFL
1086+          DC     BL2'0100000000000000' DSORG
1087+          DC     A(1) IOBAD

1089+*                          FOUNDATION EXTENSION

1091+          DC     BL1'00000000' BFTEK,BFLN,HIARCHY
1092+          DC     AL3(1) EODAD
1093+          DC     BL1'01010100' RECFM
1094+          DC     AL3(0) EXLST

1096+*                          FOUNDATION BLOCK

1098+          DC     CL8'TEAM' DDNAME
1099+          DC     BL1'00000010' OFLGS
1100+          DC     BL1'00000000' IFLG
1101+          DC     BL2'0000000000100000' MACR

1103+*                          BSAM-BPAM-QSAM INTERFACE

1105+          DC     BL1'00000000' RER1
1106+          DC     AL3(1) CHECK, GERR, PERR
1107+          DC     A(1) SYNAD
1108+          DC     H'0' CIND1, CIND2
1109+          DC     AL2(882) BLKSIZE
1110+          DC     F'0' WCPO, WCPL, OFFSR, OFFSW
1111+          DC     A(1) IOBA
1112+          DC     AL1(0) NCP
1113+          DC     AL3(1) EOBR, EOBAD

1115+*                          BSAM-BPAM INTERFACE

1117+          DC     A(1) EOBW
1118+          DC     H'0' DIRCT
1119+          DC     AL2(125) LRECL
1120+          DC     A(1) CNTRL, NOTE, POINT
1121           END
1122                  =F'1'
```

Sample Program C Illustrating a DO Macro Coded Incorrectly

```
STMT   SOURCE STATEMENT

 962 FRISBEE  ENTER     12,SAVEAREA
 963+FRISBEE  DS     0H
 964+         ENTRY  FRISBEE DECLARE NAME ENTRY
 965+         USING  *,12 DECLARE BASE ADDRESSIBILITY.
 966+         BALR   15,0 (INITIAL ADDRESSIBILITY).
 967+         B      12(,15) BRANCH AROUND ID FIELD
 968+         DC     AL1(7),CL7'FRISBEE' ID LENGTH AND ID
 969+         BCTR   15,0 (RESET INITIAL ADDRESSIBILITY
 970+         BCTR   15,0 ABSOLUTE ENTRY POINT).
 971+         STM    14,12,12(13) SAVE REGISTERS
 972+         LR     12,15 SETUP BASE REGISTER.
```

```
+               ST      13,SAVEAREA+4 CHAIN BACK
+               LA      0,SAVEAREA CHAIN FORWARD
+               ST      0,8(0,13)
+               LR      13,0 SET UP SAVE AREA POINTER
+               USING SAVEAREA,13 AND ADDRESSABILITY
 START          OPEN         (DUMPAREA,OUTPUT)
+               CNOP    0,4 ALIGN LIST TO FULLWORD
+START          BAL     1,*+8 LOAD REG1 W/LIST ADDR.
+               DC      AL1(143) OPTION BYTE
+               DC      AL3(DUMPAREA) DCB ADDRESS
+               SVC     19 ISSUE OPEN SVC
                LA          2,0
                LA          3,0
                LA          4,0
                LA          5,0
                LA          6,0
                LA          7,0
                LA          8,0
                LA          9,0
                LA          10,0
                LA          11,0
                LA          14,0
                LA          15,0
 *
 *
                    SNAP        ID=1,DCB=DUMPAREA,PDATA=REGS
+               CNOP    0,4
+               BAL     1,IHB0003 BRANCH AROUND PARAM LIST
+               DC      AL1(1) ID NUMBER
+               DC      AL1(0)
+               DC      AL1(130) OPTION FLAGS
+               DC      AL1(32) OPTION FLAGS
+               DC      A(DUMPAREA) DCB ADDRESS
+               DC      A(0) TCB ADDRESS
+               DC      A(0) ADDRESS OF SNAP-SHOT LIST
+IHB0003        DS      0H
+               SVC     51
 *
 *
                DO          LOWNUM=1,LOWREG=5,HGHNUM=30,BYNUM=1,DOLOOP=W,        X
                        LABEL=TWO
+               ST      8,SAVE8
+               ST      9,SAVE9
+               ST      10,SAVE10
+               ST      11,SAVE11
+               LA      10,0
+               A       10,=F'1'
                        *,MORE THAN ONE LOWER DO LOOP RANGE SPECIFIED
+               B       CTWO
                            A      3,=F'2'
                            A      4,=F'2'
                            A      5,=F'2'
                            A      6,=F'2'
                            A      7,=F'2'
                            A      8,=F'2'
                            A      9,=F'2'
                            A      10,=F'2'
                ENDDO       LABEL=TWO
+               B       TWO

+CTWO           L       8,SAVE8
+               L       9,SAVE9
+BTWO           L       10,SAVE10
+               L       11,SAVE11
+ATWO           EQU     *
 *
```

```
1037 *
1038                  SNAP      ID=2,DCB=DUMPAREA,PDATA=REGS
1039+          CNOP  0,4
1040+          BAL   1,IHB0006 BRANCH AROUND PARAM LIST
1041+          DC    AL1(2) ID NUMBER
1042+          DC    AL1(0)
1043+          DC    AL1(130) OPTION FLAGS
1044+          DC    AL1(32) OPTION FLAGS
1045+          DC    A(DUMPAREA) DCB ADDRESS
1046+          DC    A(0) TCB ADDRESS
1047+          DC    A(0) ADDRESS OF SNAP-SHOT LIST
1048+IHB0006   DS    0H
1049+          SVC   51
1050 *
1051 *
1052           EXIT
1053+          L     13,4(,13) POP UP SAVE AREA
1054+          LM    14,12,12(13) RESTORE REGISTERS
1055+          MVI   12(13),X'FF' FLAG EXIT
1056+          BR    14 RETURN
1057           CLOSE     DUMPAREA
1058+          CNOP  0,4 ALIGN LIST TO FULLWORD
1059+          BAL   1,*+8 LOAD REG1 W/LIST ADDR
1060+          DC    AL1(128) OPTION BYTE
1061+          DC    AL3(DUMPAREA) DCB ADDRESS
1062+          SVC   20 ISSUE CLOSE SVC

1063 SAVEAREA DC         18A(0)
1064          DS         0F
1065 ZERO     DC         1F'0'
1066 SAVE8    DS         1F
1067 SAVE9    DS         1F
1068 SAVE10   DS         1F
1069 SAVE11   DS         1F
1070 POINTER  DC         1F'0'
1071 STACK    DS         100F
1072 DUMPAREA DCB         DDNAME=TEAM,DSORG=PS,RECFM=VBA,                X
                    MACRF=W,BLKSIZE=882,LRECL=125


1074+*                           DATA CONTROL BLOCK
1075+*
1076+DUMPAREA DC      0F'0' ORIGIN ON WORD BOUNDARY

1078+*                           DIRECT ACCESS DEVICE INTERFACE

1080+          DC     BL16'0' FDAD,DVTBL
1081+          DC     A(0) KEYLE,DEVT,TRBAL

1083+*                           COMMON ACCESS METHOD INTERFACE

1085+          DC     AL1(0) BUFNO
1086+          DC     AL3(1) BUFCB
1087+          DC     AL2(0) BUFL
1088+          DC     BL2'0100000000000000' DSORG
1089+          DC     A(1) IOBAD

1091+*                           FOUNDATION EXTENSION

1093+          DC     BL1'00000000' BFTEK,BFLN,HIARCHY
1094+          DC     AL3(1) EODAD
1095+          DC     BL1'01010100' RECFM
1096+          DC     AL3(0) EXLST

1098+*                           FOUNDATION BLOCK
```

```
1100+          DC       CL8'TEAM' DDNAME
1101+          DC       BL1'00000010' OFLGS
1102+          DC       BL1'00000000' IFLG
1103+          DC       BL2'0000000000100000' MACR

1105+*                           BSAM-BPAM-QSAM INTERFACE

1107+          DC       BL1'00000000' RER1
1108+          DC       AL3(1) CHECK, GERR, PERR
1109+          DC       A(1) SYNAD
1110+          DC       H'0' CIND1, CIND2
1111+          DC       AL2(882) BLKSIZE
1112+          DC       F'0' WCPO, WCPL, OFFSR, OFFSW
1113+          DC       A(1) IOBA
1114+          DC       AL1(0) NCP
1115+          DC       AL3(1) EOBR, EOBAD

1117+*                             BSAM-BPAM INTERFACE

1119+          DC       A(1) EOBW
1120+          DC       H'0' DIRCT
1121+          DC       AL2(125) LRECL
1122+          DC       A(1) CNTRL, NOTE, POINT
1123           END
1124                    =F'1'
1125                    =F'2'
```

Sample Program D Illustrating a DO Macro Coded Incorrectly

```
STMT   SOURCE STATEMENT

 962 FRISBEE  ENTER      12,SAVEAREA
 963+FRISBEE  DS     0H
 964+         ENTRY  FRISBEE DECLARE NAME ENTRY
 965+         USING  *,12 DECLARE BASE ADDRESSIBILITY.
 966+         BALR   15,0 (INITIAL ADDRESSIBILITY).
 967+         B      12(,15) BRANCH AROUND ID FIELD
 968+         DC     AL1(7),CL7'FRISBEE' ID LENGTH AND ID
 969+         BCTR   15,0 (RESET INITIAL ADDRESSIBILITY
 970+         BCTR   15,0 ABSOLUTE ENTRY POINT).
 971+         STM    14,12,12(13) SAVE REGISTERS
 972+         LR     12,15 SETUP BASE REGISTER.
 973+         ST     13,SAVEAREA+4 CHAIN BACK
 974+         LA     0,SAVEAREA CHAIN FORWARD
 975+         ST     0,8(0,13)
 976+         LR     13,0 SET UP SAVE AREA POINTER
 977+         USING  SAVEAREA,13 AND ADDRESSABILITY
 978 START    OPEN       (DUMPAREA,OUTPUT)
 979+         CNOP   0,4 ALIGN LIST TO FULLWORD
 980+START    BAL    1,*+8 LOAD REG1 W/LIST ADDR.
 981+         DC     AL1(143) OPTION BYTE
 982+         DC     AL3(DUMPAREA) DCB ADDRESS
 983+         SVC    19 ISSUE OPEN SVC
 984          LA         2,0
 985          LA         3,0
 986          LA         4,0
 987          LA         5,0
 988          LA         6,0
 989          LA         7,0
 990          LA         8,0
 991          LA         9,0
 992          LA         10,0
```

```
          LA        11,0
          LA        14,0
          LA        15,0
*
*
*
*
              SNAP      ID=1,DCB=DUMPAREA,PDATA=REGS
+         CNOP  0,4
+         BAL   1,IHB0003 BRANCH AROUND PARAM LIST
+         DC    AL1(1) ID NUMBER
+         DC    AL1(0)
+         DC    AL1(130) OPTION FLAGS
+         DC    AL1(32) OPTION FLAGS
+         DC    A(DUMPAREA) DCB ADDRESS
+         DC    A(0) TCB ADDRESS
+         DC    A(0) ADDRESS OF SNAP-SHOT LIST
+IHB0003  DS    0H
+         SVC   51
*
*
          DO        LOWNUM=3,BYLOC=Y,HGHREG=24,DOLOOP=2,LABEL=THREE,       X
                WHILE=NN,WOP=GT,WLITA=7,WREGB=15
+         ST    8,SAVE8
+         ST    9,SAVE9
+         ST    10,SAVE10
+         ST    11,SAVE11
+         LA    10,0
+         A     10,=F'3'
                *,INVALID NUMBER FOR 'HGHREG' SPECIFIED IN DO
                *,LOOP
+         B     CTHREE
                    A     3,=F'3'
                    A     4,=F'3'
                    A     5,=F'3'
                    A     6,=F'3'
                    A     7,=F'3'
                    A     8,=F'3'
                    A     9,=F'3'
                    A     10,=F'3'
          ENDDO     LABEL=THREE
+         B     THREE

+CTHREE   L     8,SAVE8
+         L     9,SAVE9
+BTHREE   L     10,SAVE10
+         L     11,SAVE11
+ATHREE   EQU   *
*
*
              SNAP      ID=2,DCB=DUMPAREA,PDATA=REGS
+         CNOP  0,4
+         BAL   1,IHB0006 BRANCH AROUND PARAM LIST
+         DC    AL1(2) ID NUMBER
+         DC    AL1(0)
+         DC    AL1(130) OPTION FLAGS
+         DC    AL1(32) OPTION FLAGS
+         DC    A(DUMPAREA) DCB ADDRESS
+         DC    A(0) TCB ADDRESS
+         DC    A(0) ADDRESS OF SNAP-SHOT LIST
+IHB0006  DS    0H
+         SVC   51
*
*
          EXIT
+         L     13,4(,13) POP UP SAVE AREA
+         LM    14,12,12(13) RESTORE REGISTERS
```

```
1058+              MVI    12(13),X'FF' FLAG EXIT
1059+              BR     14 RETURN
1060               CLOSE      DUMPAREA
1061+              CNOP   0,4 ALIGN LIST TO FULLWORD
1062+              BAL    1,*+8 LOAD REG1 W/LIST ADDR
1063+              DC     AL1(128) OPTION BYTE
1064+              DC     AL3(DUMPAREA) DCB ADDRESS
1065+              SVC    20 ISSUE CLOSE SVC

1066 SAVEAREA DC             18A(0)
1067              DS             0F
1068 ZERO         DC             1F'0'
1069 SAVE8        DS             1F
1070 SAVE9        DS             1F
1071 SAVE10       DS             1F
1072 SAVE11       DS             1F
1073 POINTER      DC             1F'0'
1074 STACK        DS             100F
1075 Y            DC             1F'1'
1076 DUMPAREA DCB            DDNAME=TEAM,DSORG=PS,RECFM=VBA,                    X
                          MACRF=W,BLKSIZE=882,LRECL=125


1078+*                             DATA CONTROL BLOCK
1079+*
1080+DUMPAREA DC          0F'0' ORIGIN ON WORD BOUNDARY

1082+*                             DIRECT ACCESS DEVICE INTERFACE

1084+              DC     BL16'0' FDAD,DVTBL
1085+              DC     A(0) KEYLE,DEVT,TRBAL

1087+*                             COMMON ACCESS METHOD INTERFACE

1089+              DC     AL1(0) BUFNO
1090+              DC     AL3(1) BUFCB
1091+              DC     AL2(0) BUFL
1092+              DC     BL2'0100000000000000' DSORG
1093+              DC     A(1) IOBAD

1095+*                             FOUNDATION EXTENSION

1097+              DC     BL1'00000000' BFTEK,BFLN,HIARCHY
1098+              DC     AL3(1) EODAD
1099+              DC     BL1'01010100' RECFM
1100+              DC     AL3(0) EXLST

1102+*                             FOUNDATION BLOCK

1104+              DC     CL8'TEAM' DDNAME
1105+              DC     BL1'00000010' OFLGS
1106+              DC     BL1'00000000' IFLG
1107+              DC     BL2'0000000000100000' MACR

1109+*                             BSAM-BPAM-QSAM INTERFACE

1111+              DC     BL1'00000000' RER1
1112+              DC     AL3(1) CHECK, GERR, PERR
1113+              DC     A(1) SYNAD
1114+              DC     H'0' CIND1, CIND2
1115+              DC     AL2(882) BLKSIZE
1116+              DC     F'0' WCPO, WCPL, OFFSR, OFFSW
1117+              DC     A(1) IOBA
1118+              DC     AL1(0) NCP
1119+              DC     AL3(1) EOBR, EOBAD

1121+*                             BSAM-BPAM INTERFACE
```

```
         DC    A(1) EOBW
         DC    H'0' DIRCT
         DC    AL2(125) LRECL
         DC    A(1) CNTRL, NOTE, POINT
         END
               =F'3'
```

APPENDIX B

THE LEAVE MACRO

The Source Listing for the LEAVE Macro

```
STMT   SOURCE STATEMENT

    *                        THE LEAVE MACRO PROVIDES A MEANS OF ABNORMALLY TER-
    *                        MINATING A LOOPING STRUCTURE.  IT MAY INVOKE A CON-
    *                        DITIONAL OR UNCONDITIONAL BRANCH FROM WITHIN A LOOP.
    *                        CONDITION PARAMETERS, WHICH MAY OR MAY NOT BE IN-
    *                        CLUDED, ARE SPECIFIED IN MUCH THE SAME MANNER AS A
    *                        DO WHILE CONDITION.  THE 'COND' PARAMETER IS MADE
    *                        NON-NULL WHEN A CONDITIONAL LEAVE IS DESIRED, AND
    *                        THE 'LABEL' PARAMETER SPECIFIES THE DESTINATION OF
    *                        THE BRANCH WHICH SHOULD NORMALLY BE TO THE END OF
    *                        THE LOOPING STRUCTURE.  LEAVE STATEMENTS SHOULD NOT
    *                        BE USED FOR BACKWARD BRANCHING.
    *
    *
             MACRO
             LEAVE &LABEL=,&COND=,&OPRATOR=,&REGA=,&LOCA=,&LITA=,&REGB=,     X
                   &LOCB=,&LITB=
             LCLA        &LAB
    &LAB     SETA        &SYSNDX
             AGO         .SKIP
    *
    *
    *                        INITIALLY THE 'COND' PARAMETER IS TESTED TO DETER-
    *                        MINE IF THE LEAVE STATEMENT IS CONDITIONAL OR UN-
    *                        CONDITIONAL.  IF UNCONDITIONAL, THEN A BRANCH TO
    *                        'LABEL' WILL OCCUR.  'LABEL' MUST BE SPECIFIED
    *                        EITHER IN ANOTHER MACRO OR IN A PROGRAM INSTRUC-
    *                        TION.
    *
    *
    .SKIP    AIF         ('&COND' NE '').A
             B           A&LABEL
             AGO         .STOP
    *
    *
    *                        WHEN THE LEAVE STATEMENT IS CONDITIONAL, REGISTERS
    *                        10 AND 11 (WORKING REGISTERS) ARE SAVED.  THEY WILL
    *                        BE USED TO TEST THE LEAVE CONDITION.
    *
    *
    .A       ST          10,SAVE10
             ST          11,SAVE11
             AGO         .B
    *
    *
    *                        THE FOLLOWING SECTION OF CODE DETERMINES THE LEFT
    *                        OPERAND OF THE LEAVE CONDITION.  IT MAY BE SPEC-
    *                        IFIED BY A REGISTER (REGA), A LOCATION (LOCA) OR BY
    *                        A LITERAL (LITA).  THE OPERAND IS LOADED INTO REG-
    *                        ISTER 10.  ILLEGALLY SPECIFIED OPERANDS WILL GEN-
```

