AD-A031 457 GENERAL RESEARCH CORP SANTA BARBARA CALIF F/G 9/2
STRUCTURED PROGRAMMING TRANSLATORS. VOLUME I.(U)
AUG 76 R A MELTON F30602-75-C-0245
UNCLASSIFIED RADC-TR-76-253-VOL-1 NL

1 OF 1
AD A031457

END

DATE FILMED
12-76

RADC-TR-76-253, Vol I (of five)
Final Technical Report
August 1976

STRUCTURED PROGRAMMING TRANSLATORS

General Research Corporation

ROME AIR DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
GRIFFISS AIR FORCE BASE, NEW YORK 13441

This report consists of the following volumes:

I - Final Report
II - STRUCTRAN-1 User's Manual
III - STRUCTRAN-1 System Design and Implementation Manual
IV - STRUCTRAN-2 User's Manual
V - STRUCTRAN-2 System Design and Implementation Manual

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

This report has been reviewed and approved for publication.

APPROVED: Donald L. Mark

DONALD L. MARK
Project Engineer

APPROVED: Robert D. Krutz

ROBERT D. KRUTZ, Col, USAF
Chief, Information Sciences Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

Do not return this copy. Retain or destroy.

MISSION
of
Rome Air Development Center

RADC plans and conducts research, exploratory and advanced development programs in command, control, and communications (C^3) activities, and in the C^3 areas of information sciences and intelligence. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: RADC-TR-76-253 Vol I (of five)
2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle): STRUCTURED PROGRAMMING TRANSLATORS. Volume I.
5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report. May 75 - Jan 76
6. PERFORMING ORG. REPORT NUMBER: N/A

7. AUTHOR(s): R. A. Melton
8. CONTRACT OR GRANT NUMBER(s): F30602-75-C-0245

9. PERFORMING ORGANIZATION NAME AND ADDRESS:
General Research Corporation
P. O. Box 3587
Santa Barbara CA 93105
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 32010314

11. CONTROLLING OFFICE NAME AND ADDRESS:
Rome Air Development Center (ISIS)
Griffiss AFB NY 13441
12. REPORT DATE: August 1976
13. NUMBER OF PAGES: 28

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same
15. SECURITY CLASS. (of this report): UNCLASSIFIED
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)
Approved for public release; distribution unlimited.

16 AF-3201
17 320103

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

18. SUPPLEMENTARY NOTES
RADC Project Engineer: Donald L. Mark (ISIS)
DMAAC Project Engineer: Ms. Opal Power

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
Structured Programming
Precompilers
Translators
Software Tools

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
Structured programming translators, STRUCTRAN-1 and STRUCTRAN-2, are tools which facilitate structured programming using the FORTRAN language. STRUCTRAN- is a processor which accepts a structured language, DMATRAN, and is a pre-compiler for FORTRAN. DMATRAN contains five structured statement forms which can be mixed with ordinary FORTRAN constructs in the input text stream. To enhance readability of the processed source code, STRUCTRAN-1 automatically indents the listing of the source code according to control nesting level.
(cont'd)

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

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

402754 AB

To facilitate the translation of existing FORTRAN V programs into structured program form, STRUCTRAN-2 was developed. STRUCTRAN-2 provides the ability to translate unstructured control forms into logically equivalent structured control forms.

The precompiler (STRUCTRAN-1) and the translator (STRUCTRAN-2) were developed at GRC on a CDC 6400. Implementation and demonstration was on a Univac 1108 series computer at the Defense Mapping Agency Aerospace Center (DMAAC), St. Louis, Missouri. STRUCTRAN-1 uses ASA Standard FORTRAN X3.9. STRUCTRAN-2 accepts Univac 1108 FORTRAN V programs. The government has unlimited data rights to the two programs, STRUCTRAN-1 and STRUCTRAN-2, delivered under this contract.

CONTENTS

SECTION		PAGE
1	INTRODUCTION	1
	1.1 Structured Programming Translators for FORTRAN	2
2	DMATRAN CONSTRUCTS	4
	2.1 IF...THEN...ELSE...END IF	4
	2.2 DO WHILE...END WHILE	5
	2.3 CASE OF...CASE...CASE ELSE...END CASE	8
	2.4 DO UNTIL...END UNTIL	8
	2.5 BLOCK...END BLOCK and INVOKE	10
3	RESTRUCTURIZER (STRUCTRAN-2) METHODOLOGY	16
	3.1 Three Structured Formation Rules	16
	3.2 STRUCTRAN-2 Examples	18
	3.3 STRUCTRAN-2 Graphical Analysis	27
	REFERENCES	28

ILLUSTRATIONS

NO.		PAGE
1.1	Structured Programming Translators	3
2.1	DMATRAN IF...THEN...ELSE...END IF Construct	6
	(a) Using "GOTOs" for Conditional Execution	
	(b) Using DMATRAN for Conditional Execution	
	(c) STRUCTRAN-1 IF...THEN...ELSE...END IF Translation	
2.2	DMATRAN DO WHILE...END WHILE Construct	7
	(a) Using "GOTOs" for Iteration	
	(b) DMATRAN Iteration	
	(c) STRUCTRAN-1 DO WHILE...END WHILE Translation	
2.3	DMATRAN CASE OF...CASE...CASE ELSE...END CASE Construct	9
	(a) Using "GOTOs" for Selective Execution	
	(b) DMATRAN Selective Execution	
	(c) STRUCTRAN-1 CASE OF...CASE...CASE ELSE...END CASE Translation	
2.4	DMATRAN DO UNTIL...END UNTIL Construct	11
	(a) Using "GOTOs" to Iterate at Least Once	
	(b) DMATRAN Repeat at Least Once Iteration	
	(c) STRUCTRAN-1 DO UNTIL...END UNTIL Translation	
2.5	DMATRAN BLOCK...END BLOCK and INVOKE Construct	15
	(a) Using "GOTOs" for Internal Subroutines	
	(b) DMATRAN Internal Subroutine Construct	
	(c) STRUCTRAN-1 BLOCK...END BLOCK and INVOKE Translation	
3.1	Formation Rules and Related DMATRAN Constructs	17
3.2	Original FORTRAN Statements for XAMPLE1	18
3.3	Structurized DMATRAN Statements for XAMPLE1	18
3.4	Borrowing "A" with GOTOs	19
3.5	Structured Version Requires "A" to Appear Twice	20
3.6	Original FORTRAN Statements for XAMPLE2	20
3.7	Structurized DMATRAN Statements for XAMPLE2	21
3.8	Well-Structured Graph with Nested Iteration	22
3.9	DMATRAN Code Corresponding to Fig. 3.8	22
3.10	Original FORTRAN Statements for XAMPLE3	23
3.11	Structurized DMATRAN Statements for XAMPLE3	23
3.12	One Entry/Two Exit Iteration Expressed with GOTOs	24
3.13	Structured Version with Variable and Predicates Added	25
3.14	Original FORTRAN Statements for XAMPLE4	26
3.15	Structurized DMATRAN Statements for XAMPLE4	26