```
*                           ERATE A MACRO ERROR CONDITION AND THE LEAVE MACRO
*                           WILL NOT BE EXECUTED.
*
*
.B          AIF             ('&REGA' EQ '').C
            AIF             (T'&REGA NE 'N').ERR1
            AIF             ('&REGA' GT '15').ERR2
            LR              10,&REGA
            AGO             .E
.C          AIF             ('&LOCA' EQ '').D
            L               10,&LOCA
            AGO             .E1
.D          AIF             ('&LITA' EQ '').ERR3
            AIF             ('&LITB' NE '').ERR4
.RETURN     LA              10,0
            A               10,=F'&LITA'
            AGO             .E2
.E          AIF             ('&LOCA' NE '').ERR5
.E1         AIF             ('&LITA' NE '').ERR5
            AGO             .E2
*
*
*                           THE RIGHT OPERAND IS DETERMINED IN THE NEXT SECTION
*                           OF CODE.  IT TOO MAY BE SPECIFIED BY REGISTER
*                           (REGB), BY LOCATION (LOCB) OR BY A LITERAL (LITB).
*                           THE RIGHT OPERAND IS LOADED INTO REGISTER 11 AND
*                           COMPARED TO REGISTER 10 (LEFT OPERAND).  IF THE
*                           RIGHT OPERAND HAS BEEN ILLEGALLY SPECIFIED , AN
*                           ERROR MESSAGE WILL BE PRINTED AND THE MACRO WILL NOT
*                           BE EXECUTED.
*
*
.E2         AIF             ('&REGB' EQ '').F
            AIF             (T'&REGB NE 'N').ERR6
            AIF             ('&REGB' GT '15').ERR7
            LR              11,&REGB
            AGO             .H
.F          AIF             ('&LOCB' EQ '').G
            L               11,&LOCB
            AGO             .H1
.G          AIF             ('&LITB' EQ '').ERR8
            LA              11,0
            A               11,=F'&LITB'
            AGO             .H2
.H          AIF             ('&LOCB' NE '').ERR9
.H1         AIF             ('&LITB' NE '').ERR9
.H2         CR              10,11
            AGO             .I
*
*
*                           THE FOLLOWING CODE SEGMENT DETERMINES THE LEAVE
*                           CONDITION OPERATOR AND GENERATES THE APPROPRIATE
*                           BRANCH INSTRUCTION.  A MISSING OR ILLEGAL OPERATOR
*                           WILL GENERATE AN ERROR MESSAGE AND THE MACRO WILL
*                           NOT BE EXECUTED.
*
*
.I          AIF             ('&OPRATOR' EQ '').ERR10
            AIF             ('&OPRATOR' EQ 'EQ').EQUAL
            AIF             ('&OPRATOR' EQ 'LT').LESS
            AIF             ('&OPRATOR' EQ 'LE').LESSEQ
            AIF             ('&OPRATOR' EQ 'GE').GRTEQ
            AIF             ('&OPRATOR' EQ 'GT').GRT
            AIF             ('&OPRATOR' EQ 'NE').NOTEQ
            AGO             .ERR11
.EQUAL      BNE             A&LAB
```

```
                AGO       .END
.LESS           BNL       A&LAB
                AGO       .END
.LESSEQ         BH        A&LAB
                AGO       .END
.GRTEQ          BL        A&LAB
                AGO       .END
.GRT            BNH       A&LAB
                AGO       .END
.NOTEQ          BE        A&LAB
                AGO       .END
*
*
*                         THE NEXT SET OF INSTRUCTIONS RESTORES REGISTERS
*                         10 AND 11 AFTER THE LEAVE CONDITION HAS BEEN TESTED.
*                         IF THE CONDITION IS TRUE, A BRANCH TO 'LABEL' WILL
*                         OCCUR; OTHERWISE, THE PROGRAM WILL CONTINUE SE-
*                         QUENTIAL EXECUTION.
*
*
.END            L         10,SAVE10
                L         11,SAVE11
                B         A&LABEL
A&LAB           L         10,SAVE10
                L         11,SAVE11
                AGO       .STOP
*
*
*                         THIS LAST SECTION OF CODE IS THE SET OF ERROR MES-
*                         SAGES ANY ONE OF WHICH MAY BE GENERATED IF THE
*                         MACRO'S SYMBOLIC PARAMETERS HAVE BEEN INCORRECTLY
*                         SPECIFIED.  WHEN AN ERROR MESSAGE IS PRINTED, THE
*                         MACRO TERMINATES (EXCEPT IN THE CASE OF AN .ERR4
*                         MESSAGE), NEVER GENERATING A PROGRAM BRANCH.
*
*
.ERR1           MNOTE     *,'''REGA'' OF LEAVE CONDITION IS NOT A SELF-'
                MNOTE     *,'DEFINING NUMERIC'
                B         A&LAB
                AGO       .END
.ERR2           MNOTE     *,'''REGA'' OF LEAVE CONDITION DOES NOT SPECIFY'
                MNOTE     *,'A VALID REGISTER NUMBER'
                B         A&LAB
                AGO       .END
.ERR3           MNOTE     *,'LEFT OPERAND OF LEAVE CONDITION WAS NOT SPECI-'
                MNOTE     *,'FIED'
                B         A&LAB
                AGO       .END
.ERR4           MNOTE     *,'TWO LITERALS HAVE BEEN SPECIFIED IN A LEAVE CON-'
                MNOTE     *,'DITION'
                AGO       .RETURN
                AGO       .END
.ERR5           MNOTE     *,'MORE THAN ONE LEFT OPERAND SPECIFIED FOR A LEAVE'
                MNOTE     *,'CONDITION'
                B         A&LAB
                AGO       .END
.ERR6           MNOTE     *,'''REGB'' OF LEAVE CONDITION IS NOT A SELF-'
                MNOTE     *,'DEFINING NUMERIC'
                B         A&LAB
                AGO       .END
.ERR7           MNOTE     *,'''REGB'' OF LEAVE CONDITION DOES NOT SPECIFY A'
                MNOTE     *,'VALID REGISTER NUMBER'
                B         A&LAB
                AGO       .END
.ERR8           MNOTE     *,'NO RIGHT OPERAND HAS BEEN SPECIFIED FOR A LEAVE'
                MNOTE     *,'CONDITION'
```

```
 694           B        A&LAB
 695           AGO      .END
 696 .ERR9     MNOTE    *,'MORE THAN ONE RIGHT OPERAND SPECIFIED FOR A'
 697           MNOTE    *,'LEAVE CONDITION'
 698           B        A&LAB
 699           AGO      .END
 700 .ERR10    MNOTE    *,'''OPRATOR'' FOR LEAVE CONDITION HAS NOT BEEN'
 701           MNOTE    *,'SPECIFIED'
 702           B        A&LAB
 703           AGO      .END
 704 .ERR11    MNOTE    *,'ILLEGAL OPERATOR SPECIFIED FOR A LEAVE CONDITION'
 705           B        A&LAB
 706           AGO      .END
 707 .STOP     MEND
```

Sample Program E Illustrating the LEAVE Macro

```
STMT   SOURCE STATEMENT

 962            PRINT    NOGEN
 963 FRISBEE    ENTER    12,SAVEAREA
 979 START      OPEN     (DUMPAREA,OUTPUT)
 985            LA       2,0
 986            LA       3,0
 987            LA       4,0
 988            LA       5,0
 989            LA       6,0
 990            LA       7,0
 991            LA       8,0
 992            LA       9,0
 993            LA       10,0
 994            LA       11,0
 995            LA       14,0
 996            LA       15,0
 997 *
 998 *
 999            SNAP     ID=1,DCB=DUMPAREA,PDATA=REGS
1011 *
1012 *
1013            DO       LOWNUM=1,BYNUM=1,HGHNUM=100,DOLOOP=A,LABEL=Z
1076                     A     4,=F'1'
1077                     A     5,=F'1'
1078                     LEAVE     LABEL=Z,LITA=7,OPRATOR=EQ,REGB=4,COND=F
1091                     A     6,=F'1'
1092                     A     7,=F'1'
1093            ENDDO    LABEL=Z
1100 *
1101 *
1102            SNAP     ID=2,DCB=DUMPAREA,PDATA=REGS
1114 *
1115 *
1116            DO       LOWNUM=1,BYNUM=1,HGHNUM=100,DOLOOP=X,LABEL=X
1179                     A     14,=F'1'
1180                     LEAVE   LABEL=X
1182                     A     15,=F'1'
1183            ENDDO    LABEL=X
1190 *
1191 *
1192            SNAP     ID=3,DCB=DUMPAREA,PDATA=REGS
1204 *
1205 *
1206            EXIT
```

```
1211              CLOSE      DUMPAREA

1217 SAVEAREA DC           18A(0)
1218              DS         0F
1219 ZERO         DC         1F'0'
1220 SAVE8        DS         1F
1221 SAVE9        DS         1F
1222 SAVE10       DS         1F
1223 SAVE11       DS         1F
1224 POINTER      DC         1F'0'
1225 STACK        DS         100F
1226 DUMPAREA DCB          DDNAME=TEAM,DSORG=PS,RECFM=VBA,                       X
                        MACRF=W,BLKSIZE=882,LRECL=125
1277              END
1278                     =F'1'
1279                     =F'100'
1280                     =F'4'
1281                     =F'7'
```

```
REGS AT ENTRY TO SNAP                                          ID = 001


REGS 0-7              00000030   90018888   00000000   00000000
                      00000000   00000000   00000000   00000000

REGS 8-15             00000000   00000000   00000000   00000000
                      50018820   00018824   00000000   00000000
END OF SNAP
```

```
REGS AT ENTRY TO SNAP                                          ID = 002


REGS 0-7              000002A0   80018908   00000000   00000000
                      00000007   00000007   00000006   00000006

REGS 8-15             00000000   00000000   00000000   00000000
                      50018820   00018824   00000000   00000000
END OF SNAP
```

```
REGS AT ENTRY TO SNAP                                     ID = 003


REGS 0-7              000002A0   A001BAF8   00000000   00000000
                      00000007   00000007   00000006   00000006


REGS 8-15             00000000   00000000   00000000   00000000
                      5001B820   0001B824   00000001   00000000

END OF SNAP
```

Program E with Expanded Macros

```
STMT    SOURCE STATEMENT

 962 FRISBEE  ENTER      12,SAVEAREA
 963+FRISBEE  DS     0H
 964+         ENTRY FRISBEE DECLARE NAME ENTRY
 965+         USING *,12 DECLARE BASE ADDRESSIBILITY.
 966+         BALR   15,0 (INITIAL ADDRESSIBILITY).
 967+         B      12(,15) BRANCH AROUND ID FIELD
 968+         DC     AL1(7),CL7'FRISBEE' ID LENGTH AND ID
 969+         BCTR   15,0 (RESET INITIAL ADDRESSIBILITY
 970+         BCTR   15,0 ABSOLUTE ENTRY POINT).
 971+         STM    14,12,12(13) SAVE REGISTERS
 972+         LR     12,15 SETUP BASE REGISTER.
 973+         ST     13,SAVEAREA+4 CHAIN BACK
 974+         LA     0,SAVEAREA CHAIN FORWARD
 975+         ST     0,8(0,13)
 976+         LR     13,0 SET UP SAVE AREA POINTER
 977+         USING SAVEAREA,13 AND ADDRESSABILITY
 978 START    OPEN       (DUMPAREA,OUTPUT)
 979+         CNOP   0,4 ALIGN LIST TO FULLWORD
 980+START    BAL    1,*+8 LOAD REG1 W/LIST ADDR.
 981+         DC     AL1(143) OPTION BYTE
 982+         DC     AL3(DUMPAREA) DCB ADDRESS
 983+         SVC    19 ISSUE OPEN SVC
 984          LA         2,0
 985          LA         3,0
 986          LA         4,0
 987          LA         5,0
 988          LA         6,0
 989          LA         7,0
 990          LA         8,0
 991          LA         9,0
 992          LA         10,0
 993          LA         11,0
 994          LA         14,0
 995          LA         15,0
 996 *
 997 *
 998          SNAP       ID=1,DCB=DUMPAREA,PDATA=REGS
 999+         CNOP   0,4
1000+         BAL    1,IHB0003 BRANCH AROUND PARAM LIST
1001+         DC     AL1(1) ID NUMBER
1002+         DC     AL1(0)
1003+         DC     AL1(130) OPTION FLAGS
1004+         DC     AL1(32) OPTION FLAGS
1005+         DC     A(DUMPAREA) DCB ADDRESS
1006+         DC     A(0) TCB ADDRESS
1007+         DC     A(0) ADDRESS OF SNAP-SHOT LIST
```

```
+IHB0003  DS    0H
+         SVC   51
 *
 *
          DO        LOWNUM=1,BYNUM=1,HGHNUM=100,DOLOOP=A,LABEL=Z
+         ST    8,SAVE8
+         ST    9,SAVE9
+         ST    10,SAVE10
+         ST    11,SAVE11
+         LA    10,0
+         A     10,=F'1'
+         LA    11,0
+         A     11,=F'100'
+         LA    9,0
+         A     9,=F'1'
+         L     8,POINTER
+*
+         ST    9,STACK(8)
+         LA    8,4(8)
+         ST    10,STACK(8)
+         LA    8,4(8)
+         ST    11,STACK(8)
+         LA    8,4(8)
+         ST    8,POINTER
+         L     8,SAVE8
+         L     9,SAVE9
+         L     10,SAVE10
+         L     11,SAVE11
+Z        ST    8,SAVE8
+         ST    9,SAVE9
+         ST    10,SAVE10
+         ST    11,SAVE11
+         L     8,POINTER
+         S     8,=F'4'
+         L     11,STACK(8)
+         S     8,=F'4'
+         L     10,STACK(8)
+         S     8,=F'4'
+         L     9,STACK(8)
+         ST    8,POINTER
+         C     9,ZERO
+         BL    X4
+         CR    10,11
+         BH    Y4
+         B     Z4
+X4       CR    10,11
+         BL    Y4
+Z4       AR    10,9
+         L     8,POINTER
+         ST    9,STACK(8)
+         LA    8,4(8)
+         ST    10,STACK(8)
+         LA    8,4(8)
+         ST    11,STACK(8)
+         LA    8,4(8)
+         ST    8,POINTER
+         L     8,SAVE8
+         L     9,SAVE9
+         L     10,SAVE10
+         L     11,SAVE11
+         B     W4
+Y4       L     8,SAVE8
+         L     9,SAVE9
+         L     10,SAVE10
+         L     11,SAVE11
+         B     AZ
```

```
+W4           EQU      *
                            A       4,=F'1'
                            A       5,=F'1'
                            LEAVE      LABEL=Z,LITA=7,OPRATOR=EQ,REGB=4,COND=F
+             ST       10,SAVE10
+             ST       11,SAVE11
+             LA       10,0
+             A        10,=F'7'
+             LR       11,4
+             CR       10,11
+             BNE      A5
+             L        10,SAVE10
+             L        11,SAVE11
+             B        AZ
+A5           L        10,SAVE10
+             L        11,SAVE11
                            A       6,=F'1'
                            A       7,=F'1'
              ENDDO         LABEL=Z
+             B        Z
+CZ           L        8,SAVE8
+             L        9,SAVE9
+BZ           L        10,SAVE10
+             L        11,SAVE11
+AZ           EQU      *
*
*
              SNAP          ID=2,DCB=DUMPAREA,PDATA=REGS
+             CNOP     0,4
+             BAL      1,IHB0007 BRANCH AROUND PARAM LIST
+             DC       AL1(2) ID NUMBER
+             DC       AL1(0)
+             DC       AL1(130) OPTION FLAGS
+             DC       AL1(32) OPTION FLAGS
+             DC       A(DUMPAREA) DCB ADDRESS
+             DC       A(0) TCB ADDRESS
+             DC       A(0) ADDRESS OF SNAP-SHOT LIST
+IHB0007      DS       0H
+             SVC      51
*
*
              DO            LOWNUM=1,BYNUM=1,HGHNUM=100,DOLOOP=X,LABEL=X
+             ST       8,SAVE8
+             ST       9,SAVE9
+             ST       10,SAVE10
+             ST       11,SAVE11
+             LA       10,0
+             A        10,=F'1'
+             LA       11,0
+             A        11,=F'100'
+             LA       9,0
+             A        9,=F'1'
+             L        8,POINTER
+*
+             ST       9,STACK(8)
+             LA       8,4(8)
+             ST       10,STACK(8)
+             LA       8,4(8)
+             ST       11,STACK(8)
+             LA       8,4(8)
+             ST       8,POINTER
+             L        8,SAVE8
+             L        9,SAVE9
+             L        10,SAVE10
+             L        11,SAVE11
+X            ST       8,SAVE8
+             ST       9,SAVE9
```

```
1141+        ST     10,SAVE10
1142+        ST     11,SAVE11
1143+        L      8,POINTER
1144+        S      8,=F'4'
1145+        L      11,STACK(8)
1146+        S      8,=F'4'
1147+        L      10,STACK(8)
1148+        S      8,=F'4'
1149+        L      9,STACK(8)
1150+        ST     8,POINTER
1151+        C      9,ZERO
1152+        BL     X8
1153+        CR     10,11
1154+        BH     Y8
1155+        B      Z8
1156+X8      CR     10,11
1157+        BL     Y8
1158+Z8      AR     10,9
1159+        L      8,POINTER
1160+        ST     9,STACK(8)
1161+        LA     8,4(8)
1162+        ST     10,STACK(8)
1163+        LA     8,4(8)
1164+        ST     11,STACK(8)
1165+        LA     8,4(8)
1166+        ST     8,POINTER
1167+        L      8,SAVE8
1168+        L      9,SAVE9
1169+        L      10,SAVE10
1170+        L      11,SAVE11
1171+        B      W8
1172+Y8      L      8,SAVE8
1173+        L      9,SAVE9
1174+        L      10,SAVE10
1175+        L      11,SAVE11
1176+        B      AX
1177+W8      EQU    *
1178                       A      14,=F'1'
1179                       LEAVE    LABEL=X
1180+        B      AX
1181                       A      15,=F'1'
1182         ENDDO         LABEL=X
1183+        B      X
1184+CX      L      8,SAVE8
1185+        L      9,SAVE9
1186+BX      L      10,SAVE10
1187+        L      11,SAVE11
1188+AX      EQU    *
1189 *
1190 *
1191         SNAP          ID=3,DCB=DUMPAREA,PDATA=REGS
1192+        CNOP   0,4
1193+        BAL    1,IHB0011 BRANCH AROUND PARAM LIST
1194+        DC     AL1(3) ID NUMBER
1195+        DC     AL1(0)
1196+        DC     AL1(130) OPTION FLAGS
1197+        DC     AL1(32) OPTION FLAGS
1198+        DC     A(DUMPAREA) DCB ADDRESS
1199+        DC     A(0) TCB ADDRESS
1200+        DC     A(0) ADDRESS OF SNAP-SHOT LIST
1201+IHB0011 DS     0H
1202+        SVC    51
1203 *
1204 *
1205         EXIT
1206+        L      13,4(,13) POP UP SAVE AREA
```

```
1207+           LM     14,12,12(13) RESTORE REGISTERS
1208+           MVI    12(13),X'FF' FLAG EXIT
1209+           BR     14 RETURN
1210            CLOSE      DUMPAREA
1211+           CNOP   0,4 ALIGN LIST TO FULLWORD
1212+           BAL    1,*+8 LOAD REG1 W/LIST ADDR
1213+           DC     AL1(128) OPTION BYTE
1214+           DC     AL3(DUMPAREA) DCB ADDRESS
1215+           SVC    20 ISSUE CLOSE SVC

1216 SAVEAREA DC           18A(0)
1217          DS           0F
1218 ZERO     DC           1F'0'
1219 SAVE8    DS           1F
1220 SAVE9    DS           1F
1221 SAVE10   DS           1F
1222 SAVE11   DS           1F
1223 POINTER  DC           1F'0'
1224 STACK    DS           100F
1225 DUMPAREA DCB          DDNAME=TEAM,DSORG=PS,RECFM=VBA,                  X
                       MACRF=W,BLKSIZE=882,LRECL=125


1227+*                            DATA CONTROL BLOCK
1228+*
1229+DUMPAREA DC      0F'0' ORIGIN ON WORD BOUNDARY

1231+*                            DIRECT ACCESS DEVICE INTERFACE

1233+           DC     BL16'0' FDAD,DVTBL
1234+           DC     A(0) KEYLE,DEVT,TRBAL

1236+*                            COMMON ACCESS METHOD INTERFACE

1238+           DC     AL1(0) BUFNO
1239+           DC     AL3(1) BUFCB
1240+           DC     AL2(0) BUFL
1241+           DC     BL2'0100000000000000' DSORG
1242+           DC     A(1) IOBAD

1244+*                            FOUNDATION EXTENSION

1246+           DC     BL1'00000000' BFTEK,BFLN,HIARCHY
1247+           DC     AL3(1) EODAD
1248+           DC     BL1'01010100' RECFM
1249+           DC     AL3(0) EXLST

1251+*                            FOUNDATION BLOCK

1253+           DC     CL8'TEAM' DDNAME
1254+           DC     BL1'00000010' OFLGS
1255+           DC     BL1'00000000' IFLG
1256+           DC     BL2'0000000000100000' MACR

1258+*                            BSAM-BPAM-QSAM INTERFACE

1260+           DC     BL1'00000000' RER1
1261+           DC     AL3(1) CHECK, GERR, PERR
1262+           DC     A(1) SYNAD
1263+           DC     H'0' CIND1, CIND2
1264+           DC     AL2(882) BLKSIZE
1265+           DC     F'0' WCPO, WCPL, OFFSR, OFFSW
1266+           DC     A(1) IOBA
1267+           DC     AL1(0) NCP
1268+           DC     AL3(1) EOBR, EOBAD

1270+*                              BSAM-BPAM INTERFACE
```

```
1272+         DC    A(1) EOBW
1273+         DC    H'0' DIRCT
1274+         DC    AL2(125) LRECL
1275+         DC    A(1) CNTRL, NOTE, POINT
1276          END
1277                =F'1'
1278                =F'100'
1279                =F'4'
1280                =F'7'
```

Sample Program F Illustrating a LEAVE Macro Coded Incorrectly

```
STMT   SOURCE STATEMENT

 962 FRISBEE  ENTER     12,SAVEAREA
 963+FRISBEE  DS    0H
 964+         ENTRY FRISBEE DECLARE NAME ENTRY
 965+         USING *,12 DECLARE BASE ADDRESSIBILITY.
 966+         BALR  15,0 (INITIAL ADDRESSIBILITY).
 967+         B     12(,15) BRANCH AROUND ID FIELD
 968+         DC    AL1(7),CL7'FRISBEE' ID LENGTH AND ID
 969+         BCTR  15,0 (RESET INITIAL ADDRESSIBILITY
 970+         BCTR  15,0 ABSOLUTE ENTRY POINT).
 971+         STM   14,12,12(13) SAVE REGISTERS
 972+         LR    12,15 SETUP BASE REGISTER.
 973+         ST    13,SAVEAREA+4 CHAIN BACK
 974+         LA    0,SAVEAREA CHAIN FORWARD
 975+         ST    0,8(0,13)
 976+         LR    13,0 SET UP SAVE AREA POINTER
 977+         USING SAVEAREA,13 AND ADDRESSABILITY
 978 START    OPEN      (DUMPAREA,OUTPUT)
 979+         CNOP  0,4 ALIGN LIST TO FULLWORD
 980+START    BAL   1,*+8 LOAD REG1 W/LIST ADDR.
 981+         DC    AL1(143) OPTION BYTE
 982+         DC    AL3(DUMPAREA) DCB ADDRESS
 983+         SVC   19 ISSUE OPEN SVC
 984          LA        2,0
 985          LA        3,0
 986          LA        4,0
 987          LA        5,0
 988          LA        6,0
 989          LA        7,0
 990          LA        8,0
 991          LA        9,0
 992          LA        10,0
 993          LA        11,0
 994          LA        14,0
 995          LA        15,0
 996 *
 997 *
 998               SNAP      ID=1,DCB=DUMPAREA,PDATA=REGS
 999+         CNOP  0,4
1000+         BAL   1,IHB0003 BRANCH AROUND PARAM LIST
1001+         DC    AL1(1) ID NUMBER
1002+         DC    AL1(0)
1003+         DC    AL1(130) OPTION FLAGS
1004+         DC    AL1(32) OPTION FLAGS
1005+         DC    A(DUMPAREA) DCB ADDRESS
1006+         DC    A(0) TCB ADDRESS
1007+         DC    A(0) ADDRESS OF SNAP-SHOT LIST
1008+IHB0003  DS    0H
```

```
1009+           SVC    51
1010 *
1011 *
1012            DO          LOWNUM=1,BYNUM=1,HGHNUM=100,DOLOOP=A,LABEL=Z
1013+           ST     8,SAVE8
1014+           ST     9,SAVE9
1015+           ST     10,SAVE10
1016+           ST     11,SAVE11
1017+           LA     10,0
1018+           A      10,=F'1'
1019+           LA     11,0
1020+           A      11,=F'100'
1021+           LA     9,0
1022+           A      9,=F'1'
1023+           L      8,POINTER
1024+           ST     9,STACK(8)
1025+           LA     8,4(8)
1026+           ST     10,STACK(8)
1027+           LA     8,4(8)
1028+           ST     11,STACK(8)
1029+           LA     8,4(8)
1030+           ST     8,POINTER
1031+           L      8,SAVE8
1032+           L      9,SAVE9
1033+           L      10,SAVE10
1034+           L      11,SAVE11
1035+Z          ST     8,SAVE8
1036+           ST     9,SAVE9
1037+           ST     10,SAVE10
1038+           ST     11,SAVE11
1039+           L      8,POINTER
1040+           S      8,=F'4'
1041+           L      11,STACK(8)
1042+           S      8,=F'4'
1043+           L      10,STACK(8)
1044+           S      8,=F'4'
1045+           L      9,STACK(8)
1046+           ST     8,POINTER
1047+           C      9,ZERO
1048+           BL     X4
1049+           CR     10,11
1050+           BH     Y4
1051+           B      Z4
1052+X4         CR     10,11
1053+           BL     Y4
1054+Z4         AR     10,9
1055+           L      8,POINTER
1056+           ST     9,STACK(8)
1057+           LA     8,4(8)
1058+           ST     10,STACK(8)
1059+           LA     8,4(8)
1060+           ST     11,STACK(8)
1061+           LA     8,4(8)
1062+           ST     8,POINTER
1063+           L      8,SAVE8
1064+           L      9,SAVE9
1065+           L      10,SAVE10
1066+           L      11,SAVE11
1067+           B      W4
1068+Y4         L      8,SAVE8
1069+           L      9,SAVE9
1070+           L      10,SAVE10
1071+           L      11,SAVE11
1072+           B      AZ
1073+W4         EQU    *
1074                        A       4,=F'1'
```

```
1075                      A     5,=F'1'
1076                      LEAVE    LABEL=Z,REGA=-5,OPRATJR=EQ,REGB=4,COND=V
1077+          ST     10,SAVE10
1078+          ST     11,SAVE11
1079                  *,'REGA' OF LEAVE CONDITION IS NOT A SELF-
1080                  *,DEFINING NUMERIC
1081+          B      A5
1082+          L      10,SAVE10
1083+          L      11,SAVE11
1084+          B      AZ
1085+A5        L      10,SAVE10
1086+          L      11,SAVE11
1087                      A     6,=F'1'
1088                      A     7,=F'1'
1089           ENDDO      LABEL=Z
1090+          B      Z
1091+CZ        L      8,SAVE8
1092+          L      9,SAVE9
1093+BZ        L      10,SAVE10
1094+          L      11,SAVE11
1095+AZ        EQU    *
1096 *
1097 *
1098                SNAP      ID=2,DCB=DUMPAREA,PDATA=REGS
1099+          CNOP   0,4
1100+          BAL    1,IHB0007 BRANCH AROUND PARAM LIST
1101+          DC     AL1(2) ID NUMBER
1102+          DC     AL1(0)
1103+          DC     AL1(130) OPTION FLAGS
1104+          DC     AL1(32) OPTION FLAGS
1105+          DC     A(DUMPAREA) DCB ADDRESS
1106+          DC     A(0) TCB ADDRESS
1107+          DC     A(0) ADDRESS OF SNAP-SHOT LIST
1108+IHB0007   DS     0H
1109+          SVC    51
1110 *
1111 *
1112           EXIT
1113+          L      13,4(,13) POP UP SAVE AREA
1114+          LM     14,12,12(13) RESTDRE REGISTERS
1115+          MVI    12(13),X'FF' FLAG EXIT
1116+          BR     14 RETURN
1117           CLOSE     DUMPAREA
1118+          CNOP   0,4 ALIGN LIST TO FULLWORD
1119+          BAL    1,*+8 LOAD REG1 W/LIST ADDR
1120+          DC     AL1(128) OPTION BYTE
1121+          DC     AL3(DUMPAREA) DCB ADDRESS
1122+          SVC    20 ISSUE CLOSE SVC

1123 SAVEAREA DC          18A(0)
1124           DS          0F
1125 ZERO      DC          1F'0'
1126 SAVE8     DS          1F
1127 SAVE9     DS          1F
1128 SAVE10    DS          1F
1129 SAVE11    DS          1F
1130 POINTER   DC          1F'0'
1131 STACK     DS          100F
1132 DUMPAREA DCB          DDNAME=TEAM,DSORG=PS,RECFM=VBA,                    X
                   MACRF=W,BLKSIZE=882,LRECL=125


1134+*                           DATA CONTROL BLOCK
1135+*
1136+DUMPAREA DC     OF'0' ORIGIN ON WORD BOUNDARY

1138+*                           DIRECT ACCESS DEVICE INTERFACE
```

```
1140+          DC     BL16'0' FDAD,DVTBL
1141+          DC     A(0) KEYLE,DEVT,TRBAL

1143+*                     COMMON ACCESS METHOD INTERFACE

1145+          DC     AL1(0) BUFNO
1146+          DC     AL3(1) BUFCB
1147+          DC     AL2(0) BUFL
1148+          DC     BL2'0100000000000000' DSORG
1149+          DC     A(1) IOBAD

1151+*                      FOUNDATION EXTENSION

1153+          DC     BL1'00000000' BFTEK,BFLN,HIARCHY
1154+          DC     AL3(1) EODAD
1155+          DC     BL1'01010100' RECFM
1156+          DC     AL3(0) EXLST

1158+*                      FOUNDATION BLOCK

1160+          DC     CL8'TEAM' DDNAME
1161+          DC     BL1'00000010' OFLGS
1162+          DC     BL1'00000000' IFLG
1163+          DC     BL2'0000000000100000' MACR

1165+*                      BSAM-BPAM-QSAM INTERFACE

1167+          DC     BL1'00000000' RERI
1168+          DC     AL3(1) CHECK, GERR, PERR
1169+          DC     A(1) SYNAD
1170+          DC     H'0' CIND1, CIND2
1171+          DC     AL2(882) BLKSIZE
1172+          DC     F'0' WCPO, WCPL, OFFSR, OFFSW
1173+          DC     A(1) IOBA
1174+          DC     AL1(0) NCP
1175+          DC     AL3(1) EOBR, EOBAD

1177+*                        BSAM-BPAM INTERFACE

1179+          DC     A(1) EOBW
1180+          DC     H'0' DIRCT
1181+          DC     AL2(125) LRECL
1182+          DC     A(1) CNTRL, NOTE, POINT
1183           END
1184                  =F'1'
1185                  =F'100'
1186                  =F'4'
```

Sample Program G Illustrating a LEAVE Macro Coded Incorrectly

```
STMT   SOURCE STATEMENT

 962 FRISBEE  ENTER      12,SAVEAREA
 963+FRISBEE  DS     0H
 964+         ENTRY FRISBEE DECLARE NAME ENTRY
 965+         USING *,12 DECLARE BASE ADDRESSIBILITY.
 966+         BALR  15,0 (INITIAL ADDRESSIBILITY).
 967+         B     12(,15) BRANCH AROUND ID FIELD
 968+         DC    AL1(7),CL7'FRISBEE' ID LENGTH AND ID
 969+         BCTR  15,0 (RESET INITIAL ADDRESSIBILITY
 970+         BCTR  15,0 ABSOLUTE ENTRY POINT).
 971+         STM   14,12,12(13) SAVE REGISTERS
```

```
 972+           LR     12,15 SETUP BASE REGISTER.
 973+           ST     13,SAVEAREA+4 CHAIN BACK
 974+           LA     0,SAVEAREA CHAIN FORWARD
 975+           ST     0,8(0,13)
 976+           LR     13,0 SET UP SAVE AREA POINTER
 977+           USING SAVEAREA,13 AND ADDRESSABILITY
 978 START      OPEN       (DUMPAREA,OUTPUT)
 979+           CNOP   0,4 ALIGN LIST TO FULLWORD
 980+START      BAL    1,*+8 LOAD REG1 W/LIST ADDR.
 981+           DC     AL1(143) OPTION BYTE
 982+           DC     AL3(DUMPAREA) DCB ADDRESS
 983+           SVC    19 ISSUE OPEN SVC
 984            LA         2,0
 985            LA         3,0
 986            LA         4,0
 987            LA         5,0
 988            LA         6,0
 989            LA         7,0
 990            LA         8,0
 991            LA         9,0
 992            LA         10,0
 993            LA         11,0
 994            LA         14,0
 995            LA         15,0
 996 *
 997 *
 998               SNAP      ID=1,DCB=DUMPAREA,PDATA=REGS
 999+           CNOP  0,4
1000+           BAL    1,IHB0003 BRANCH AROUND PARAM LIST
1001+           DC     AL1(1) ID NUMBER
1002+           DC     AL1(0)
1003+           DC     AL1(130) OPTION FLAGS
1004+           DC     AL1(32) OPTION FLAGS
1005+           DC     A(DUMPAREA) DCB ADDRESS
1006+           DC     A(0) TCB ADDRESS
1007+           DC     A(0) ADDRESS OF SNAP-SHOT LIST
1008+IHB0003    DS     0H
1009+           SVC    51
1010 *
1011 *
1012            DO        LOWNUM=1,BYNUM=1,HGHNUM=100,DOLOOP=A,LABEL=Z
1013+           ST     8,SAVE8
1014+           ST     9,SAVE9
1015+           ST     10,SAVE10
1016+           ST     11,SAVE11
1017+           LA     10,0
1018+           A      10,=F'1'
1019+           LA     11,0
1020+           A      11,=F'100'
1021+           LA     9,0
1022+           A      9,=F'1'
1023+           L      8,POINTER
1024+           ST     9,STACK(8)
1025+           LA     8,4(8)
1026+           ST     10,STACK(8)
1027+           LA     8,4(8)
1028+           ST     11,STACK(8)
1029+           LA     8,4(8)
1030+           ST     8,POINTER
1031+           L      8,SAVE8
1032+           L      9,SAVE9
1033+           L      10,SAVE10
1034+           L      11,SAVE11
1035+Z          ST     8,SAVE8
1036+           ST     9,SAVE9
1037+           ST     10,SAVE10
```

```
+          ST      11,SAVE11
+          L       8,POINTER
+          S       8,=F'4'
+          L       11,STACK(8)
+          S       8,=F'4'
+          L       10,STACK(8)
+          S       8,=F'4'
+          L       9,STACK(8)
+          ST      8,POINTER
+          C       9,ZERO
+          BL      X4
+          CR      10,11
+          BH      Y4
+          B       Z4
+X4        CR      10,11
+          BL      Y4
+Z4        AR      10,9
+          L       8,POINTER
+          ST      9,STACK(8)
+          LA      8,4(8)
+          ST      10,STACK(8)
+          LA      8,4(8)
+          ST      11,STACK(8)
+          LA      8,4(8)
+          ST      8,POINTER
+          L       8,SAVE8
+          L       9,SAVE9
+          L       10,SAVE10
+          L       11,SAVE11
+          B       W4
+Y4        L       8,SAVE8
+          L       9,SAVE9
+          L       10,SAVE10
+          L       11,SAVE11
+          B       AZ
+W4        EQU     *
                        A      4,=F'1'
                        A      5,=F'1'
                        LEAVE     LABEL=Z,OPRATOR=EQ,REGB=4,COND=K
+          ST      10,SAVE10
+          ST      11,SAVE11
                   *,LEFT OPERAND OF LEAVE CONDITION WAS NOT SPECI-
                   *,FIED
+          B       A5
+          L       10,SAVE10
+          L       11,SAVE11
+          B       AZ
+A5        L       10,SAVE10
+          L       11,SAVE11
                        A      6,=F'1'
                        A      7,=F'1'
           ENDDO        LABEL=Z
+          B       Z
+CZ        L       8,SAVE8
+          L       9,SAVE9
+BZ        L       10,SAVE10
+          L       11,SAVE11
+AZ        EQU     *
 *
 *
                SNAP       ID=2,DCB=DUMPAREA,PDATA=REGS
+          CNOP    0,4
+          BAL     1,IHB0007 BRANCH AROUND PARAM LIST
+          DC      AL1(2) ID NUMBER
+          DC      AL1(0)
+          DC      AL1(130) OPTION FLAGS
```

```
1104+           DC     AL1(32) OPTION FLAGS
1105+           DC     A(DUMPAREA) DCB ADDRESS
1106+           DC     A(0) TCB ADDRESS
1107+           DC     A(0) ADDRESS OF SNAP-SHOT LIST
1108+IHB0007    DS     0H
1109+           SVC    51
1110 *
1111 *
1112            EXIT
1113+           L      13,4(,13) POP UP SAVE AREA
1114+           LM     14,12,12(13) RESTORE REGISTERS
1115+           MVI    12(13),X'FF' FLAG EXIT
1116+           BR     14 RETURN
1117            CLOSE     DUMPAREA
1118+           CNOP   0,4 ALIGN LIST TO FULLWORD
1119+           BAL    1,*+8 LOAD REG1 W/LIST ADDR
1120+           DC     AL1(128) OPTION BYTE
1121+           DC     AL3(DUMPAREA) DCB ADDRESS
1122+           SVC    20 ISSUE CLOSE SVC

1123 SAVEAREA DC          18A(0)
1124          DS          0F
1125 ZERO     DC          1F'0'
1126 SAVE8    DS          1F
1127 SAVE9    DS          1F
1128 SAVE10   DS          1F
1129 SAVE11   DS          1F
1130 POINTER  DC          1F'0'
1131 STACK    DS          100F
1132 DUMPAREA DCB         DDNAME=TEAM,DSORG=PS,RECFM=VBA,                  X
                   MACRF=W,BLKSIZE=882,LRECL=125