Evaluation

Recent advances in software development, specifically structured programming, have provided a basis for a new programming development process. Structured programming has the potential to provide significant improvements in programming activities. When fully practiced, it is estimated structured programming will decrease the cost of software by 60% while increasing the reliability and ease of maintenance of software.

Structured programs are, however, difficult to write in programming languages which do have the necessary structure needed for GOTO - free code. The Structured Programming Translators allow the benefits of structured programming to impact FORTRAN software development and maintenance. The purpose of the STRUCTRAN-1 pre-compiler is to allow the use of structured programming in FORTRAN, which, currently does not support it. It was essential that the translation from a structured program to a logically equivalent FORTRAN program be simple to perform and not consume excessive resources (in comparison to the compilation process).

The translation from a FORTRAN program to structured form was more difficult to accomplish. This translation need only be performed once, in contrast to the repetitive translations of a structured program being developed. STRUCTRAN-2 facilitates this translation process from FORTRAN to structured form in a more cost-effective and reliable manner than by manual means.

The advantages of the STRUCTRAN-1 pre-compiler and STRUCTRAN-2 translator are that they are expeditious solutions for interim problems and do not require compiler modifications. Users may now program in a structured programming environment, thus realizing the advantages stated above.

Donald L Mark

DONALD L. MARK
Project Engineer

1 INTRODUCTION

As part of its program for applying advanced technology to the important issue of software development and maintenance, Rome Air Development Center has contracted with General Research Corporation for the design, development, installation, and documentation of STRUCTRAN-1 and STRUCTRAN-2 (structured programming translators). The Defense Mapping Agency Aerospace Center (DMAAC) in St. Louis, Missouri is the current user of these translators. The work involves application of existing General Research algorithms and techniques to the problem of structured programming translators for FORTRAN. The specific tasks are to develop systems to automatically translate a structured extension of FORTRAN into compilable FORTRAN code, and to provide assistance in translating arbitrary FORTRAN programs into structured form.

This report describes the methodology for performing the structured programming translation processes. Section 2 describes DMATRAN, the structured extension to FORTRAN, and STRUCTRAN-1 the preprocessor that translates DMATRAN into pure FORTRAN. STRUCTRAN-1 is implemented as a computer-independent FORTRAN system. It consists of approximately 2,000 source statements, requires roughly 15K of memory, and is compatible with any FORTRAN compiler which accepts ANSI FORTRAN. Section 3 describes STRUCTRAN-2, a system which translates most FORTRAN programs into DMATRAN automatically. STRUCTRAN-2 is a larger system, consisting of approximately 15,000 source statements. It requires approximately 40K of memory when executed in overlay mode on the Univac 1108, and must have available random access disk or drum files. STRUCTRAN-2 is also readily machine transferrable.

STRUCTRAN-1 and STRUCTRAN-2 are both written in FORTRAN and both are operational on the Univac 1108 computer at the Defense Mapping Agency Aerospace Center (DMAAC) in St. Louis, Missouri, and the CDC 6400 computer at General Research Corporation in Santa Barbara, California, where they were developed.

In addition to this report, a number of other reports have been prepared as part of this effort. STRUCTRAN-1 computer program documentation consists of these three reports:

- STRUCTRAN-1 User's Manual. This report describes the various structured constructs and statement syntax of DMATRAN, a structured extension to FORTRAN. It also details the use of the STRUCTRAN-1 preprocessor, which translates DMATRAN into pure FORTRAN. Procedures for using either the machine independent or Univac 1108 adapted version of STRUCTRAN-1 are included.

- STRUCTRAN-1 System Design and Implementation. This report contains a description of STRUCTRAN-1 software design, the processing performed by STRUCTRAN-1, and a description of the subroutines in STRUCTRAN-1.

- STRUCTRAN-1 Software Analysis Collection. This is a set of automatically produced documentation reports. The source for each STRUCTRAN-1 routine has been automatically analyzed to produce enhanced source listings with indentation and control structure identification, intermodule dependence, all module invocations,

module control structure, and symbol cross reference. A Univac 1108 documentation facility has also been used to produce system-wide calling matrices and common-block reference reports.

The STRUCTRAN-2 computer program documentation consists of three reports:

- STRUCTRAN-2 User's Manual. This report describes the use of the STRUCTRAN-2 processor in translating FORTRAN programs into structured form expressed in DMATRAN.
- STRUCTRAN-2 System Design and Implementation Manual. This contains a description of STRUCTRAN-2 software design, the organization and contents of the STRUCTRAN-2 data base, and a description of the subroutines in each STRUCTRAN-2 component.
- STRUCTRAN-2 Software Analysis Collection. This is a set of automatically produced documentation reports. The source for each STRUCTRAN-2 component has been analyzed to produce enhanced source listings with indentation and control structure identification, intermodule dependence, all module invocations, module control structure, and symbol cross reference. A Univac 1108 documentation facility has also been used to produce systemwide calling matrices and common block reference reports.