1134+*                           DATA CONTROL BLOCK
1135+*
1136+DUMPAREA DC       0F'0' ORIGIN ON WORD BOUNDARY

1138+*                           DIRECT ACCESS DEVICE INTERFACE

1140+           DC     BL16'0' FDAD,DVTBL
1141+           DC     A(0) KEYLE,DEVT,TRBAL

1143+*                           COMMON ACCESS METHOD INTERFACE

1145+           DC     AL1(0) BUFNO
1146+           DC     AL3(1) BUFCB
1147+           DC     AL2(0) BUFL
1148+           DC     BL2'0100000000000000' DSORG
1149+           DC     A(1) IOBAD

1151+*                           FOUNDATION EXTENSION

1153+           DC     BL1'00000000' BFTEK,BFLN,HIARCHY
1154+           DC     AL3(1) EODAD
1155+           DC     BL1'01010100' RECFM
1156+           DC     AL3(0) EXLST

1158+*                           FOUNDATION BLOCK

1160+           DC     CL8'TEAM' DDNAME
1161+           DC     BL1'00000010' OFLGS
1162+           DC     BL1'00000000' IFLG
1163+           DC     BL2'0000000000100000' MACR

1165+*                           BSAM-BPAM-QSAM INTERFACE
```

```
1167+          DC       BL1'00000000' PER1
1168+          DC       AL3(1) CHECK, GERR, PERR
1169+          DC       A(1) SYNAD
1170+          DC       H'0' CIND1, CIND2
1171+          DC       AL2(882) BLKSIZE
1172+          DC       F'0' WCPO, WCPL, OFFSR, OFFSW
1173+          DC       A(1) IOBA
1174+          DC       AL1(0) NCP
1175+          DC       AL3(1) EOBR, EOBAD

1177+*                               BSAM-BPAM INTERFACE

1179+          DC       A(1) EOBW
1180+          DC       H'0' DIRCT
1181+          DC       AL2(125) LRECL
1182+          DC       A(1) CNTRL, NOTE, POINT
1183           END
1184                    =F'1'
1185                    =F'100'
1186                    =F'4'
```

Sample Program H Illustrating a LEAVE Macro Coded Incorrectly

```
STMT   SOURCE STATEMENT

 962 FRISBEE  ENTER      12,SAVEAREA
 963+FRISBEE  DS     0H
 964+         ENTRY FRISBEE DECLARE NAME ENTRY
 965+         USING *,12 DECLARE BASE ADDRESSIBILITY.
 966+         BALR  15,0 (INITIAL ADDRESSIBILITY).
 967+         B     12(,15) BRANCH AROUND ID FIELD
 968+         DC    AL1(7),CL7'FRISBEE' ID LENGTH AND ID
 969+         BCTR  15,0 (RESET INITIAL ADDRESSIBILITY
 970+         BCTR  15,0 ABSOLUTE ENTRY POINT).
 971+         STM   14,12,12(13) SAVE REGISTERS
 972+         LR    12,15 SETUP BASE REGISTER.
 973+         ST    13,SAVEAREA+4 CHAIN BACK
 974+         LA    0,SAVEAREA CHAIN FORWARD
 975+         ST    0,8(0,13)
 976+         LR    13,0 SET UP SAVE AREA POINTER
 977+         USING SAVEAREA,13 AND ADDRESSABILITY
 978 START    OPEN      (DUMPAREA,OUTPUT)
 979+         CNOP  0,4 ALIGN LIST TO FULLWORD
 980+START    BAL   1,*+8 LOAD REG1 W/LIST ADDR.
 981+         DC    AL1(143) OPTION BYTE
 982+         DC    AL3(DUMPAREA) DCB ADDRESS
 983+         SVC   19 ISSUE OPEN SVC
 984          LA        2,0
 985          LA        3,0
 986          LA        4,0
 987          LA        5,0
 988          LA        6,0
 989          LA        7,0
 990          LA        8,0
 991          LA        9,0
 992          LA        10,0
 993          LA        11,0
 994          LA        14,0
 995          LA        15,0
 996 *
 997 *
 998                SNAP      ID=1,DCB=DUMPAREA,PDATA=REGS
```

```
         CNOP  0,4
         BAL   1,IHB0003 BRANCH AROUND PARAM LIST
         DC    AL1(1) ID NUMBER
         DC    AL1(0)
         DC    AL1(130) OPTION FLAGS
         DC    AL1(32) OPTION FLAGS
         DC    A(DUMPAREA) DCB ADDRESS
         DC    A(0) TCB ADDRESS
         DC    A(0) ADDRESS OF SNAP-SHOT LIST
IHB0003  DS    0H
         SVC   51
*
*
         DO       LOWNUM=1,BYNUM=1,HGHNUM=100,DOLOOP=A,LABEL=Z
         ST    8,SAVE8
         ST    9,SAVE9
         ST    10,SAVE10
         ST    11,SAVE11
         LA    10,0
         A     10,=F'1'
         LA    11,0
         A     11,=F'100'
         LA    9,0
         A     9,=F'1'
         L     8,POINTER
         ST    9,STACK(8)
         LA    8,4(8)
         ST    10,STACK(8)
         LA    8,4(8)
         ST    11,STACK(8)
         LA    8,4(8)
         ST    8,POINTER
         L     8,SAVE8
         L     9,SAVE9
         L     10,SAVE10
         L     11,SAVE11
Z        ST    8,SAVE8
         ST    9,SAVE9
         ST    10,SAVE10
         ST    11,SAVE11
         L     8,POINTER
         S     8,=F'4'
         L     11,STACK(8)
         S     8,=F'4'
         L     10,STACK(8)
         S     8,=F'4'
         L     9,STACK(8)
         ST    8,POINTER
         C     9,ZERO
         BL    X4
         CR    10,11
         BH    Y4
         B     Z4
X4       CR    10,11
         BL    Y4
Z4       AR    10,9
         L     8,POINTER
         ST    9,STACK(8)
         LA    8,4(8)
         ST    10,STACK(8)
         LA    8,4(8)
         ST    11,STACK(8)
         LA    8,4(8)
         ST    8,POINTER
         L     8,SAVE8
         L     9,SAVE9
```

```
1065+           L       10,SAVE10
1066+           L       11,SAVE11
1067+           B       W4
1068+Y4         L       8,SAVE8
1069+           L       9,SAVE9
1070+           L       10,SAVE10
1071+           L       11,SAVE11
1072+           B       AZ
1073+W4         EQU     *
1074                            A       4,=F'1'
1075                            A       5,=F'1'
1076                            LEAVE      LABEL=Z,LITA=7,OPRATOR=QQ,REGB=4,COND=T
1077+           ST      10,SAVE10
1078+           ST      11,SAVE11
1079+           LA      10,0
1080+           A       10,=F'7'
1081+           LR      11,4
1082+           CR      10,11
1083                    *,ILLEGAL OPERATOR SPECIFIED FOR A LEAVE CONDITION
1084+           B       A5
1085+           L       10,SAVE10
1086+           L       11,SAVE11
1087+           B       AZ
1088+A5         L       10,SAVE10
1089+           L       11,SAVE11
1090                            A       6,=F'1'
1091                            A       7,=F'1'
1092            ENDDO           LABEL=Z
1093+           B       Z
1094+CZ         L       8,SAVE8
1095+           L       9,SAVE9
1096+BZ         L       10,SAVE10
1097+           L       11,SAVE11
1098+AZ         EQU     *
1099 *
1100 *
1101                  SNAP      ID=2,DCB=DUMPAREA,PDATA=REGS
1102+           CNOP    0,4
1103+           BAL     1,IHB0007 BRANCH AROUND PARAM LIST
1104+           DC      AL1(2) ID NUMBER
1105+           DC      AL1(0)
1106+           DC      AL1(130) OPTION FLAGS
1107+           DC      AL1(32) OPTION FLAGS
1108+           DC      A(DUMPAREA) DCB ADDRESS
1109+           DC      A(0) TCB ADDRESS
1110+           DC      A(0) ADDRESS OF SNAP-SHOT LIST
1111+IHB0007    DS      0H
1112+           SVC     51
1113 *
1114 *
1115            EXIT
1116+           L       13,4(,13) POP UP SAVE AREA
1117+           LM      14,12,12(13) RESTORE REGISTERS
1118+           MVI     12(13),X'FF' FLAG EXIT
1119+           BR      14 RETURN
1120            CLOSE      DUMPAREA
1121+           CNOP    0,4 ALIGN LIST TO FULLWORD
1122+           BAL     1,*+8 LOAD REG1 W/LIST ADDR
1123+           DC      AL1(128) OPTION BYTE
1124+           DC      AL3(DUMPAREA) DCB ADDRESS
1125+           SVC     20 ISSUE CLOSE SVC
1126 SAVEAREA   DC          18A(0)
1127            DS          0F
1128 ZERO       DC          1F'0'
1129 SAVE8      DS          1F
1130 SAVE9      DS          1F
1131 SAVE10     DS          1F
```

```
1132 SAVE11    DS         1F
1133 POINTER   DC         1F'0'
1134 STACK     DS         100F
1135 DUMPAREA DCB         DDNAME=TEAM,DSORG=PS,RECFM=VBA,                      X
                       MACRF=W,BLKSIZE=882,LRECL=125

1137+*                           DATA CONTROL BLOCK
1138+*
1139+DUMPAREA DC       OF'0' ORIGIN ON WORD BOUNDARY

1141+*                           DIRECT ACCESS DEVICE INTERFACE

1143+           DC     BL16'0' FDAD,DVTBL
1144+           DC     A(0) KEYLE,DEVT,TRBAL

1146+*                           COMMON ACCESS METHOD INTERFACE

1148+           DC     AL1(0) BUFNO
1149+           DC     AL3(1) BUFCB
1150+           DC     AL2(0) BUFL
1151+           DC     BL2'0100000000000000' DSORG
1152+           DC     A(1) IOBAD

1154+*                           FOUNDATION EXTENSION

1156+           DC     BL1'00000000' BFTEK,BFLN,HIARCHY
1157+           DC     AL3(1) EODAD
1158+           DC     BL1'01010100' RECFM
1159+           DC     AL3(0) EXLST

1161+*                           FOUNDATION BLOCK

1163+           DC     CL8'TEAM' DDNAME
1164+           DC     BL1'00000010' OFLGS
1165+           DC     BL1'00000000' IFLG
1166+           DC     BL2'0000000000100000' MACR

1168+*                           BSAM-BPAM-QSAM INTERFACE

1170+           DC     BL1'00000000' RER1
1171+           DC     AL3(1) CHECK, GERR, PERR
1172+           DC     A(1) SYNAD
1173+           DC     H'0' CIND1, CIND2
1174+           DC     AL2(882) BLKSIZE
1175+           DC     F'0' WCPO, WCPL, OFFSR, OFFSW
1176+           DC     A(1) IOBA
1177+           DC     AL1(0) NCP
1178+           DC     AL3(1) EOBR, EOBAD

1180+*                             BSAM-BPAM INTERFACE

1182+           DC     A(1) EOBW
1183+           DC     H'0' DIRCT
1184+           DC     AL2(125) LRECL
1185+           DC     A(1) CNTRL, NOTE, POINT
1186            END
1187                   =F'1'
1188                   =F'100'
1189                   =F'4'
1190                   =F'7'
```

APPENDIX C

THE CASE, ENDCASE, ELSE, AND ENDELSE MACROS

The Source Listing for the CASE Macro

```
STMT   SOURCE STATEMENT

 709 *                         THE CASE MACRO PROVIDES A SELECTIVE CONTROL STRUC-
 710 *                         TURE FOR ALC.  A CONDITION IS SPECIFIED WHEN CODING
 711 *                         THE MACRO, AND DURING PROGRAM EXECUTION, IF THE
 712 *                         CONDITION IS TRUE, PROGRAM INSTRUCTIONS FOLLOWING
 713 *                         THE CASE STATEMENT (UP TO THE 'ENDCASE' MACRO) WILL
 714 *                         BE EXECUTED.  IF THE CONDITION IS FALSE, AN IMMED-
 715 *                         IATE BRANCH TO THE 'ENDCASE' MACRO WILL OCCUR.
 716 *
 717 *
 718             MACRO
 719             CASE    &LABEL=,&LITA=,&REGA=,&LOCA=,&OPRATOR=,&LITB=,&REGB=,    X
                         &LOCB=
 720             LCLA          &LAB
 721 &LAB        SETA          &SYSNDX
 722             AGO           .SKIP1
 723 *
 724 *
 725 *                         TO TEST THE CASE CONDITION REGISTERS 10 AND 11 ARE
 726 *                         USED AS WORK AREAS; CONSEQUENTLY, THE CONTENTS OF
 727 *                         THESE REGISTERS ARE INITIALLY SAVED.
 728 *
 729 *
 730 .SKIP1      ST            10,SAVE10
 731             ST            11,SAVE11
 732             AGO           .SKIP2
 733 *
 734 *
 735 *                         THE LEFT OPERAND OF THE CASE CONDITION IS DETER-
 736 *                         MINED IN THE FOLLOWING SECTION OF CODE.  IT MAY BE
 737 *                         SPECIFIED BY A LITERAL (LITA), BY REGISTER (REGA) OR
 738 *                         BY LOCATION (LOCA).  ITS VALUE IS LOADED INTO RE-
 739 *                         ISTER 10.  IF THE LEFT OPERAND HAS BEEN ILLEGALLY
 740 *                         SPECIFIED, AN ERROR MESSAGE WILL BE OUTPUT, AND THE
 741 *                         CASE CONDITION WILL BE ASSUMED FALSE.
 742 *
 743 *
 744 .SKIP2      AIF           ('&LITA' EQ '').A
 745             AIF           ('&REGA' NE '').ERR1
 746             AIF           ('&LOCA' NE '').ERR1
 747             AIF           ('&LITB' NE '').ERR2
 748 .RETURN     LA            10,0
 749             A             10,=F'&LITA'
 750             AGO           .C
 751 .A          AIF           ('&REGA' EQ '').B
 752             AIF           ('&LOCA' NE '').ERR1
 753             AIF           (T'&REGA NE 'N').ERR3
 754             AIF           ('&REGA' GT '15').ERR4
 755             LR            10,&REGA
 756             AGO           .C
```

```
.B       AIF      ('&LOCA' EQ '').ERR5
         L        10,&LOCA
         AGO      .C
*
*
*                 THE NEXT SECTION OF CODE DETERMINES THE RIGHT OPER-
*                 AND, AND LOADS IT INTO REGISTER 11.  IT MAY BE SPEC-
*                 IFIED IN THE SAME MANNER AS THE LEFT OPERAND WITH
*                 THE RESPECTIVE PARAMETERS BEING LITB, REGB AND LOCB.
*                 A MISSING OR ILLEGALLY SPECIFIED PARAMETER WILL
*                 CAUSE THE GENERATION OF AN ERROR MESSAGE, AND THE
*                 CASE CONDITION WILL BE ASSUMED FALSE.
*
*
.C       AIF      ('&LITB' EQ '').D
         AIF      ('&REGB' NE '').ERR6
         AIF      ('&LOCB' NE '').ERR6
         LA       11,0
         A        11,=F'&LITB'
         AGO      .F
.D       AIF      ('&REGB' EQ '').E
         AIF      ('&LOCB' NE '').ERR6
         AIF      (T'&REGB NE 'N').ERR7
         AIF      ('&REGB' GT '15').ERR8
         LR       11,&REGB
         AGO      .F
.E       AIF      ('&LOCB' EQ '').ERR6
         L        11,&LOCB
         AGO      .F
*
*
*                 THE NEXT SECTION OF CODE COMPARES THE LEFT AND RIGHT
*                 OPERANDS AND GENERATES AN APPROPRIATE BRANCH IN-
*                 STRUCTION BASED ON THE VALUE OF THE SPECIFIED LOG-
*                 ICAL OPERATOR.  IF THE OPERATOR IS ILLEGAL OR MISS-
*                 ING FROM THE PARAMETER LIST, AN ERROR MESSAGE WILL
*                 BE OUTPUT, AND THE CASE CONDITION WILL BE ASSUMED
*                 FALSE.
*
*
.F       CR       10,11
         AIF      ('&OPRATOR' EQ '').ERR9
         AIF      ('&OPRATOR' EQ 'LT').LESS
         AIF      ('&OPRATOR' EQ 'LE').LESSEQ
         AIF      ('&OPRATOR' EQ 'EQ').EQUAL
         AIF      ('&OPRATOR' EQ 'GE').GRTEQ
         AIF      ('&OPRATOR' EQ 'GT').GRTER
         AIF      ('&OPRATOR' EQ 'NE').NOTEQ
         AGO      .ERR10
.EQUAL   BNE      B&LAB
         B        C&LAB
         AGO      .END
.LESS    BNL      B&LAB
         B        C&LAB
         AGO      .END
.LESSEQ  BH       B&LAB
         B        C&LAB
         AGO      .END
.GRTEQ   BL       B&LAB
         B        C&LAB
         AGO      .END
.GRTER   BNH      B&LAB
         B        C&LAB
         AGO      .END
.NOTEQ   BE       B&LAB
         B        C&LAB
```

```
              AGO       .END
*
*
*                       THE FOLLOWING SET OF INSTRUCTIONS RESTORES THE ORIG-
*                       INAL CONTENTS OF THE WORKING REGISTERS AND CAUSES
*                       EITHER THE EXECUTION OF PROGRAM INSTRUCTIONS (IF
*                       THE CASE CONDITION IS TRUE) OR A BRANCH TO THE 'END-
*                       CASE' MACRO (IF THE CASE CONDITION IS FALSE).
*
*
.END          ANOP
B&LAB         L         10,SAVE10
              L         11,SAVE11
              B         &LABEL
C&LAB         L         10,SAVE10
              L         11,SAVE11
              AGO       .STOP
*
*
*                       BELOW ARE THE ERROR CONDITIONS WHICH THE MACRO MAY
*                       DETECT.  WHENEVER AN ERROR CONDITION IS RAISED, A
*                       MESSAGE IS PRINTED AND THE CASE CONDITION IS SET TO
*                       FALSE.
*
*
.ERR1         MNOTE     *,'MORE THAN ONE LEFT OPERAND SPECIFIED FOR CASE'
              MNOTE     *,'CONDITION'
              AGO       .END
.ERR2         MNOTE     *,'MORE THAN ONE LITERAL SPECIFIED IN CONDITION'
              MNOTE     *,'STATEMENT'
              AGO       .RETURN
.ERR3         MNOTE     *,'''REGA'' SPECIFIED IS NOT A SELF-DEFINING '
              MNOTE     *,'NUMERIC'
              AGO       .END
.ERR4         MNOTE     *,'''REGA'' DOES NOT SPECIFY A VALID REGISTER'
              MNOTE     *,'NUMBER'
              AGO       .END
.ERR5         MNOTE     *,'NO LEFT OPERAND HAS BEEN SPECIFIED FOR CASE'
              MNOTE     *,'CONDITION'
              AGO       .END
.ERR6         MNOTE     *,'MORE THAN ONE RIGHT OPERAND SPECIFIED FOR CASE'
              MNOTE     *,'CONDITION'
              AGO       .END
.ERR7         MNOTE     *,'''REGB'' SPECIFIED IS NOT A SELF-DEFINING'
              MNOTE     *,'NUMERIC'
              AGO       .END
.ERR8         MNOTE     *,'''REGB'' DOES NOT SPECIFY A VALID REGISTER'
              MNOTE     *,'NUMBER'
              AGO       .END
.ERR9         MNOTE     *,'OPERATOR FOR CASE CONDITION HAS NOT BEEN'
              MNOTE     *,'SPECIFIED'
              AGO       .END
.ERR10        MNOTE     *,'ILLEGAL OPERATOR SPECIFIED FOR A CASE CONDITION'
              AGO       .END
.STOP         MEND
```

The Source Listing for the ENDCASE Macro

```
STMT   SOURCE STATEMENT

*                       THE ENDCASE MACRO PROVIDES THE BRANCHING LOCATION
*                       FOR A CASE MACRO WHOSE CONDITION IS FALSE.  IT ALSO
```

```
*                         MARKS THE END OF THE PROGRAM INSTRUCTIONS TO BE IN-
*                         CLUDED WITHIN THE CASE STATEMENT.  THE ENDCASE MACRO
*                         MAY BE SPECIFIED IN ONE OF TWO MODES.  THE FIRST
*                         MODE (OPTION=1) WILL CAUSE A BRANCH TO A SPECIFIED
*                         LOCATION 'LAB' IF THE CASE CONDITION WAS TRUE (THIS
*                         WILL PERMIT THE PROGRAMMER TO BYPASS ADDITIONAL CASE
*                         STATEMENTS WHICH FOLLOW IN LINE).  THE SECOND MODE
*                         (OPTION=2) CAUSES PROGRAM CONTROL TO BE PASSED TO
*                         THE NEXT INSTRUCTION (THIS WILL PERMIT THE TESTING
*                         OF SUCCESSIVE CASE STATEMENTS).  'LABEL' SHOULD BE
*                         SPECIFIED WITH THE SAME VALUE AS THE 'LABEL' PARAM-
*                         ETER IN THE CORRESPONDING CASE MACRO.
*
*
              MACRO
              ENDCASE     &OPTION=,&LABEL=,&LAB=
              AGO         .SKIP1
*
*
*                         THE FOLLOWING STATEMENTS DETERMINE THE OPTION TO
*                         USE.  IF THE PARAMETER HAS NOT BEEN SPECIFIED OR IF
*                         IT HAS BEEN INCORRECTLY SPECIFIED, A MESSAGE IS
*                         PRINTED AND OPTION=1 IS ASSUMED.
*
*
.SKIP1        AIF         ('&OPTION' EQ '1').OPTION1
              AIF         ('&OPTION' EQ '2').OPTION2
              AIF         ('&OPTION' EQ '').ERR1
              AIF         (T'&OPTION NE 'N').ERR2
              AGO         .ERR3
*
*
*                         BELOW ARE THE ERROR MESSAGES GENERATED BY THE MACRO.
*                         EACH ONE REFERS TO THE 'OPTION' PARAMETER AND CAUSES
*                         IT TO BE SET TO ONE.
*
*
.ERR1         MNOTE       *,'''OPTION'' NOT SPECIFIED FOR ENDCASE (OPTION'
              MNOTE       *,'SET TO ONE)'
              AGO         .OPTION1
.ERR2         MNOTE       *,'''OPTION'' SPECIFIED IS NOT A SELF-DEFINING'
              MNOTE       *,'NUMERIC (OPTION SET TO ONE)'
              AGO         .OPTION1
.ERR3         MNOTE       *,'''OPTION'' SPECIFIED IS AN ILLEGAL NUMERIC VALUE'
              MNOTE       *,'(OPTION SET TO ONE)'
              AGO         .OPTION1
*
*
*                         THE FOLLOWING STATEMENTS INCLUDE THE BRANCH AND/OR
*                         'LABEL' INSTRUCTIONS GENERATED BY THE MACRO.
*
*
.OPTION1      B           &LAB
.OPTION2      ANOP
&LABEL        EQU         *
              MEND
```

The Source Listing for the ELSE Macro

```
STMT   SOURCE STATEMENT

*                         THE ELSE MACRO IS MERELY A MACRO DEFINITION NAME TO
*                         DENOTE THE BEGINNING OF THE ALTERNATIVE SET OF IN-
```

```
 940 *                      STRUCTIONS TO BE EXECUTED WHEN ALL OF THE PROCEED-
 941 *                      ING CASE STATEMENTS HAVE FAILED.  AN 'ENDELSE' MACRO
 942 *                      SHOULD ALWAYS BE SPECIFIED WHEN THE ELSE MACRO IS
 943 *                      USED.
 944 *
 945 *
 946            MACRO
 947            ELSE
 948            MEND
```

The Source Listing for the ENDELSE Macro

```
STMT   SOURCE STATEMENT

 950 *                      THE ENDELSE MACRO GENERATES A LABEL TO WHICH A SET
 951 *                      OF CASE STATEMENTS WILL BRANCH WHEN ANY OR ALL HAVE
 952 *                      HAD 'TRUE' CONDITIONS.  IT ALSO MARKS THE END OF
 953 *                      THE SET OF INSTRUCTIONS ENCOMPASSED BY THE ELSE
 954 *                      MACRO.
 955 *
 956 *
 957            MACRO
 958            ENDELSE  &LAB=
 959 &LAB       EQU      *
 960            MEND
```

Sample Program I Illustrating the CASE, ENDCASE, ELSE, and ENDELSE Macros

```
STMT   SOURCE STATEMENT

 962            PRINT    NOGEN
 963 FRISBEE    ENTER    12,SAVEAREA
 979 START      OPEN     (DUMPAREA,OUTPUT)
 985            LA       2,0
 986            LA       2,0
 987            LA       3,0
 988            LA       4,0
 989            LA       5,0
 990            LA       6,0
 991            LA       7,0
 992            LA       8,0
 993            LA       9,0
 994            LA       10,0
 995            LA       11,0
 996            LA       14,0
 997            LA       15,0
 998 *
 999 *
1000            SNAP     ID=1,DCB=DUMPAREA,PDATA=REGS
1012 *
1013 *
1014            CASE     LABEL=A,LITA=0,OPRATOR=NE,REGB=15
1028                     A     4,=F'1'
1029                     A     5,=F'1'
1030                     A     6,=F'1'
1031                     A     7,=F'1'
1032                     A     8,=F'1'
1033                     A     9,=F'1'
```

```
1034            ENDCASE     OPTION=1,LABEL=A,LAB=HERE
1037 *
1038 *
1039            CASE        LABEL=B,REGA=14,OPRATOR=EQ,LITB=0
1053                        A     4,=F'2'
1054                        A     5,=F'2'
1055                        A     6,=F'2'
1056                        A     7,=F'2'
1057                        A     8,=F'2'
1058                        A     9,=F'2'
1059            ENDCASE     OPTION=1,LABEL=B,LAB=HERE
1062 *
1063 *
1064            CASE        LOCA=X,LOCB=Y,LABEL=C,OPRATOR=LT
1077                        A     4,=F'3'
1078                        A     5,=F'3'
1079                        A     6,=F'3'
1080                        A     7,=F'3'
1081                        A     8,=F'3'
1082                        A     9,=F'3'
1083            ENDCASE     OPTION=1,LABEL=C,LAB=HERE
1086 *
1087 *
1088            ELSE
1089                        A     4,=F'4'
1090                        A     5,=F'4'
1091                        A     6,=F'4'
1092                        A     7,=F'4'
1093                        A     8,=F'4'
1094                        A     9,=F'4'
1095            ENDELSE     LAB=HERE
1097 *
1098 *
1099            SNAP        ID=2,DCB=DUMPAREA,PDATA=REGS
1111 *
1112 *
1113            EXIT
1118            CLOSE       DUMPAREA

1124 SAVEAREA DC          18A(0)
1125          DS          0F
1126 ZERO     DC          1F'0'
1127 SAVE8    DS          1F
1128 SAVE9    DS          1F
1129 SAVE10   DS          1F
1130 SAVE11   DS          1F
1131 POINTER  DC          1F'0'
1132 STACK    DS          100F
1133 X        DC          1F'4'
1134 Y        DC          1F'8'
1135 DUMPAREA DCB           DDNAME=TEAM,DSORG=PS,RECFM=VBA,                X
                       MACRF=W,BLKSIZE=882,LRECL=125
1186          END
1187                   =F'0'
1188                   =F'1'
1189                   =F'2'
1190                   =F'3'
1191                   =F'4'
```

```
REGS AT ENTRY TO SNAP                                          ID = 001


REGS 0-7          00000030   9001888C   00000000   00000000
                  00000000   00000000   00000000   00000000

REGS 8-15         00000000   00000000   00000000   00000000
                  50018820   0001B9C8   00000000   00000000
END OF SNAP
```

```
REGS AT ENTRY TO SNAP                                          ID = 002


REGS 0-7          000002A0   A001899C   00000000   00000000
                  00000002   00000002   00000002   00000002

REGS 8-15         00000002   00000002   00000000   00000000
                  50018820   0001B9C8   00000000   00000000
END OF SNAP
```

Program I with Expanded Macros

```
STMT    SOURCE STATEMENT

 962 FRISBEE  ENTER     12,SAVEAREA
 963+FRISBEE  DS    0H
 964+         ENTRY FRISBEE DECLARE NAME ENTRY
 965+         USING *,12 DECLARE BASE ADDRESSIBILITY.
 966+         BALR  15,0 (INITIAL ADDRESSIBILITY).
 967+         B     12(,15) BRANCH AROUND ID FIELD
 968+         DC    AL1(7),CL7'FRISBEE' ID LENGTH AND ID
 969+         BCTR  15,0 (RESET INITIAL ADDRESSIBILITY
 970+         BCTR  15,0 ABSOLUTE ENTRY POINT).
 971+         STM   14,12,12(13) SAVE REGISTERS
 972+         LR    12,15 SETUP BASE REGISTER.
 973+         ST    13,SAVEAREA+4 CHAIN BACK
 974+         LA    0,SAVEAREA CHAIN FORWARD
 975+         ST    0,8(0,13)
 976+         LR    13,0 SET UP SAVE AREA POINTER
 977+         USING SAVEAREA,13 AND ADDRESSABILITY
 978 START    OPEN      (DUMPAREA,OUTPUT)
 979+         CNOP  0,4 ALIGN LIST TO FULLWORD
 980+START    BAL   1,*+8 LOAD REG1 W/LIST ADDR.
 981+         DC    AL1(143) OPTION BYTE
 982+         DC    AL3(DUMPAREA) DCB ADDRESS
 983+         SVC   19 ISSUE OPEN SVC
 984          LA        2,0
 985          LA        2,0
 986          LA        3,0
```

```
         LA       4,0
         LA       5,0
         LA       6,0
         LA       7,0
         LA       8,0
         LA       9,0
         LA       10,0
         LA       11,0
         LA       14,0
         LA       15,0
 *
 *
         SNAP         ID=1,DCB=DUMPAREA,PDATA=REGS
+        CNOP   0,4
+        BAL    1,IHB0003 BRANCH AROUND PARAM LIST
+        DC     AL1(1) ID NUMBER
+        DC     AL1(0)
+        DC     AL1(130) OPTION FLAGS
+        DC     AL1(32) OPTION FLAGS
+        DC     A(DUMPAREA) DCB ADDRESS
+        DC     A(0) TCB ADDRESS
+        DC     A(0) ADDRESS OF SNAP-SHOT LIST
+IHB0003 DS     0H
+        SVC    51
 *
 *
         CASE         LABEL=A,LITA=0,OPRATOR=NE,REGB=15
+        ST     10,SAVE10
+        ST     11,SAVE11
+        LA     10,0
+        A      10,=F'0'
+        LR     11,15
+        CR     10,11
+        BE     B4
+        B      C4
+B4      L      10,SAVE10
+        L      11,SAVE11
+        B      A
+C4      L      10,SAVE10
+        L      11,SAVE11
                      A      4,=F'1'
                      A      5,=F'1'
                      A      6,=F'1'
                      A      7,=F'1'
                      A      8,=F'1'
                      A      9,=F'1'
         ENDCASE      OPTION=1,LABEL=A,LAB=HERE
+        B       HERE
+A       EQU     *
 *
 *
         CASE         LABEL=B,REGA=14,OPRATOR=EQ,LITB=0
+        ST     10,SAVE10
+        ST     11,SAVE11
+        LR     10,14
+        LA     11,0
+        A      11,=F'0'
+        CR     10,11
+        BNE    B6
+        B      C6
+B6      L      10,SAVE10
+        L      11,SAVE11
+        B      B
+C6      L      10,SAVE10
+        L      11,SAVE11
                      A      4,=F'2'
                      A      5,=F'2'
```

```
                     A     6,=F'2'
                     A     7,=F'2'
                     A     8,=F'2'
                     A     9,=F'2'
         ENDCASE     OPTION=1,LABEL=B,LAB=HERE
+        B      HERE
+B       EQU    *
 *
 *
         CASE        LOCA=X,LOCB=Y,LABEL=C,OPRATOR=LT
+        ST     10,SAVE10
+        ST     11,SAVE11
+        L      10,X
+        L      11,Y
+        CR     10,11
+        BNL    B8
+        B      C8
+B8      L      10,SAVE10
+        L      11,SAVE11
+        B      C
+C8      L      10,SAVE10
+        L      11,SAVE11
                     A     4,=F'3'
                     A     5,=F'3'
                     A     6,=F'3'
                     A     7,=F'3'
                     A     8,=F'3'
                     A     9,=F'3'
         ENDCASE     OPTION=1,LABEL=C,LAB=HERE
+        B      HERE
+C       EQU    *
 *
 *
         ELSE
                     A     4,=F'4'
                     A     5,=F'4'
                     A     6,=F'4'
                     A     7,=F'4'
                     A     8,=F'4'
                     A     9,=F'4'
         ENDELSE     LAB=HERE
+HERE    EQU    *
 *
 *
         SNAP        ID=2,DCB=DUMPAREA,PDATA=REGS
+        CNOP   0,4
+        BAL    1,IHB0012 BRANCH AROUND PARAM LIST
+        DC     AL1(2) ID NUMBER
+        DC     AL1(0)
+        DC     AL1(130) OPTION FLAGS
+        DC     AL1(32) OPTION FLAGS
+        DC     A(DUMPAREA) DCB ADDRESS
+        DC     A(0) TCB ADDRESS
+        DC     A(0) ADDRESS OF SNAP-SHOT LIST
+IHB0012 DS     0H
+        SVC    51
 *
 *
         EXIT
+        L      13,4(,13) POP UP SAVE AREA
+        LM     14,12,12(13) RESTORE REGISTERS
+        MVI    12(13),X'FF' FLAG EXIT
+        BR     14 RETURN
         CLOSE      DUMPAREA
+        CNOP   0,4 ALIGN LIST TO FULLWORD
+        BAL    1,*+8 LOAD REG1 W/LIST ADDR
+        DC     AL1(128) OPTION BYTE
```

```
1121+           DC      AL3(DUMPAREA) DCB ADDRESS
1122+           SVC     20 ISSUE CLOSE SVC

1123 SAVEAREA DC            18A(0)
1124            DS          0F
1125 ZERO       DC          1F'0'
1126 SAVE8      DS          1F
1127 SAVE9      DS          1F
1128 SAVE10     DS          1F
1129 SAVE11     DS          1F
1130 POINTER    DC          1F'0'
1131 STACK      DS          100F
1132 X          DC          1F'4'
1133 Y          DC          1F'8'
1134 DUMPAREA DCB           DDNAME=TEAM,DSORG=PS,RECFM=VBA,                X
                        MACRF=W,BLKSIZE=882,LRECL=125


1136+*                                DATA CONTROL BLOCK
1137+*
1138+DUMPAREA DC       0F'0' ORIGIN ON WORD BOUNDARY

1140+*                                DIRECT ACCESS DEVICE INTERFACE

1142+           DC      BL16'0' FDAD,DVTBL
1143+           DC      A(0) KEYLE,DEVT,TRBAL

1145+*                                COMMON ACCESS METHOD INTERFACE

1147+           DC      AL1(0) BUFNO
1148+           DC      AL3(1) BUFCB
1149+           DC      AL2(0) BUFL
1150+           DC      BL2'0100000000000000' DSORG
1151+           DC      A(1) IOBAD

1153+*                                FOUNDATION EXTENSION

1155+           DC      BL1'00000000' BFTEK,BFLN,HIARCHY
1156+           DC      AL3(1) EODAD
1157+           DC      BL1'01010100' RECFM
1158+           DC      AL3(0) EXLST

1160+*                                FOUNDATION BLOCK

1162+           DC      CL8'TEAM' DDNAME
1163+           DC      BL1'00000010' OFLGS
1164+           DC      BL1'00000000' IFLG
1165+           DC      BL2'0000000000100000' MACR

1167+*                                BSAM-BPAM-QSAM INTERFACE

1169+           DC      BL1'00000000' RER1
1170+           DC      AL3(1) CHECK, GERR, PERR
1171+           DC      A(1) SYNAD
1172+           DC      H'0' CIND1, CIND2
1173+           DC      AL2(882) BLKSIZE
1174+           DC      F'0' WCPO, WCPL, OFFSR, OFFSW
1175+           DC      A(1) IOBA
1176+           DC      AL1(0) NCP
1177+           DC      AL3(1) EOBR, EOBAD

1179+*                                BSAM-BPAM INTERFACE

1181+           DC      A(1) EOBW
1182+           DC      H'0' DIRCT
1183+           DC      AL2(125) LRECL
1184+           DC      A(1) CNTRL, NOTE, POINT
```

```
1185          END
1186                =F'0'
1187                =F'1'
1188                =F'2'
1189                =F'3'
1190                =F'4'
```