The System Design and Implementation Manuals and the Software Analysis Collections are intended primarily for software maintenance.

1.1 STRUCTURED PROGRAMMING TRANSLATORS FOR FORTRAN

Recent advances in software development, specifically structured programming practices, have provided a basis for a new programming development process. Structured programming has the potential to provide significant improvements in programming activities. Improvements in the development and management of software may likewise provide a basis for improving software reliability and quality. Structured programming is based on a mathematically proven Structure Theorem which states that any proper program, which is defined as a program with one entry and one exit, is equivalent to a program that contains as logic structures only the following items:

- Sequences of two or more operations
- Conditional branch to one of two operations and return (IF y THEN b ELSE c)
- Repetition of an operation while a condition is true (DO WHILE)

Each of the three items represents a proper program. A large and complex program may be developed by the appropriate nesting of these within each other. The logic flow of such a program always proceeds from the beginning to the end without unnecessary branching. Where only these structures are used in the programming, there are no unconditional branches or statement labels to branch to.

A major characteristic of programs written using these structures is that they can be read literally from top to bottom--there is no "jumping around" which is typical of code which contains unconditional branches. This readability is a major advantage for testing, maintaining, or otherwise referencing the code.

Another advantage of possibly greater benefit is the additional program design work required to produce structured code. The programmer must think through the processing problem: he must not only write down everything, but must do so in such a way that there are no afterthoughts with subsequent jump-outs and jump-backs, eliminating indiscriminate use of a section of code from several locations because it "just happens" to be there. Instead, the programmer must completely think through the control logic of the module at one time in order to provide the proper structural framework for control. The result is programs written in a more uniform way because there is less freedom for arbitrary variety.

Such a program is much easier to understand than an unstructured one; readability is improved and the danger that the programmer will overlook logical errors during implementation is minimized. Improved readability, in combination with the greater simplicity obtained by structured programming, leads to improved maintainability. Further, because of the simplicity of structured code, a programmer can generate a much larger amount of code in a given time.

This report is concerned with expanding the benefits of structured programming to FORTRAN system development and maintenance. Figure 1.1 indicates the translation processes which are required. The translation from a DMATRAN program to a logically equivalent FORTRAN program has to be performed whenever the DMATRAN program is modified. It is essential that this translation be simple to perform and not consume excessive resources (compared to the compilation process). Implementing STRUCTRAN-1 as a preprocessor accomplishes these objectives. Translation from a FORTRAN program to structured form is much more difficult. It is appropriate to perform this translation process when operational FORTRAN programs are to be modified, maintained, or documented. The translation need only be performed once, in contrast to the repetitive translations of a DMATRAN program being developed. Of course, a FORTRAN program translated to DMATRAN does not have the benefits of additional program design work required to write a structured program in the first place. STRUCTRAN-2 facilitates the translation process from FORTRAN to DMATRAN in a more cost-effective and reliable manner than manual translation.

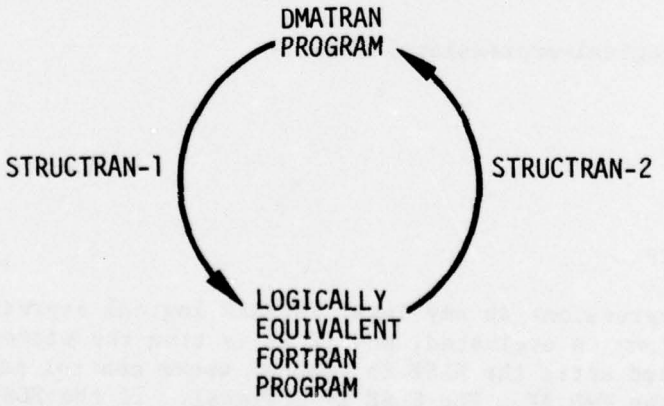

Figure 1.1. Structured Programming Translators

2 DMATRAN CONSTRUCTS

DMATRAN replaces FORTRAN control statements with five structured constructs.

- IF...THEN...ELSE...END IF. This construct provides block structuring of conditionally executable sequences of statements.
- DO WHILE...END WHILE. This construct permits iteration of a code segment while a specified condition remains true.
- CASE OF...CASE...CASE ELSE...END CASE. This construct allows multiple choices for program action selection.
- DO UNTIL...END UNTIL. This construct permits iteration until a specified condition becomes true.
- BLOCK<name>...END BLOCK. This construct and corresponding INVOKE<name> statement provide a facility for top-down programming and internal subroutines.

These statement forms can be intermixed with ordinary FORTRAN non-control statements in the text stream processed by the STRUCTRAN-1 preprocessor. DMATRAN statements are converted by the preprocessor to the FORTRAN equivalent, and the resulting file can be compiled by the FORTRAN compiler in the normal manner.

A structured GOTO-free program has a highly visible form which reveals its intended function more readily than a FORTRAN program containing GOTO statements (which has no apparent form). Well-defined blocks of code are executed in a sequential, top-down manner. The possible sequences of blocks which can be executed are also well-defined. The following simple examples illustrate these constructs.

2.1 IF...THEN...ELSE...END IF

The general form of the IF-construct consists of three DMATRAN statements:

```
IF(<logical-expression>) THEN
   .
   .
   .
ELSE
   .
   .
   .
END IF
```

where <logical-expression> is any legal FORTRAN logical expression. The <logical-expression> is evaluated, and if it is true the statements following the IF are executed until the ELSE is reached where control passes to the first statement after the END IF. The ELSE is optional. If the ELSE is absent and the <logical-expression> is false, control passes to the first statement after the END IF. If the ELSE is present and the <logical-expression> is false,

control passes to the statement following the ELSE so that the statements after the ELSE are executed.

Figure 2.1a is an example of a FORTRAN program which conditionally executes one of two assignment statements. By tracing the paths (two in this case) through this program, it can be seen that "OUT" is set to zero if "IN .GE. 10," otherwise "OUT" is incremented by one. An equivalent DMATRAN program is shown in Fig. 2.1b. The indentation indicates that there are two conditionally executable blocks of code. The IF...THEN...ELSE...END IF construct states that one or the other, but not both, blocks will be executed. It is immediately apparent that "OUT" is set to zero only if "IN .GE. 10." The translation template used by STRUCTRAN-1 for the IF...THEN...ELSE...END IF construct is shown in Fig. 2.1c.

2.2 DO WHILE...END WHILE

The general form of the DO WHILE-construct consists of two DMATRAN statements:

```
DO WHILE(<logical-expression>)
   .
   .
   .
END WHILE
```

where <logical-expression> is any legal FORTRAN logical expression. The DO WHILE construct represents an iteration in which execution occurs in the following manner:

(1) The value of <logical-expression> is found: if true, the statements contained within the DO WHILE block are executed; if false, control passes to the statement immediately following the END WHILE.

(2) If the statements within the DO WHILE block have been executed, the value of <logical-expression> is checked again, with the same consequences as in (1).

The iterative block in the DO WHILE...END WHILE may be executed zero or more times. In general it is necessary to initialize the loop control variable before entering the DO WHILE construct, and also to modify it within the DO WHILE construct.

A simple iteration is implemented with FORTRAN GOTOs in Fig. 2.2a. An equivalent DMATRAN form of this program (Fig. 2.2b) explicitly indicates the block of code which can be repeated, and takes fewer statements to write. As in the FORTRAN equivalent of Fig. 2.2b, the iterative block in the DO WHILE... END WHILE may be executed zero or more times. Figure 2.2c provides the STRUCTRAN-1 translation template for the DO WHILE construct.

The IF construct and the DO WHILE construct are sufficient to express the control portion of any algorithm which can be implemented in FORTRAN. However, for greatest convenience in implementation of software systems with

```
      SUBROUTINE IFTHEN (IN,OUT)
C
C     FORTRAN CODE TO EXECUTE ONE
C     OF TWO STATEMENTS
C
      INTEGER OUT
      IF (IN.GE.10) GO TO 10
      OUT=OUT+1
      GO TO 20
   10 OUT=0
   20 CONTINUE
      RETURN
      END
```

(a) Using "GOTOs" for Conditional Execution

```
      SUBROUTINE IFTHEN (IN,OUT)


C           ONE OR THE OTHER STATEMENTS
C           BUT NOT BOTH STATEMENTS
C           WILL BE EXECUTED
C
      INTEGER OUT
      IF (IN.GE.10) THEN
         OUT=0
      ELSE
         OUT=OUT+1
      END IF
      RETURN
      END
```

(b) Using DMATRAN for Conditional Execution

```
      SUBROUTINE IFTHEN (IN,OUT)
      INTEGER OUT
      IF (IN.GE.10) GO TO 99998
      GO TO 99997
99998 CONTINUE
        OUT=0
      GO TO 99996
99997 CONTINUE
        OUT=OUT+1
99996 CONTINUE
      RETURN
      END
```

(c) STRUCTRAN-1 IF...THEN...ELSE...END IF Translation

```
      SUBROUTINE DOWHIL (IN,OUT,I)
C
C     FORTRAN ITERATION USING GOTO
C
      INTEGER OUT
      DIMENSION IN(50)
      I=1
   10 CONTINUE
      IF (IN(I).EQ.OUT .AND. I.LE.50)GO TO 20
      I=I+1
      GO TO 10
   20 CONTINUE
      RETURN
      END
```

(a) Using "GOTOs" for Iteration

```
      SUBROUTINE DOWHIL (IN,OUT,I)

C           ITERATION IS PERFORMED
C           WHILE IN(I) DOES NOT EQUAL OUT
C
      INTEGER OUT
      DIMENSION IN(50)
      I=1
      DO WHILE (IN(I).NE.OUT .AND. I.LE.50)
         I=I+1
      END WHILE
      RETURN
      END
```

(b) DMATRAN Iteration

```
      SUBROUTINE DOWHIL (IN,OUT,I)
      INTEGER OUT
      DIMENSION IN(50)
      I=1
99998 IF (IN(I).NE.OUT .AND. I.LE.50) GO TO 99997
      GO TO 99996
99997 CONTINUE
        I=I+1
      GO TO 99998
99996 CONTINUE
      RETURN
      END
```

(c) STRUCTRAN-1 DO WHILE...END WHILE Translation

Figure 2.2. DMATRAN DO WHILE...END WHILE Construct

structured programming techniques, some additional statement forms are highly desirable. The CASE construct is a selection structure and the DO UNTIL is an iteration structure, while the BLOCK construct enhances modularity.

2.3 CASE OF...CASE...CASE ELSE...END CASE

The CASE statement provides a way to select which group of statements will be executed. The general form of the CASE construct consists of the following DMATRAN statements:

```
CASE OF(<integer-expression>)
CASE(<i>)
  .
  .
  .
CASE(<j>)
  .
  .
  .
CASE ELSE
  .
  .
  .
END CASE
```

<i> and <j> represent integers of positive value. They may be in any order, and there is no limit to how many integers may be listed.

The <integer-expression> is computed, and if any of the specified integers in the CASE list are equal to the value of the expression, then the transfer of control is to the statements which follow that particular CASE. If there is no such CASE, and the CASE ELSE statement is present, then the block of statements following the CASE ELSE is executed; otherwise, no block is executed. If there are two CASE statements with the same CASE index, the first occurring one is executed (if the CASE expression has that value). After the block of statements selected has been executed, control transfers to the statement after the END CASE.

Figure 2.3a is a FORTRAN program which selects one of three assignment statements for execution. When this selection is based on the possible positive values of an integer expression, a CASE construct explicitly indicates this structure. Figure 2.3b is an example of an equivalent **DMATRAN** program which utilizes the CASE construct. The CASE construct's translation by STRUCTRAN-1 is demonstrated on Fig. 2.3c.