```
STMT   SOURCE STATEMENT

 962 FRISBEE  ENTER      12,SAVEAREA
 963+FRISBEE  DS    0H
 964+         ENTRY FRISBEE DECLARE NAME ENTRY
 965+         USING *,12 DECLARE BASE ADDRESSIBILITY.
 966+         BALR  15,0 (INITIAL ADDRESSIBILITY).
 967+         B     12(,15) BRANCH AROUND ID FIELD
 968+         DC    AL1(7),CL7'FRISBEE' ID LENGTH AND ID
 969+         BCTR  15,0 (RESET INITIAL ADDRESSIBILITY
 970+         BCTR  15,0 ABSOLUTE ENTRY POINT).
 971+         STM   14,12,12(13) SAVE REGISTERS
 972+         LR    12,15 SETUP BASE REGISTER.
 973+         ST    13,SAVEAREA+4 CHAIN BACK
 974+         LA    0,SAVEAREA CHAIN FORWARD
 975+         ST    0,8(0,13)
 976+         LR    13,0 SET UP SAVE AREA POINTER
 977+         USING SAVEAREA,13 AND ADDRESSABILITY
 978 START    OPEN       (DUMPAREA,OUTPUT)
 979+         CNOP  0,4 ALIGN LIST TO FULLWORD
 980+START    BAL   1,*+8 LOAD REG1 W/LIST ADDR.
 981+         DC    AL1(143) OPTION BYTE
 982+         DC    AL3(DUMPAREA) DCB ADDRESS
 983+         SVC   19 ISSUE OPEN SVC
 984          LA        2,0
 985          LA        3,0
 986          LA        4,0
 987          LA        5,0
 988          LA        6,0
 989          LA        7,0
 990          LA        8,0
 991          LA        9,0
 992          LA        10,0
 993          LA        11,0
 994          LA        14,0
 995          LA        15,0
 996 *
 997 *
 998              SNAP     ID=1,DCB=DUMPAREA,PDATA=REGS
 999+         CNOP  0,4
1000+         BAL   1,IHB0003 BRANCH AROUND PARAM LIST
1001+         DC    AL1(1) ID NUMBER
1002+         DC    AL1(0)
1003+         DC    AL1(130) OPTION FLAGS
1004+         DC    AL1(32) OPTION FLAGS
1005+         DC    A(DUMPAREA) DCB ADDRESS
1006+         DC    A(0) TCB ADDRESS
1007+         DC    A(0) ADDRESS OF SNAP-SHOT LIST
1008+IHB0003  DS    0H
1009+         SVC   51
1010 *
1011 *
1012          CASE      LABEL=A,LITA=0,OPRATOR=NE,REGB=15
```

```
+         ST     10,SAVE10
+         ST     11,SAVE11
+         LA     10,0
+         A      10,=F'0'
+         LR     11,15
+         CR     10,11
+         BE     B4
+         B      C4
+B4       L      10,SAVE10
+         L      11,SAVE11
+         B      A
+C4       L      10,SAVE10
+         L      11,SAVE11
                       A      4,=F'1'
                       A      5,=F'1'
                       A      6,=F'1'
                       A      7,=F'1'
                       A      8,=F'1'
                       A      9,=F'1'
          ENDCASE    OPTION=4,LABEL=A,LAB=HERE
                 *,'OPTION' SPECIFIED IS AN ILLEGAL NUMERIC VALUE
                 *,(OPTION SET TO ONE)
+         B      HERE
+A        EQU    *
 *
 *
          CASE       LABEL=B,REGA=14,OPRATOR=EQ,LITB=0
+         ST     10,SAVE10
+         ST     11,SAVE11
+         LR     10,14
+         LA     11,0
+         A      11,=F'0'
+         CR     10,11
+         BNE    B6
+         B      C6
+B6       L      10,SAVE10
+         L      11,SAVE11
+         B      B
+C6       L      10,SAVE10
+         L      11,SAVE11
                       A      4,=F'2'
                       A      5,=F'2'
                       A      6,=F'2'
                       A      7,=F'2'
                       A      8,=F'2'
                       A      9,=F'2'
          ENDCASE    OPTION=1,LABEL=B,LAB=HERE
+         B      HERE
+B        EQU    *
 *
 *
          CASE       LOCA=X,LOCB=Y,LABEL=C,OPRATOR=LT
+         ST     10,SAVE10
+         ST     11,SAVE11
+         L      10,X
+         L      11,Y
+         CR     10,11
+         BNL    B8
+         B      C8
+B8       L      10,SAVE10
+         L      11,SAVE11
+         B      C
+C8       L      10,SAVE10
+         L      11,SAVE11
                       A      4,=F'3'
                       A      5,=F'3'
```

```
1079                    A      6,=F'3'
1080                    A      7,=F'3'
1081                    A      8,=F'3'
1082                    A      9,=F'3'
1083           ENDCASE    OPTION=1,LABEL=C,LAB=HERE
1084+          B      HERE
1085+C         EQU    *
1086 *
1087 *
1088           ELSE
1089                    A      4,=F'4'
1090                    A      5,=F'4'
1091                    A      6,=F'4'
1092                    A      7,=F'4'
1093                    A      8,=F'4'
1094                    A      9,=F'4'
1095           ENDELSE    LAB=HERE
1096+HERE      EQU    *
1097 *
1098 *
1099 *
1100 *
1101                SNAP      ID=2,DCB=DUMPAREA,PDATA=REGS
1102+          CNOP   0,4
1103+          BAL    1,IHB0012 BRANCH AROUND PARAM LIST
1104+          DC     AL1(2) ID NUMBER
1105+          DC     AL1(0)
1106+          DC     AL1(130) OPTION FLAGS
1107+          DC     AL1(32) OPTION FLAGS
1108+          DC     A(DUMPAREA) DCB ADDRESS
1109+          DC     A(0) TCB ADDRESS
1110+          DC     A(0) ADDRESS OF SNAP-SHOT LIST
1111+IHB0012   DS     0H
1112+          SVC    51
1113 *
1114 *
1115           EXIT
1116+          L      13,4(,13) POP UP SAVE AREA
1117+          LM     14,12,12(13) RESTORE REGISTERS
1118+          MVI    12(13),X'FF' FLAG EXIT
1119+          BR     14 RETURN
1120           CLOSE      DUMPAREA
1121+          CNOP   0,4 ALIGN LIST TO FULLWORD
1122+          BAL    1,*+8 LOAD REG1 W/LIST ADDR
1123+          DC     AL1(128) OPTION BYTE
1124+          DC     AL3(DUMPAREA) DCB ADDRESS
1125+          SVC    20 ISSUE CLOSE SVC
1126 SAVEAREA DC         18A(0)
1127           DS         0F
1128 ZERO      DC         1F'0'
1129 SAVE8     DS         1F
1130 SAVE9     DS         1F
1131 SAVE10    DS         1F
1132 SAVE11    DS         1F
1133 POINTER   DC         1F'0'
1134 STACK     DS         100F
1135 X         DC         1F'4'
1136 Y         DC         1F'8'
1137 DUMPAREA DCB        DDNAME=TEAM,DSORG=PS,RECFM=VBA,                  X
                      MACRF=W,BLKSIZE=882,LRECL=125

1139+*                           DATA CONTROL BLOCK
1140+*
1141+DUMPAREA DC      0F'0' ORIGIN ON WORD BOUNDARY

1143+*                           DIRECT ACCESS DEVICE INTERFACE
```

```
1145+          DC     BL16'0' FDAD,DVTBL
1146+          DC     A(0) KEYLE,DEVT,TRBAL

1148+*                         COMMON ACCESS METHOD INTERFACE

1150+          DC     AL1(0) BUFNO
1151+          DC     AL3(1) BUFCB
1152+          DC     AL2(0) BUFL
1153+          DC     BL2'0100000000000000' DSORG
1154+          DC     A(1) IOBAD

1156+*                         FOUNDATION EXTENSION

1158+          DC     BL1'00000000' BFTEK,BFLN,HIARCHY
1159+          DC     AL3(1) EODAD
1160+          DC     BL1'01010100' RECFM
1161+          DC     AL3(0) EXLST

1163+*                         FOUNDATION BLOCK

1165+          DC     CL8'TEAM' DDNAME
1166+          DC     BL1'00000010' OFLGS
1167+          DC     BL1'00000000' IFLG
1168+          DC     BL2'0000000000100000' MACR

1170+*                         BSAM-BPAM-QSAM INTERFACE

1172+          DC     BL1'00000000' RERI
1173+          DC     AL3(1) CHECK, GERR, PERR
1174+          DC     A(1) SYNAD
1175+          DC     H'0' CIND1, CIND2
1176+          DC     AL2(882) BLKSIZE
1177+          DC     F'0' WCPO, WCPL, OFFSR, OFFSW
1178+          DC     A(1) IOBA
1179+          DC     AL1(0) NCP
1180+          DC     AL3(1) EOBR, EOBAD

1182+*                          BSAM-BPAM INTERFACE

1184+          DC     A(1) EOBW
1185+          DC     H'0' DIRCT
1186+          DC     AL2(125) LRECL
1187+          DC     A(1) CNTRL, NOTE, POINT
1188           END
1189                  =F'0'
1190                  =F'1'
1191                  =F'2'
1192                  =F'3'
1193                  =F'4'
```

Sample Program K Illustrating a CASE Macro Coded Incorrectly

```
STMT   SOURCE STATEMENT

 962 FRISBEE  ENTER      12,SAVEAREA
 963+FRISBEE  DS     0H
 964+         ENTRY FRISBEE DECLARE NAME ENTRY
 965+         USING *,12 DECLARE BASE ADDRESSIBILITY.
 966+         BALR  15,0 (INITIAL ADDRESSIBILITY).
 967+         B      12(,15) BRANCH AROUND ID FIELD
 968+         DC     AL1(7),CL7'FRISBEE' ID LENGTH AND ID
 969+         BCTR  15,0 (RESET INITIAL ADDRESSIBILITY
```

```
+               BCTR  15,0 ABSOLUTE ENTRY POINT).
+               STM   14,12,12(13) SAVE REGISTERS
+               LR    12,15 SETUP BASE REGISTER.
+               ST    13,SAVEAREA+4 CHAIN BACK
+               LA    0,SAVEAREA CHAIN FORWARD
+               ST    0,8(0,13)
+               LR    13,0 SET UP SAVE AREA POINTER
+               USING SAVEAREA,13 AND ADDRESSABILITY
 START          OPEN       (DUMPAREA,OUTPUT)
+               CNOP  0,4 ALIGN LIST TO FULLWORD
+START          BAL   1,*+8 LOAD REG1 W/LIST ADDR.
+               DC    AL1(143) OPTION BYTE
+               DC    AL3(DUMPAREA) DCB ADDRESS
+               SVC   19 ISSUE OPEN SVC
                LA         2,0
                LA         3,0
                LA         4,0
                LA         5,0
                LA         6,0
                LA         7,0
                LA         8,0
                LA         9,0
                LA         10,0
                LA         11,0
                LA         14,0
                LA         15,0
 *
 *
                     SNAP       ID=1,DCB=DUMPAREA,PDATA=REGS
+               CNOP  0,4
+               BAL   1,IHB0003 BRANCH AROUND PARAM LIST
+               DC    AL1(1) ID NUMBER
+               DC    AL1(0)
+               DC    AL1(130) OPTION FLAGS
+               DC    AL1(32) OPTION FLAGS
+               DC    A(DUMPAREA) DCB ADDRESS
+               DC    A(0) TCB ADDRESS
+               DC    A(0) ADDRESS OF SNAP-SHOT LIST
+IHB0003        DS    0H
+               SVC   51
 *
 *
                CASE       LABEL=A,LITA=0,OPRATOR=NE,REGB=15
+               ST    10,SAVE10
+               ST    11,SAVE11
+               LA    10,0
+               A     10,=F'0'
+               LR    11,15
+               CR    10,11
+               BE    B4
+               B     C4
+B4             L     10,SAVE10
+               L     11,SAVE11
+               B     A
+C4             L     10,SAVE10
+               L     11,SAVE11
                         A     4,=F'1'
                         A     5,=F'1'
                         A     6,=F'1'
                         A     7,=F'1'
                         A     8,=F'1'
                         A     9,=F'1'
                ENDCASE    OPTION=1,LABEL=A,LAB=HERE
+               B     HERE
+A              EQU   *
 *
 *
```

```
        CASE        LABEL=B,REGA=14,OPRATOR=EQ,LITA=0
+       ST     10,SAVE10
+       ST     11,SAVE11
               *,MORE THAN ONE LEFT OPERAND SPECIFIED FOR CASE
               *,CONDITION
+B6     L      10,SAVE10
+       L      11,SAVE11
+       B      B
+C6     L      10,SAVE10
+       L      11,SAVE11
                      A      4,=F'2'
                      A      5,=F'2'
                      A      6,=F'2'
                      A      7,=F'2'
                      A      8,=F'2'
                      A      9,=F'2'
        ENDCASE    OPTION=1,LABEL=B,LAB=HERE
+       B      HERE
+B      EQU    *
 *
 *
        CASE        LOCA=X,LOCB=Y,LABEL=C,OPRATOR=LT
+       ST     10,SAVE10
+       ST     11,SAVE11
+       L      10,X
+       L      11,Y
+       CR     10,11
+       BNL    B8
+       B      C8
+B8     L      10,SAVE10
+       L      11,SAVE11
+       B      C
+C8     L      10,SAVE10
+       L      11,SAVE11
                      A      4,=F'3'
                      A      5,=F'3'
                      A      6,=F'3'
                      A      7,=F'3'
                      A      8,=F'3'
                      A      9,=F'3'
        ENDCASE    OPTION=1,LABEL=C,LAB=HERE
+       B      HERE
+C      EQU    *
 *
 *
        ELSE
                      A      4,=F'4'
                      A      5,=F'4'
                      A      6,=F'4'
                      A      7,=F'4'
                      A      8,=F'4'
                      A      9,=F'4'
        ENDELSE    LAB=HERE
+HERE   EQU    *
 *
 *
 *
 *
               SNAP       ID=2,DCB=DUMPAREA,PDATA=REGS
+       CNOP   0,4
+       BAL    1,IHB0012 BRANCH AROUND PARAM LIST
+       DC     AL1(2) ID NUMBER
+       DC     AL1(0)
+       DC     AL1(130) OPTION FLAGS
+       DC     AL1(32) OPTION FLAGS
+       DC     A(DUMPAREA) DCB ADDRESS
```

```
1103+           DC     A(0) TCB ADDRESS
1104+           DC     A(0) ADDRESS OF SNAP-SHOT LIST
1105+IHB0012    DS     0H
1106+           SVC    51
1107 *
1108 *
1109            EXIT
1110+           L      13,4(,13) POP UP SAVE AREA
1111+           LM     14,12,12(13) RESTORE REGISTERS
1112+           MVI    12(13),X'FF' FLAG EXIT
1113+           BR     14 RETURN
1114            CLOSE      DUMPAREA
1115+           CNOP   0,4 ALIGN LIST TO FULLWORD
1116+           BAL    1,*+8 LOAD REG1 W/LIST ADDR
1117+           DC     AL1(128) OPTION BYTE
1118+           DC     AL3(DUMPAREA) DCB ADDRESS
1119+           SVC    20 ISSUE CLOSE SVC

1120 SAVEAREA DC            18A(0)
1121            DS            0F
1122 ZERO       DC            1F'0'
1123 SAVE8      DS            1F
1124 SAVE9      DS            1F
1125 SAVE10     DS            1F
1126 SAVE11     DS            1F
1127 POINTER    DC            1F'0'
1128 STACK      DS            100F
1129 X          DC            1F'4'
1130 Y          DC            1F'8'
1131 DUMPAREA DCB           DDNAME=TEAM,DSORG=PS,RECFM=VBA,                 X
                       MACRF=W,BLKSIZE=882,LRECL=125


1133+*                            DATA CONTROL BLOCK
1134+*
1135+DUMPAREA DC       0F'0' ORIGIN ON WORD BOUNDARY

1137+*                            DIRECT ACCESS DEVICE INTERFACE

1139+           DC     BL16'0' FDAD,DVTBL
1140+           DC     A(0) KEYLE,DEVT,TRBAL

1142+*                            COMMON ACCESS METHOD INTERFACE

1144+           DC     AL1(0) BUFNO
1145+           DC     AL3(1) BUFCB
1146+           DC     AL2(0) BUFL
1147+           DC     BL2'0100000000000000' DSORG
1148+           DC     A(1) IOBAD

1150+*                            FOUNDATION EXTENSION

1152+           DC     BL1'00000000' BFTEK,BFLN,HIARCHY
1153+           DC     AL3(1) EODAD
1154+           DC     BL1'01010100' RECFM
1155+           DC     AL3(0) EXLST

1157+*                            FOUNDATION BLOCK

1159+           DC     CL8'TEAM' DDNAME
1160+           DC     BL1'00000010' OFLGS
1161+           DC     BL1'00000000' IFLG
1162+           DC     BL2'0000000000100000' MACR

1164+*                            BSAM-BPAM-QSAM INTERFACE

1166+           DC     BL1'00000000' RER1
```

```
1167+          DC     AL3(1) CHECK, GERR, PERR
1168+          DC     A(1) SYNAD
1169+          DC     H'0' CIND1, CIND2
1170+          DC     AL2(882) BLKSIZE
1171+          DC     F'0' WCPO, WCPL, OFFSR, OFFSW
1172+          DC     A(1) IOBA
1173+          DC     AL1(0) NCP
1174+          DC     AL3(1) EOBR, EOBAD

1176+*                           BSAM-BPAM INTERFACE

1178+          DC     A(1) EOBW
1179+          DC     H'0' DIRCT
1180+          DC     AL2(125) LRECL
1181+          DC     A(1) CNTRL, NOTE, POINT
1182           END
1183                  =F'0'
1184                  =F'1'
1185                  =F'2'
1186                  =F'3'
1187                  =F'4'
```

Sample Program L Illustrating a CASE Macro Coded Incorrectly

```
STMT   SOURCE STATEMENT

 962 FRISBEE  ENTER     12,SAVEAREA
 963+FRISBEE  DS     0H
 964+         ENTRY FRISBEE DECLARE NAME ENTRY
 965+         USING *,12 DECLARE BASE ADDRESSIBILITY.
 966+         BALR   15,0 (INITIAL ADDRESSIBILITY).
 967+         B      12(,15) BRANCH AROUND ID FIELD
 968+         DC     AL1(7),CL7'FRISBEE' ID LENGTH AND ID
 969+         BCTR   15,0 (RESET INITIAL ADDRESSIBILITY
 970+         BCTR   15,0 ABSOLUTE ENTRY POINT).
 971+         STM    14,12,12(13) SAVE REGISTERS
 972+         LR     12,15 SETUP BASE REGISTER.
 973+         ST     13,SAVEAREA+4 CHAIN BACK
 974+         LA     0,SAVEAREA CHAIN FORWARD
 975+         ST     0,8(0,13)
 976+         LR     13,0 SET UP SAVE AREA POINTER
 977+         USING SAVEAREA,13 AND ADDRESSABILITY
 978 START    OPEN      (DUMPAREA,OUTPUT)
 979+         CNOP   0,4 ALIGN LIST TO FULLWORD
 980+START    BAL    1,*+8 LOAD REG1 W/LIST ADDR.
 981+         DC     AL1(143) OPTION BYTE
 982+         DC     AL3(DUMPAREA) DCB ADDRESS
 983+         SVC    19 ISSUE OPEN SVC
 984          LA        2,0
 985          LA        3,0
 986          LA        4,0
 987          LA        5,0
 988          LA        6,0
 989          LA        7,0
 990          LA        8,0
 991          LA        9,0
 992          LA        10,0
 993          LA        11,0
 994          LA        14,0
 995          LA        15,0
 996 *
 997 *
```

```
                SNAP      ID=1,DCB=DUMPAREA,PDATA=REGS
+          CNOP  0,4
+          BAL   1,IHB0003 BRANCH AROUND PARAM LIST
+          DC    AL1(1) ID NUMBER
+          DC    AL1(0)
+          DC    AL1(130) OPTION FLAGS
+          DC    AL1(32) OPTION FLAGS
+          DC    A(DUMPAREA) DCB ADDRESS
+          DC    A(0) TCB ADDRESS
+          DC    A(0) ADDRESS OF SNAP-SHOT LIST
+IHB0003   DS    0H
+          SVC   51
 *
 *
           CASE        LABEL=A,LITA=0,OPRATOR=NE,REGB=15
+          ST    10,SAVE10
+          ST    11,SAVE11
+          LA    10,0
+          A     10,=F'0'
+          LR    11,15
+          CR    10,11
+          BE    B4
+          B     C4
+B4        L     10,SAVE10
+          L     11,SAVE11
+          B     A
+C4        L     10,SAVE10
+          L     11,SAVE11
                       A     4,=F'1'
                       A     5,=F'1'
                       A     6,=F'1'
                       A     7,=F'1'
                       A     8,=F'1'
                       A     9,=F'1'
           ENDCASE     OPTION=1,LABEL=A,LAB=HERE
+          B     HERE
+A         EQU   *
 *
 *
           CASE        LABEL=B,REGA=14,OPRATOR=EQ,LITB=0
+          ST    10,SAVE10
+          ST    11,SAVE11
+          LR    10,14
+          LA    11,0
+          A     11,=F'0'
+          CR    10,11
+          BNE   B6
+          B     C6
+B6        L     10,SAVE10
+          L     11,SAVE11
+          B     B
+C6        L     10,SAVE10
+          L     11,SAVE11
                       A     4,=F'2'
                       A     5,=F'2'
                       A     6,=F'2'
                       A     7,=F'2'
                       A     8,=F'2'
                       A     9,=F'2'
           ENDCASE     OPTION=1,LABEL=B,LAB=HERE
+          B     HERE
+B         EQU   *
 *
 *
           CASE        LOCA=X,LOCB=Y,LABEL=C,OPRATOR=LL
+          ST    10,SAVE10
```

```
+          ST     11,SAVE11
+          L      10,X
+          L      11,Y
+          CR     10,11
                  *,ILLEGAL OPERATOR SPECIFIED FOR A CASE CONDITION
+B8        L      10,SAVE10
+          L      11,SAVE11
+          B      C
+C8        L      10,SAVE10
+          L      11,SAVE11
                        A     4,=F'3'
                        A     5,=F'3'
                        A     6,=F'3'
                        A     7,=F'3'
                        A     8,=F'3'
                        A     9,=F'3'
           ENDCASE   OPTION=1,LABEL=C,LAB=HERE
+          B      HERE
+C         EQU    *
 *
 *
           ELSE
                        A     4,=F'4'
                        A     5,=F'4'
                        A     6,=F'4'
                        A     7,=F'4'
                        A     8,=F'4'
                        A     9,=F'4'
           ENDELSE   LAB=HERE
+HERE      EQU    *
 *
 *
 *
 *
                SNAP      ID=2,DCB=DUMPAREA,PDATA=REGS
+          CNOP   0,4
+          BAL    1,IHB0012 BRANCH AROUND PARAM LIST
+          DC     AL1(2) ID NUMBER
+          DC     AL1(0)
+          DC     AL1(130) OPTION FLAGS
+          DC     AL1(32) OPTION FLAGS
+          DC     A(DUMPAREA) DCB ADDRESS
+          DC     A(0) TCB ADDRESS
+          DC     A(0) ADDRESS OF SNAP-SHOT LIST
+IHB0012   DS     0H
+          SVC    51
 *
 *
           EXIT
+          L      13,4(,13) POP UP SAVE AREA
+          LM     14,12,12(13) RESTORE REGISTERS
+          MVI    12(13),X'FF' FLAG EXIT
+          BR     14 RETURN
           CLOSE     DUMPAREA
+          CNOP   0,4 ALIGN LIST TO FULLWORD
+          BAL    1,*+8 LOAD REG1 W/LIST ADDR
+          DC     AL1(128) OPTION BYTE
+          DC     AL3(DUMPAREA) DCB ADDRESS
+          SVC    20 ISSUE CLOSE SVC

 SAVEAREA  DC         18A(0)
           DS         0F
 ZERO      DC         1F'0'
 SAVE8     DS         1F
 SAVE9     DS         1F
 SAVE10    DS         1F
 SAVE11    DS         1F
```

```
1130 POINTER  DC        1F'0'
1131 STACK    DS        100F
1132 X        DC        1F'4'
1133 Y        DC        1F'8'
1134 DUMPAREA DCB         DDNAME=TEAM,DSORG=PS,RECFM=VBA,                      X
                    MACRF=W,BLKSIZE=882,LRECL=125

1136+*                          DATA CONTROL BLOCK
1137+*
1138+DUMPAREA DC     OF'0' ORIGIN ON WORD BOUNDARY

1140+*                          DIRECT ACCESS DEVICE INTERFACE

1142+         DC     BL16'0' FDAD,DVTBL
1143+         DC     A(0) KEYLE,DEVT,TRBAL

1145+*                          COMMON ACCESS METHOD INTERFACE

1147+         DC     AL1(0) BUFNO
1148+         DC     AL3(1) BUFCB
1149+         DC     AL2(0) BUFL
1150+         DC     BL2'0100000000000000' DSORG
1151+         DC     A(1) IOBAD

1153+*                          FOUNDATION EXTENSION

1155+         DC     BL1'00000000' BFTEK,BFLN,HIARCHY
1156+         DC     AL3(1) EODAD
1157+         DC     BL1'01010100' RECFM
1158+         DC     AL3(0) EXLST

1160+*                          FOUNDATION BLOCK

1162+         DC     CL8'TEAM' DDNAME
1163+         DC     BL1'00000010' OFLGS
1164+         DC     BL1'00000000' IFLG
1165+         DC     BL2'0000000000100000' MACR

1167+*                          BSAM-BPAM-QSAM INTERFACE

1169+         DC     BL1'00000000' RER1
1170+         DC     AL3(1) CHECK, GERR, PERR
1171+         DC     A(1) SYNAD
1172+         DC     H'0' CIND1, CIND2
1173+         DC     AL2(882) BLKSIZE
1174+         DC     F'0' WCPO, WCPL, OFFSR, OFFSW
1175+         DC     A(1) IOBA
1176+         DC     AL1(0) NCP
1177+         DC     AL3(1) EOBR, EOBAD

1179+*                            BSAM-BPAM INTERFACE

1181+         DC     A(1) EOBW
1182+         DC     H'0' DIRCT
1183+         DC     AL2(125) LRECL
1184+         DC     A(1) CNTRL, NOTE, POINT
1185          END
1186                  =F'0'
1187                  =F'1'
1188                  =F'2'
1189                  =F'3'
1190                  =F'4'
```

APPENDIX D

THE USE OF NESTED DO MACROS

Sample Program M Illustrating Nested DO and ENDDO Macros

```
STMT    SOURCE STATEMENT

 962             PRINT     NOGEN
 963 FRISBEE     ENTER     12,SAVEAREA
 979 START       OPEN      (DUMPAREA,OUTPUT)
 985             LA        2,0
 986             LA        3,0
 987             LA        4,0
 988             LA        5,0
 989             LA        6,0
 990             LA        7,0
 991             LA        8,0
 992             LA        9,0
 993             LA        10,0
 994             LA        11,0
 995             LA        14,0
 996             LA        15,0
 997 *
 998 *
 999             SNAP      ID=1,DCB=DUMPAREA,PDATA=REGS
1011 *
1012 *
1013             DO        LOWNUM=1,BYNUM=1,HGHNUM=5,DOLOOP=S,LABEL=LOOPA
1075                       A      4,=F'1'
1076                       A      5,=F'1'
1077                       A      6,=F'1'
1078                       SNAP      ID=2,DCB=DUMPAREA,PDATA=REGS
1090 *
1091                  DO   WHILE=A,WREGA=8,WOP=LT,WLITB=10,LABEL=LOOPB
1107                       A      8,=F'1'
1108                       A      9,=F'1'
1109               ENDDO   LABEL=LOOPB
1116 *
1117                       SNAP      ID=3,DCB=DUMPAREA,PDATA=REGS
1129                       A      14,=F'1'
1130                       A      15,=F'1'
1131             ENDDO     LABEL=LOOPA
1138 *
1139 *
1140             CASE      LABEL=TESTA,REGA=4,OPRATOR=EQ,REGB=15
1153                       LA    4,0
1154                       LA    5,0
1155                       LA    6,0
1156                       LA    8,0
1157                       LA    9,0
1158                       LA    14,0
1159                       LA    15,0
1160                       SNAP      ID=4,DCB=DUMPAREA,PDATA=REGS
1172             ENDCASE   OPTION=2,LABEL=TESTA
1174 *
1175 *
```

```
1176           CASE      LABEL=TESTB,LITA=0,OPRATOR=NE,REGB=12
1190                     LA    4,5
1191                     LA    5,5
1192                     LA    6,5
1193                     LA    8,5
1194                     LA    9,5
1195                     SNAP     ID=5,DCB=DUMPAREA,PDATA=REGS
1207 *
1208                DO   LOWREG=4,BYREG=4,HGHNUM=100,DOLOOP=G,LABEL=EEE
1268                     A     14,=F'1'
1269                     A     15,=F'1'
1270             ENDDO LABEL=EEE
1277 *
1278                     SNAP     ID=6,DCB=DUMPAREA,PDATA=REGS
1290           ENDCASE   LABEL=TESTB,OPTION=2
1292 *
1293 *
1294           SNAP      ID=7,DCB=DUMPAREA,PDATA=REGS
1306 *
1307 *
1308           EXIT
1313           CLOSE     DUMPAREA

1319 SAVEAREA DC         18A(0)
1320          DS         0F
1321 ZERO     DC         1F'0'
1322 SAVE8    DS         1F
1323 SAVE9    DS         1F
1324 SAVE10   DS         1F
1325 SAVE11   DS         1F
1326 POINTER  DC         1F'0'
1327 STACK    DS         100F
1328 DUMPAREA DCB        DDNAME=TEAM,DSORG=PS,RECFM=VBA,                 X
                     MACRF=W,BLKSIZE=882,LRECL=125
1379          END
1380                 =F'1'
1381                 =F'5'
1382                 =F'4'
1383                 =F'10'
1384                 =F'0'
1385                 =F'100'
```

```
REGS AT ENTRY TO SNAP                                  ID = 001


REGS 0-7            00000030   9001B888   00000000   00000000
                    00000000   00000000   00000000   00000000

REGS 8-15           00000000   00000000   00000000   00000000
                    5001B820   0001BC20   00000000   00000000
END OF SNAP
```

REGS AT ENTRY TO SNAP ID = 002

```
REGS 0-7     000002A0  A0018994  00000000  00000000
             00000001  00000001  00000001  00000000

REGS 8-15    00000000  00000000  00000000  00000000
             50018820  00018C20  00000000  00000000
```

END OF SNAP

REGS AT ENTRY TO SNAP ID = 003

```
REGS 0-7     000002A0  800189F8  00000000  00000000
             00000001  00000001  00000001  00000000

REGS 8-15    0000000A  0000000A  00000000  00000000
             50018820  00018C20  00000000  00000000
```

END OF SNAP

REGS AT ENTRY TO SNAP ID = 002

```
REGS 0-7     000002A0  A0018994  00000000  00000000
             00000002  00000002  00000002  00000000

REGS 8-15    0000000A  0000000A  00000000  00000000
             50018820  00018C20  00000001  00000001
```

END OF SNAP

REGS AT ENTRY TO SNAP ID = 003

```
REGS 0-7     000002A0  800189F8  00000000  00000000
             00000002  00000002  00000002  00000000

REGS 8-15    0000000A  0000000A  00000000  00000000
             50018820  00018C20  00000001  00000000
```

END OF SNAP

```
REGS AT ENTRY TO SNAP                                ID = 002

REGS 0-7           000002A0  A0018994  00000000  00000000
                   00000003  00000003  00000003  00000000

REGS 8-15          0000000A  0000000A  00000000  00000000
                   50018820  0001BC20  00000002  00000001
END OF SNAP


REGS AT ENTRY TO SNAP                                ID = 003

REGS 0-7           000002A0  800189F8  00000000  00000000
                   00000003  00000003  00000003  00000000

REGS 8-15          0000000A  0000000A  00000000  00000000
                   50018820  0001BC20  00000002  00000000
END OF SNAP


REGS AT ENTRY TO SNAP                                ID = 002

REGS 0-7           000002A0  A0018994  00000000  00000000
                   00000004  00000004  00000004  00000000

REGS 8-15          0000000A  0000000A  00000000  00000000
                   50018820  0001BC20  00000003  00000001
END OF SNAP


REGS AT ENTRY TO SNAP                                ID = 003

REGS 0-7           000002A0  800189F8  00000000  00000000
                   00000004  00000004  00000004  00000000

REGS 8-15          0000000A  0000000A  00000000  00000000
                   50018820  0001BC20  00000003  00000000
END OF SNAP
```

REGS AT ENTRY TO SNAP ID = 002

REGS 0-7	000002A0	A0018994	00000000	00000000
	00000005	00000005	00000005	00000000
REGS 8-15	0000000A	0000000A	00000000	00000000
	50018820	00018C20	00000004	00000001

END OF SNAP

REGS AT ENTRY TO SNAP ID = 003

REGS 0-7	000002A0	800189F8	00000000	00000000
	00000005	00000005	00000005	00000000
REGS 8-15	0000000A	0000000A	00000000	00000000
	50018820	00018C20	00000004	00000000

END OF SNAP

REGS AT ENTRY TO SNAP ID = 005

REGS 0-7	000002A0	90018ACC	00000000	00000000
	00000005	00000005	00000005	00000000
REGS 8-15	00000005	00000005	00000000	00000000
	50018820	00018C20	00000005	00000001

END OF SNAP

REGS AT ENTRY TO SNAP ID = 006

REGS 0-7	000002A0	A0018BDC	00000000	00000000
	00000005	00000005	00000005	00000000
REGS 8-15	00000005	00000005	00000000	00000000
	50018820	00018C20	00000019	00000014

END OF SNAP

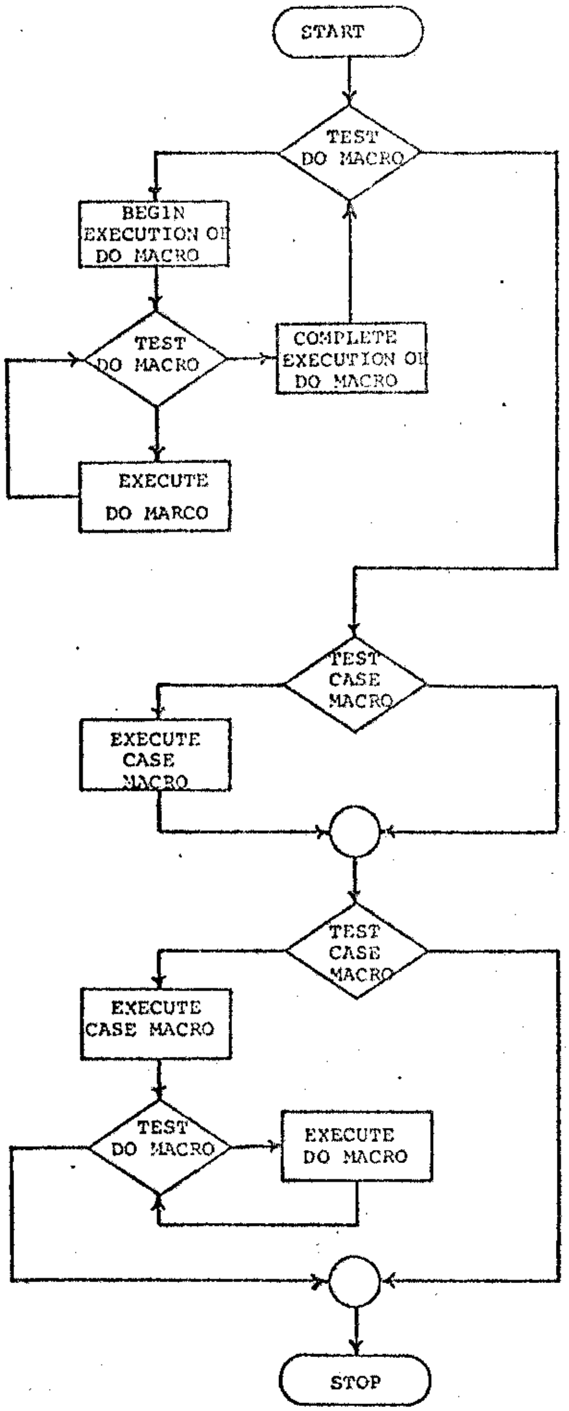

Fig. 27--Flowchart showing structured logic of Program M.