2.4 DO UNTIL...END UNTIL

The general form of the DO UNTIL consists of two DMATRAN statements.

```
DO UNTIL(<logical-expression>)
    :
    :
END UNTIL
```

```
      SUBROUTINE KASE (IN,OUT)
C
C     FORTRAN CODE TO EXECUTE ONE
C     OF THREE STATEMENTS
C
      INTEGER OUT
      IF (IN.EQ.10) GO TO 30
      IF (IN.EQ.15) GO TO 40
      OUT=IN*10
      GO TO 50
   30 OUT=IN
      GO TO 50
   40 OUT=IN-3
   50 CONTINUE
      RETURN
      END
```

(a) Using "GOTOs" for Selective Execution

```
      SUBROUTINE KASE (IN,OUT)

C     THE SEQUENCE OF STATEMENTS EXECUTED
C     DEPENDS ON WHETHER IN IS EQUAL TO
C     10 OR 15 OR NEITHER
C
      INTEGER OUT
      CASE OF (IN)
      CASE (10)
         OUT=IN
      CASE (15)
         OUT=IN-3
      CASE ELSE
         OUT=IN*10
      END CASE
      RETURN
      END
```

(b) DMATRAN Selective Execution

```
      SUBROUTINE KASE (IN,OUT)
      INTEGER OUT
      I99998=IN
      IF (I99998.NE.(10)) GO TO 99996
        OUT=IN
      GO TO 99997
99996 CONTINUE
      IF (I99998.NE.(15)) GO TO 99995
        OUT=IN-3
      GO TO 99997
99995 CONTINUE
        OUT=IN*10
99997 CONTINUE
      RETURN
      END
```

(c) STRUCTRAN-1 CASE OF...CASE...CASE ELSE...END CASE Translation

Figure 2.3. DMATRAN CASE OF...CASE...CASE ELSE...END CASE Construct

The statements enclosed within the DO UNTIL and the END UNTIL are executed once. Then the <logical-expression> is evaluated and, if not true, iteration and evaluation of the expression continue until it is true. At that time execution of the statements following the END UNTIL begins.

Figure 2.4a and 2.4b indicate that the DO UNTIL construct is essentially a generalization of the FORTRAN DO loop. The iteration part of a DO UNTIL... END UNTIL is performed at least once as is the case with a FORTRAN DO loop. After completion of the DO UNTIL iteration, J will have the value 16 and I the value 11. It is important to note that when using DO WHILE or DO UNTIL constructs, the iteration variable must be initialized before entering the iteration and modified within the iteration. Figure 2.4c demonstrates the STRUCTRAN-1 translation template for the DO UNTIL construct.

2.5 BLOCK...END BLOCK AND INVOKE

The constructs described in the preceding paragraphs allow most programming tasks to be done in a well-structured manner. However, they do not always permit top-down programming. To implement this method, one must be able to refer to an action (such as "compute array element") before the code for it is actually available.

```
      SUBROUTINE DOUNTL (IN,OUT)
C
C     FORTRAN DO-LOOP
C
      DIMENSION IN(10)
      INTEGER OUT(20)
      J=6
      DO 20 I=1,10
      OUT(J)=IN(I)
      J=J+1
   20 CONTINUE
      RETURN
      END
```

(a) Using "GOTOs" to Iterate at Least Once

```
      SUBROUTINE DOUNTL (IN,OUT)


C          ITERATION IS PERFORMED AT LEAST ONCE
C          UNTIL I IS GREATER THAN 10
C
      DIMENSION IN(10)
      INTEGER OUT(20)
      I=1
      J=6
      DO UNTIL (I.GT.10)
         OUT(J)=IN(I)
         I=I+1
         J=J+1
      END UNTIL
      RETURN
      END
```

(b) DMATRAN Repeat at Least Once Iteration

```
      SUBROUTINE DOUNTL (IN,OUT)
      DIMENSION IN(10)
      INTEGER OUT(20)
      I=1
      J=6
      GO TO 99998
99997 IF (I.GT.10) GO TO 99996
99998 CONTINUE
        OUT(J)=IN(I)
        I=I+1
        J=J+1
      GO TO 99997
99996 CONTINUE
      RETURN
      END
```

(c) STRUCTRAN-1 DO UNTIL...END UNTIL Translation

Figure 2.4. DMATRAN DO UNTIL...END UNTIL Construct

The standard method for doing this is by calling subroutines. However, subroutines have disadvantages. The overhead involved in calling them is often high. Also, even though the subroutine and the routine calling it may share many (in fact, often all) variables, in FORTRAN (and DMATRAN) those variables used in both a routine and a subroutine must either be placed in COMMON or passed as parameters.

In many cases, a subroutine uses only variables which are already in the routine which calls it. Use of a subroutine internal to the calling routine eliminates the need for any mechanism (such as parameters or common blocks) for referring to the variables required.

A facility for creating and using this type of subroutine exists in DMATRAN. This construct is called a BLOCK which may be defined as an internal, parameterless procedure with all variables global. A BLOCK can be called only from the individual routine (main program, subroutine, or function) in which it is compiled; it cannot be called from an external routine, nor can it be passed as a parameter to another routine. A BLOCK is simply a segment of code of the routine which contains it. The BLOCK is exercised only if it is invoked.

The general form of a BLOCK construct consists of two DMATRAN statements:

```
BLOCK(<block-name>)
  .
  .
  .
END BLOCK
```

where <block-name> is any string of characters (e.g., COMPUTE.INDEX or PRINT-CURRENT-STATUS). The name of a BLOCK may be arbitrarily long, so that the name can have mnemonic significance. However, the first six characters must be unique.

A BLOCK is called by an INVOKE statement, whose format is:

```
INVOKE(<block-name>)
```

When an INVOKE statement is executed, control is transferred to the first statement in the BLOCK; when the END BLOCK is reached, control goes to the statement following the INVOKE of the BLOCK. Though BLOCKs can be nested (one BLOCK completely inside of another), no recursion is allowed in the calling of BLOCKs (i.e., a BLOCK cannot invoke itself). Also, the name of a BLOCK is known throughout the whole routine in which it is contained.

The following are examples of the two major uses of the BLOCK constructs.

Example 1: Top-Down Programming

```
I=1
DO UNTIL (I .GT. N)
      J=1
      DO UNTIL (J .GT. N)
            INVOKE(COMPUTE.ARRAY.ELEMENT)
            J=J+1
      END UNTIL
      I=I+1
END UNTIL
```

and, some place later in the same routine have the code

```
BLOCK(COMPUTE.ARRAY.ELEMENT)
      code to compute A(I,J)
      A(I,J) = value computed
      J=J+1
END BLOCK
```

The use of a BLOCK construct enhances readability and understandability of the program.

Example 2: Internal Subroutine

S_1 and S_2 in the following code represent two sets of statements. The use of BLOCK in Method 2 below eliminates the need for duplicating code.

Method 1:

```
IF(A) THEN
      IF(C) THEN
            S1
      ELSE
            S2
      END IF
ELSE
      IF(D) THEN
            S2
      ELSE
            S1
      END IF
END IF
```

Method 2:

```
IF(A) THEN
      IF(C) THEN
            INVOKE(BLOCK-A)
      ELSE
            INVOKE(BLOCK-B)
      END IF
ELSE
      IF(D) THEN
            INVOKE(BLOCK-B)
      ELSE
            INVOKE(BLOCK-A)
      END IF
END IF
```

where the BLOCKs are defined as

```
BLOCK(BLOCK-A)
      S1
END BLOCK
```

and

```
BLOCK(BLOCK-B)
      S2
END BLOCK
```

The FORTRAN assigned GOTO statement is often used to implement segments of the code which can be used as internal subroutines. The FORTRAN program in Fig. 2.5a is a simple example of this. The same program is much cleaner and clearer when expressed with the DMATRAN BLOCK statement (Fig. 2.5b). The BLOCK structures are clearly identified, and the ability to use a long BLOCK name indicates the intended purpose of the BLOCK. Note that all INVOKEs must occur before the corresponding BLOCK statement, the first six characters of each BLOCK name must be unique within a module, and BLOCK constructs can only be entered by using an INVOKE statement. The STRUCTRAN-1 translation of BLOCKs is indicated in Fig. 2.5c.

```
      SUBROUTINE BLOK (WIDTH,LENGTH)
C
C     FORTRAN ASSIGNED GOTO
C
      INTEGER AREA,WIDTH
      LENGTH=LENGTH+20
      WIDTH=WIDTH+30
      ASSIGN 3 TO JUMP
    1 GO TO JUMP,(7,3,6)
    3 AREA=LENGTH*WIDTH
      ASSIGN 6 TO JUMP
      GO TO 1
    6 WRITE (6,1) AREA
    1 FORMAT (10X,I20)
      ASSIGN 7 TO JUMP
      GO TO 1
    7 RETURN
      END
```

(a) Using "GOTOs" for Internal Subroutines

```
      SUBROUTINE BLOK (WIDTH,LENGTH)

      INTEGER AREA,WIDTH
      LENGTH=LENGTH+20
      WIDTH=WIDTH+30
      INVOKE (COMPUTE AREA)
      INVOKE (PRINT AREA)
      BLOCK (COMPUTE AREA)
         AREA=LENGTH*WIDTH
      END BLOCK
      BLOCK (PRINT AREA)
         WRITE (6,1)AREA
    1    FORMAT (10X,I20)
      END BLOCK
      RETURN
      END
```

(b) DMATRAN Internal Subroutine Construct

```
      SUBROUTINE BLOK (WIDTH,LENGTH)
      INTEGER AREA,WIDTH
      LENGTH=LENGTH+20
      WIDTH=WIDTH+30
      ASSIGN 99997 TO I99998
      GO TO 99998
99997 CONTINUE
      ASSIGN 99995 TO I99996
      GO TO 99996
99995 CONTINUE
      GO TO 99994
99998 CONTINUE
        AREA=LENGTH*WIDTH
      GO TO I99998,(99997)
99994 CONTINUE
      GO TO 99993
99996 CONTINUE
        WRITE (6,1)AREA
    1   FORMAT (10X,I20)
      GO TO I99996,(99995)
99993 CONTINUE
      RETURN
```

(c) STRUCTRAN-1 BLOCK...END BLOCK and INVOKE Translation

Figure 2.5. DMATRAN BLOCK...END BLOCK and INVOKE Construct

3 RESTRUCTURIZER (STRUCTRAN-2) METHODOLOGY

The effort spent in understanding, maintaining, modifying, testing, and documenting existing unstructured FORTRAN software can be minimized by automatic translation to a structured extension of FORTRAN. Automatic translation is efficient and does not introduce any errors. STRUCTRAN-2 provides automatic translation from **FORTRAN to DMATRAN.**

The system produces programs with five formation rules (sequential, conditional, two kinds of iteration, and invocation). The literature on translating "GOTO" programs to DO WHILE programs has established a class of programs which can be rewritten without adding any extra variables or new predicates; additionally, it has provided algorithms for translating programs that require additional information for their maintenance. The techniques employed in STRUCTRAN-2 closely parallel those described in the literature.

The advantages of translating a "GOTO" program to a DO WHILE program all result from the fact that it is easier to understand what the structured version is meant to do. The DO WHILE programs produced by STRUCTRAN-2 clearly exhibit the sequences of actions which can be performed by revealing the structure of the original "GOTO" programs. Translating "GOTO" programs is useful in maintaining, modifying, testing, and documenting existing software. The structured version can replace the original or be used to refer back to statements in the original program. The advantage of performing the translation automatically lies in the efficiency of the operation and the reliability of the resulting DO WHILE program. The translated program performs the same algorithm as the original, and requires no additional debugging if the original program was correct.