```
REGS AT ENTRY TO SNAP                                          ID = 007

REGS 0-7             000002A0    A0018BF4    00000000    00000000
                     00000005    00000005    00000005    00000000

REGS 8-15            00000005    00000005    00000000    00000000
                     50018820    00018C20    00000019    00000000
END OF SNAP
```

Program M with Expanded Macros

```
STMT   SOURCE STATEMENT

 962 FRISBEE  ENTER     12,SAVEAREA
 963+FRISBEE  DS    0H
 964+         ENTRY FRISBEE DECLARE NAME ENTRY
 965+         USING *,12 DECLARE BASE ADDRESSIBILITY.
 966+         BALR  15,0 (INITIAL ADDRESSIBILITY).
 967+         B     12(,15) BRANCH AROUND ID FIELD
 968+         DC    AL1(7),CL7'FRISBEE' ID LENGTH AND ID
 969+         BCTR  15,0 (RESET INITIAL ADDRESSIBILITY
 970+         BCTR  15,0 ABSOLUTE ENTRY POINT).
 971+         STM   14,12,12(13) SAVE REGISTERS
 972+         LR    12,15 SETUP BASE REGISTER.
 973+         ST    13,SAVEAREA+4 CHAIN BACK
 974+         LA    0,SAVEAREA CHAIN FORWARD
 975+         ST    0,8(0,13)
 976+         LR    13,0 SET UP SAVE AREA POINTER
 977+         USING SAVEAREA,13 AND ADDRESSABILITY
 978 START    OPEN      (DUMPAREA,OUTPUT)
 979+         CNOP  0,4 ALIGN LIST TO FULLWORD
 980+START    BAL   1,*+8 LOAD REG1 W/LIST ADDR.
 981+         DC    AL1(143) OPTION BYTE
 982+         DC    AL3(DUMPAREA) DCB ADDRESS
 983+         SVC   19 ISSUE OPEN SVC
 984          LA        2,0
 985          LA        3,0
 986          LA        4,0
 987          LA        5,0
 988          LA        6,0
 989          LA        7,0
 990          LA        8,0
 991          LA        9,0
 992          LA        10,0
 993          LA        11,0
 994          LA        14,0
 995          LA        15,0
 996 *
 997 *
 998          SNAP      ID=1,DCB=DUMPAREA,PDATA=REGS
 999+         CNOP  0,4
1000+         BAL   1,IHB0003 BRANCH AROUND PARAM LIST
1001+         DC    AL1(1) ID NUMBER
1002+         DC    AL1(0)
1003+         DC    AL1(130) OPTION FLAGS
1004+         DC    AL1(32) OPTION FLAGS
1005+         DC    A(DUMPAREA) DCB ADDRESS
1006+         DC    A(0) TCB ADDRESS
1007+         DC    A(0) ADDRESS OF SNAP-SHOT LIST
```

```
1008+IHB0003  DS    0H
1009+         SVC   51
1010 *
1011 *
1012          DO          LOWNUM=1,BYNUM=1,HGHNUM=5,DOLOOP=S,LABEL=LOOPA
1013+         ST    8,SAVE8
1014+         ST    9,SAVE9
1015+         ST    10,SAVE10
1016+         ST    11,SAVE11
1017+         LA    10,0
1018+         A     10,=F'1'
1019+         LA    11,0
1020+         A     11,=F'5'
1021+         LA    9,0
1022+         A     9,=F'1'
1023+         L     8,POINTER
1024+         ST    9,STACK(8)
1025+         LA    8,4(8)
1026+         ST    10,STACK(8)
1027+         LA    8,4(8)
1028+         ST    11,STACK(8)
1029+         LA    8,4(8)
1030+         ST    8,POINTER
1031+         L     8,SAVE8
1032+         L     9,SAVE9
1033+         L     10,SAVE10
1034+         L     11,SAVE11
1035+LOOPA    ST    8,SAVE8
1036+         ST    9,SAVE9
1037+         ST    10,SAVE10
1038+         ST    11,SAVE11
1039+         L     8,POINTER
1040+         S     8,=F'4'
1041+         L     11,STACK(8)
1042+         S     8,=F'4'
1043+         L     10,STACK(8)
1044+         S     8,=F'4'
1045+         L     9,STACK(8)
1046+         ST    8,POINTER
1047+         C     9,ZERO
1048+         BL    X4
1049+         CR    10,11
1050+         BH    Y4
1051+         B     Z4
1052+X4       CR    10,11
1053+         BL    Y4
1054+Z4       AR    10,9
1055+         L     8,POINTER
1056+         ST    9,STACK(8)
1057+         LA    8,4(8)
1058+         ST    10,STACK(8)
1059+         LA    8,4(8)
1060+         ST    11,STACK(8)
1061+         LA    8,4(8)
1062+         ST    8,POINTER
1063+         L     8,SAVE8
1064+         L     9,SAVE9
1065+         L     10,SAVE10
1066+         L     11,SAVE11
1067+         B     W4
1068+Y4       L     8,SAVE8
1069+         L     9,SAVE9
1070+         L     10,SAVE10
1071+         L     11,SAVE11
1072+         B     ALOOPA
1073+W4       EQU   *
1074                      A      4,=F'1'
```

```
                   A     5,=F'1'
                   A     6,=F'1'
                   SNAP     ID=2,DCB=DUMPAREA,PDATA=REGS
+         CNOP  0,4
+         BAL   1,IHB0005 BRANCH AROUND PARAM LIST
+         DC    AL1(2) ID NUMBER
+         DC    AL1(0)
+         DC    AL1(130) OPTION FLAGS
+         DC    AL1(32) OPTION FLAGS
+         DC    A(DUMPAREA) DCB ADDRESS
+         DC    A(0) TCB ADDRESS
+         DC    A(0) ADDRESS OF SNAP-SHOT LIST
+IHB0005  DS    0H
+         SVC   51
 *
               DO      WHILE=A,WREGA=8,WOP=LT,WLITB=10,LABEL=LOOPB
+LOOPB    EQU   *
+         ST    10,SAVE10
+         ST    11,SAVE11
+         LR    10,8
+         LA    11,0
+         A     11,=F'10'
+         CR    10,11
+         BNL   T6
+         L     10,SAVE10
+         L     11,SAVE11
+         B     V6
+T6       L     10,SAVE10
+         L     11,SAVE11
+         B     ALOOPB
+V6       EQU   *
                   A     8,=F'1'
                   A     9,=F'1'
               ENDDO LABEL=LOOPB
+         B     LOOPB
+CLOOPB   L     8,SAVE8
+         L     9,SAVE9
+BLOOPB   L     10,SAVE10
+         L     11,SAVE11
+ALOOPB   EQU   *
 *
                   SNAP     ID=3,DCB=DUMPAREA,PDATA=REGS
+         CNOP  0,4
+         BAL   1,IHB0008 BRANCH AROUND PARAM LIST
+         DC    AL1(3) ID NUMBER
+         DC    AL1(0)
+         DC    AL1(130) OPTION FLAGS
+         DC    AL1(32) OPTION FLAGS
+         DC    A(DUMPAREA) DCB ADDRESS
+         DC    A(0) TCB ADDRESS
+         DC    A(0) ADDRESS OF SNAP-SHOT LIST
+IHB0008  DS    0H
+         SVC   51
                   A     14,=F'1'
                   A     15,=F'1'
          ENDDO      LABEL=LOOPA
+         B     LOOPA
+CLOOPA   L     8,SAVE8
+         L     9,SAVE9
+BLOOPA   L     10,SAVE10
+         L     11,SAVE11
+ALOOPA   EQU   *
 *
 *
          CASE       LABEL=TESTA,REGA=4,OPRATOR=EQ,REGB=15
+         ST    10,SAVE10
+         ST    11,SAVE11
```

```
+               LR      10,4
+               LR      11,15
+               CR      10,11
+               BNE     B10
+               B       C10
+B10            L       10,SAVE10
+               L       11,SAVE11
+               B       TESTA
+C10            L       10,SAVE10
+               L       11,SAVE11
                                LA   4,0
                                LA   5,0
                                LA   6,0
                                LA   8,0
                                LA   9,0
                                LA   14,0
                                LA   15,0
                                SNAP     ID=4,DCB=DUMPAREA,PDATA=REGS
+               CNOP    0,4
+               BAL     1,IHB0011 BRANCH AROUND PARAM LIST
+               DC      AL1(4) ID NUMBER
+               DC      AL1(0)
+               DC      AL1(130) OPTION FLAGS
+               DC      AL1(32) OPTION FLAGS
+               DC      A(DUMPAREA) DCB ADDRESS
+               DC      A(0) TCB ADDRESS
+               DC      A(0) ADDRESS OF SNAP-SHOT LIST
+IHB0011        DS      0H
+               SVC     51
                ENDCASE    OPTION=2,LABEL=TESTA
+TESTA          EQU     *
 *
 *
                CASE       LABEL=TESTB,LITA=0,OPRATOR=NE,REGB=12
+               ST      10,SAVE10
+               ST      11,SAVE11
+               LA      10,0
+               A       10,=F'0'
+               LR      11,12
+               CR      10,11
+               BE      B13
+               B       C13
+B13            L       10,SAVE10
+               L       11,SAVE11
+               B       TESTB
+C13            L       10,SAVE10
+               L       11,SAVE11
                                LA   4,5
                                LA   5,5
                                LA   6,5
                                LA   8,5
                                LA   9,5
                                SNAP     ID=5,DCB=DUMPAREA,PDATA=REGS
+               CNOP    0,4
+               BAL     1,IHB0014 BRANCH AROUND PARAM LIST
+               DC      AL1(5) ID NUMBER
+               DC      AL1(0)
+               DC      AL1(130) OPTION FLAGS
+               DC      AL1(32) OPTION FLAGS
+               DC      A(DUMPAREA) DCB ADDRESS
+               DC      A(0) TCB ADDRESS
+               DC      A(0) ADDRESS OF SNAP-SHOT LIST
+IHB0014        DS      0H
+               SVC     51
 *
                    DO      LOWREG=4,BYREG=4,HGHNUM=100,DOLOOP=G,LABEL=EEE
```

```
+         ST    8,SAVE8
+         ST    9,SAVE9
+         ST    10,SAVE10
+         ST    11,SAVE11
+         LR    10,4
+         LA    11,0
+         A     11,=F'100'
+         LR    9,4
+         L     8,POINTER
+         ST    9,STACK(8)
+         LA    8,4(8)
+         ST    10,STACK(8)
+         LA    8,4(8)
+         ST    11,STACK(8)
+         LA    8,4(8)
+         ST    8,POINTER
+         L     8,SAVE8
+         L     9,SAVE9
+         L     10,SAVE10
+         L     11,SAVE11
+EEE      ST    8,SAVE8
+         ST    9,SAVE9
+         ST    10,SAVE10
+         ST    11,SAVE11
+         L     8,POINTER
+         S     8,=F'4'
+         L     11,STACK(8)
+         S     8,=F'4'
+         L     10,STACK(8)
+         S     8,=F'4'
+         L     9,STACK(8)
+         ST    8,POINTER
+         C     9,ZERO
+         BL    X15
+         CR    10,11
+         BH    Y15
+         B     Z15
+X15      CR    10,11
+         BL    Y15
+Z15      AR    10,9
+         L     8,POINTER
+         ST    9,STACK(8)
+         LA    8,4(8)
+         ST    10,STACK(8)
+         LA    8,4(8)
+         ST    11,STACK(8)
+         LA    8,4(8)
+         ST    8,POINTER
+         L     8,SAVE8
+         L     9,SAVE9
+         L     10,SAVE10
+         L     11,SAVE11
+         B     W15
+Y15      L     8,SAVE8
+         L     9,SAVE9
+         L     10,SAVE10
+         L     11,SAVE11
+         B     AEEE
+W15      EQU   *
                       A     14,=F'1'
                       A     15,=F'1'
                 ENDDO LABEL=EEE
+         B     EEE
+CEEE     L     8,SAVE8
+         L     9,SAVE9
+BEEE     L     10,SAVE10
+         L     11,SAVE11
```

```
1275+AEEE      EQU    *
1276 *
1277                  SNAP     ID=6,DCB=DUMPAREA,PDATA=REGS
1278+          CNOP   0,4
1279+          BAL    1,IHB0017 BRANCH AROUND PARAM LIST
1280+          DC     AL1(6) ID NUMBER
1281+          DC     AL1(0)
1282+          DC     AL1(130) OPTION FLAGS
1283+          DC     AL1(32) OPTION FLAGS
1284+          DC     A(DUMPAREA) DCB ADDRESS
1285+          DC     A(0) TCB ADDRESS
1286+          DC     A(0) ADDRESS OF SNAP-SHOT LIST
1287+IHB0017   DS     0H
1288+          SVC    51
1289           ENDCASE   LABEL=TESTB,OPTION=2
1290+TESTB     EQU    *
1291 *
1292 *
1293           SNAP       ID=7,DCB=DUMPAREA,PDATA=REGS
1294+          CNOP   0,4
1295+          BAL    1,IHB0019 BRANCH AROUND PARAM LIST
1296+          DC     AL1(7) ID NUMBER
1297+          DC     AL1(0)
1298+          DC     AL1(130) OPTION FLAGS
1299+          DC     AL1(32) OPTION FLAGS
1300+          DC     A(DUMPAREA) DCB ADDRESS
1301+          DC     A(0) TCB ADDRESS
1302+          DC     A(0) ADDRESS OF SNAP-SHOT LIST
1303+IHB0019   DS     0H
1304+          SVC    51
1305 *
1306 *
1307           EXIT
1308+          L      13,4(,13) POP UP SAVE AREA
1309+          LM     14,12,12(13) RESTORE REGISTERS
1310+          MVI    12(13),X'FF' FLAG EXIT
1311+          BR     14 RETURN
1312           CLOSE     DUMPAREA
1313+          CNOP   0,4 ALIGN LIST TO FULLWORD
1314+          BAL    1,*+8 LOAD REG1 W/LIST ADDR
1315+          DC     AL1(128) OPTION BYTE
1316+          DC     AL3(DUMPAREA) DCB ADDRESS
1317+          SVC    20 ISSUE CLOSE SVC

1318 SAVEAREA DC            18A(0)
1319           DS            0F
1320 ZERO      DC            1F'0'
1321 SAVE8     DS            1F
1322 SAVE9     DS            1F
1323 SAVE10    DS            1F
1324 SAVE11    DS            1F
1325 POINTER   DC            1F'0'
1326 STACK     DS            100F
1327 DUMPAREA DCB            DDNAME=TEAM,DSORG=PS,RECFM=VBA,                X
                      MACRF=W,BLKSIZE=882,LRECL=125


1329+*                            DATA CONTROL BLOCK
1330+*
1331+DUMPAREA DC      0F'0' ORIGIN ON WORD BOUNDARY

1333+*                            DIRECT ACCESS DEVICE INTERFACE

1335+          DC     BL16'0' FDAD,DVTBL
1336+          DC     A(0) KEYLE,DEVT,TRBAL
```

```
1338+*                               COMMON ACCESS METHOD INTERFACE

1340+           DC     AL1(0) BUFNO
1341+           DC     AL3(1) BUFCB
1342+           DC     AL2(0) BUFL
1343+           DC     BL2'0100000000000000' DSORG
1344+           DC     A(1) IOBAD

1346+*                               FOUNDATION EXTENSION

1348+           DC     BL1'00000000' BFTEK,BFLN,HIARCHY
1349+           DC     AL3(1) EODAD
1350+           DC     BL1'01010100' RECFM
1351+           DC     AL3(0) EXLST

1353+*                               FOUNDATION BLOCK

1355+           DC     CL8'TEAM' DDNAME
1356+           DC     BL1'00000010' OFLGS
1357+           DC     BL1'00000000' IFLG
1358+           DC     BL2'0000000000100000' MACR

1360+*                               BSAM-BPAM-QSAM INTERFACE

1362+           DC     BL1'00000000' RERI
1363+           DC     AL3(1) CHECK, GERR, PERR
1364+           DC     A(1) SYNAD
1365+           DC     H'0' CIND1, CIND2
1366+           DC     AL2(882) BLKSIZE
1367+           DC     F'0' WCPO, WCPL, OFFSR, OFFSW
1368+           DC     A(1) IOBA
1369+           DC     AL1(0) NCP
1370+           DC     AL3(1) EOBR, EOBAD

1372+*                                 BSAM-BPAM INTERFACE

1374+           DC     A(1) EOBW
1375+           DC     H'0' DIRCT
1376+           DC     AL2(125) LRECL
1377+           DC     A(1) CNTRL, NOTE, POINT
1378            END
1379                   =F'1'
1380                   =F'5'
1381                   =F'4'
1382                   =F'10'
1383                   =F'0'
1384                   =F'100'
```

BIBLIOGRAPHY

Books

Chapin, Ned, 360/370 Programming in Assembly Language, New York, McGraw-Hill Book Co., 1968.

Dahl, O. J. and others, Structured Programming, London, Academic Press, Inc., Ltd., 1972.

Hannula, Reino, Computers and Programming: A System/360-370 Assembler Language Approach, Boston, Ma., Houghton Mifflin Co., 1974.

Organick, Elliott I., The Multics System: An Examination of Its Structure, Cambridge, Ma., The MIT Press, 1972.

Weinberg, G., The Psychology of Computer Programming, New York, Von Nostrand Reinhold Co., 1971.

Wirth, Miklaus, Systematic Programming: An Introduction, Englewood Cliffs, N.J., Prentice-Hall, Inc., 1973.

Articles

Baker, F. T., "System Quality Through Structured Programming," Proc. FJCC, (1972), 339-343.

____________ and H. D. Mills, "Chief Programmer Teams," Datamation, XIX (December, 1973), 58-61.

Bauer, F. L., "Software and Software Engineering," SIAM Review, XV (April, 1973), 469-480.

Benson, Jeoffrey, "Structured Programming Techniques," Record of the 1973 IEEE Symposium on Computer Software Reliability, New York, (April, 1973), 143-147.

Bohm, C. and G. Jacopini, "Flow Diagrams, Turing Machines and Languages with only Two Formation Rules," Communications of the ACM, IX (May, 1966), 366-371.

Denning, P. J., "Is It Not Time to Define 'Structured Programming'?" SIGPLAN Notices, (February, 1974), 6-7.

Dijkstra, E. W., "The Humble Programmer," Communications of the ACM, XV (October, 1972), 859-866.

_______________, "Structured Programming," Software Engineering, Report on a Conference Sponsored by the NATO Science Committee, Rome, Italy, (October, 1969), 84-88.

_______________, "The Structure of the 'THE' Multiprogramming System," Communications of the ACM, XI (May, 1968), 341-356.

_______________, "GOTO Statement Considered Harmful," Communications of the ACM, XI (March, 1968), 147-148.

_______________, "Complexity Controlled by Hierarchecal Ordering of Function and Variability," Software Engineering, Report on a Conference Sponsored by the NATO Science Committee, Garmish, Germany, (October, 1968), 181-185.

Donaldson, James, "Structured Programming," Datamation, XIX (December, 1973), 52-54.

Fisher, David, "A Survey of Control Structures in Programming Languages," SIGPLAN Notices, (November, 1972), 1-13.

Hopkins, Martin, "A Case for the GOTO," SIGPLAN Notices, (November, 1972), 59-62.

Kent, William, "Assembler-Language Macroprogramming: A Tutorial Oriented Toward the IBM 360," Computing Surveys, (December, 1969), 183-196.

Leavenworth, B. M., "Programming With(out) the GOTO," SIGPLAN Notices, (November, 1972), 54-58.

McCracken, Daniel, "Revolution in Programming: An Overview," Datamation, XIX (December, 1973), 50-52.

Miller, E. and G. Landamood, "Structured Programming: Top-Down Approach," Datamation, XIX (December, 1973), 55-57.

Mills, H. D., "On the Development of Large Reliable Programs," Record of the 1973 IEEE Symposium on Computer Software Reliability, New York, (May, 1973), 155-159.

Parnas, D. L., "Information Distribution Aspects of Design Methodology," Information Processing 71, North-Holland Publishing Co., 339-344.

Peterson, W. W., et al, "On the Capabilities of While, Repeat, and Exit Statements," Communications of the ACM, XVI (August, 1973), 503-512.

Rosen, Saul, "Electronic Computers: A Historical Survey," Computing Surveys, (March, 1969), 7-36.

Tenny, Ted, "Structured Programming in FORTRAN," Datamation, XX (July, 1974), 110-115.

Wulf, William, "A Case Against the GOTO," SIGPLAN Notices, (November, 1972), 63-69.

Reports

IBM System/360 Operating System Assembler Language, IBM Form No. GC28-6514-8.

IBM System/360 Operating System Principles of Operation, IBM Form No. GA22-6821-8.

Improved Programming Technologies Management Overview, IBM Corporation, Data Processing Division, Systems Marketing, Installation Productivity Programs Department, August, 1973.

Kessler, M. M., *CONCEPTS* Report 4, OS/360 Assembly Language Block Structured Programming Macros, IBM.

Mills, H. D., Chief Programmer Teams: Principles and Procedures, Report No. FSC 71-5108, IBM Federal Systems Division, Gaithersburg, Maryland.

___________, How to Write Correct Programs and Know It, Report No. FSC 73-5998, IBM, Gaithersburg, Maryland, December, 1972.

___________, Mathematical Foundations for Structured Programming, Report No. FSC 72-6012, IBM, Gaithersburg, Maryland, February, 1972.

Special Interest Group on Programming Languages, Special Issue on Control Structures in Programming Languages, New York, New York, Association for Computing Machinery. Vol. VII (November, 1972).