3.1 THREE STRUCTURED FORMATION RULES

FORTRAN relies almost exclusively on various forms of the "GOTO" to modify basic sequential flow of control. This allows a large variety of control structures to be implemented. Boehm and Jacopini[1] proved that all flowcharts (programs) can be redrawn (rewritten) using only three formation rules. The three formation rules are shown in Fig. 3.1, along with the DMATRAN[2] statements which correspond to each formation rule. To improve the efficiency of restructured code, STRUCTRAN-2 translates appropriate code segments into DO UNTIL...END UNTIL and BLOCK...END BLOCK constructs.

Boehm and Jacopini demonstrate that flowcharts (programs) exist which can be translated only if new predicates and new Boolean variables are added to the translated version. They describe a conceptual technique of performing such transformations. Ashcroft and Manna[3] are concerned with programs which require additional predicates and variables to be added if they are to be rewritten using only the three formation rules. They describe two algorithms for performing this transformation and **prove** that a program exists which can be translated only by such an algorithm. Sullivan[4] proves that the class of programs which do not contain iterations with single entries/multiple exits or multiple entries/multiple exits can be equivalently rewritten in structured form without adding any variables or predicates.

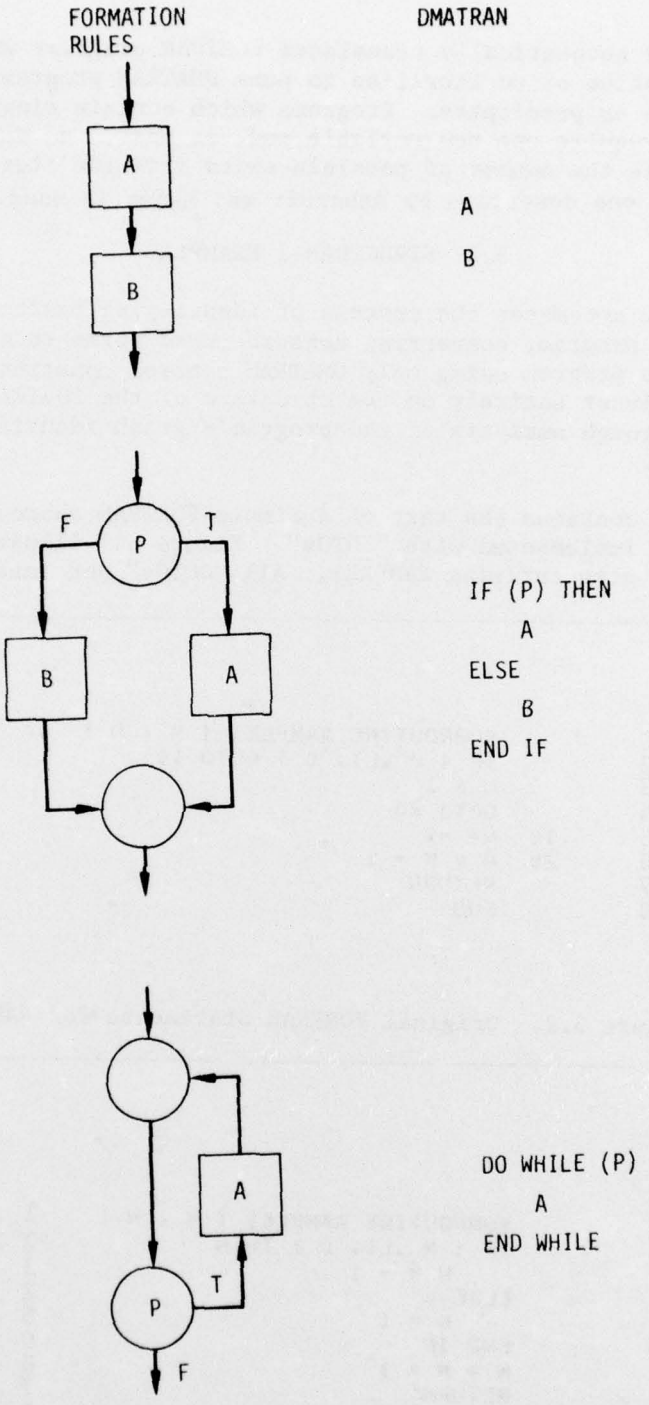

Figure 3.1. Formation Rules and Related DMATRAN Constructs

NOTE: Labeled circles are decision nodes; blank circles are junctions.

STRUCTRAN-2 automatically translates FORTRAN programs which contain single exit iteration or no iteration to pure DMATRAN programs without adding any new variables or predicates. Programs which contain single entry/multiple exit iterations require one new variable and, at most, n additional predicates where n is the number of possible exits from the iteration. A technique similar to one described by Ashcroft and Manna is used.

3.2 STRUCTRAN-2 EXAMPLES

STRUCTRAN-2 automates the process of identifying basic structured forms within a FORTRAN program, converting nonstructured forms to structured form and rewriting the program using only DMATRAN control constructs. This technique is based almost entirely on the structure of the FORTRAN program. A complete and thorough analysis of the program's graph identifies well-nested structured forms.

Figure 3.2 contains the text of a simple FORTRAN subroutine XAMPLE1 whose control is implemented with "GOTOs". Figure 3.3 illustrates the result of automatically structurizing XAMPLE1. All "GOTOs" and labels have been

```
1           SUBROUTINE XAMPLE1 ( M , N )
2           IF ( M .LT. 0 ) GOTO 10
3           N = 1
4           GOTO 20
5       10  N= -1
6       20  M = M + 1
7           RETURN
8           END
```

Figure 3.2. Original FORTRAN Statements for XAMPLE1

```
SUBROUTINE XAMPLE1 ( M , N )      1
IF ( M .LT. 0 ) THEN              2
    N = - 1                       3
ELSE                              2
    N = 1                         5
END IF                            0
M = M + 1                         6
RETURN                            7
END                               8
```

Figure 3.3. Structurized DMATRAN Statements for XAMPLE1

replaced with an IF...THEN...ELSE...END IF clause which determines whether statement 3 or statement 5 will be executed based on the same criterion used in the original "GOTO" version. The rightmost column of numbers in Fig. 3.3 refers to the original statement number of the rewritten non-control statements. XAMPLE1 is an instance of a "GOTO" program whose graph is structured, and hence can be translated without any duplicated code or additional variables and predicates.

Figure 3.4 illustrates the graph of a program that uses "GOTOs" to borrow code segment "A" under different sets of conditions. This is an unstructured graph which must be modified to be translated to a "GOTO"-free form. Figure 3.5 shows the way STRUCTRAN-2 reconfigures borrowed code into a structured graph. The code segment "A" is included twice in the structured reconfiguration. If code segment "A" is a large piece of code, it will be replaced with an INVOKE statement which results in "A" being executed. This technique reduces the code expansion which can result in transforming programs into a "GOTO"-free form. Figure 3.6 contains a subroutine XAMPLE2 which uses "GOTOs" to borrow statement number 6. The structurized version (Fig. 3.7) indicates that statement number 6 has been rewritten twice. (The rightmost column of numbers indicates the statement number in the original version.) The nested structure IF...THEN...ELSE...END IF is exactly that indicated in Fig. 3.4.

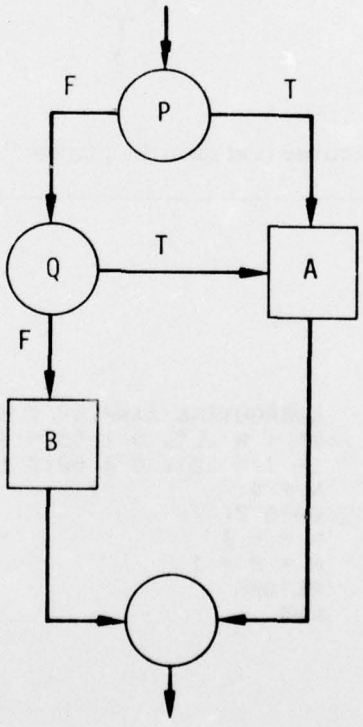

Figure 3.4. Borrowing "A" with GOTO's

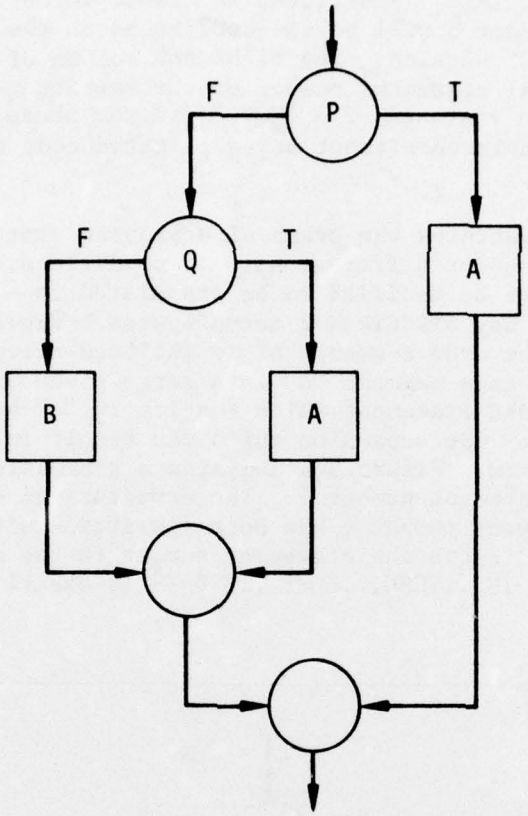

Figure 3.5. Structured Version Requires "A" to Appear Twice

```
1               SUBROUTINE XAMPLE2 ( M , N )
2               IF ( M .LT. 0 ) GOTO 10
3               IF ( M .GT. 0 ) GOTO 10
4               N = 0
5               GOTO 20
6         10    N = - 1
7         20    M = M + 1
8               RETURN
9               END
```

Figure 3.6. Original FORTRAN Statements for XAMPLE2

```
SUBROUTINE XAMPLE2 ( M , N )                1
IF ( M .LT. 0 ) THEN                        2
    N = - 1                                 6
ELSE                                        2
    IF ( M .GT. 0 ) THEN                    3
        N = - 1                             6
    ELSE                                    3
        N = 0                               4
    END IF                                  0
END IF                                      0
M = M + 1                                   7
RETURN                                      8
END                                         9
```

Figure 3.7. Structurized DMATRAN Statements for XAMPLE2

The graph of a program which contains a single entry/single exit (SE/SE) iteration is shown in Fig. 3.8. The DMATRAN code which may be directly written to implement this graph is shown in Fig. 3.9. The DMATRAN version presents all the structural information included in the program graph. Figure 3.10 contains the FORTRAN text of a module composed of nested SE/SE iterations. This fact is not evident from the FORTRAN "GOTOs" which implement the iterations. The DMATRAN version (Fig. 3.11) automatically produced by STRUCTRAN-2 does represent the nested iterations. Since each iteration is SE/SE, and "GOTOs" are not used to borrow any code, this program was directly rewritten without duplicating any statements or adding any variables.

Figure 3.12 depicts the form of a single entry/two exit iteration. To rewrite this form in a structured manner, the graph of the program must be altered and a new variable and predicate added to make the logically possible flows in the new graph execute the same structurally possible flows which occurred in the original graph. The manner in which this is automatically performed is indicated in Fig. 3.13. A single entry/multiple exit (SE/ME) iteration implemented in "GOTOs" is shown in Fig. 3.14. The automatically translated version is shown in Fig. 3.15. Two assignment statements, one new variable, a new predicate, and a modified predicate have been added to allow this program to be written in DMATRAN. The technique used in STRUCTRAN-2 automatically rewrites all SE/ME iterations.

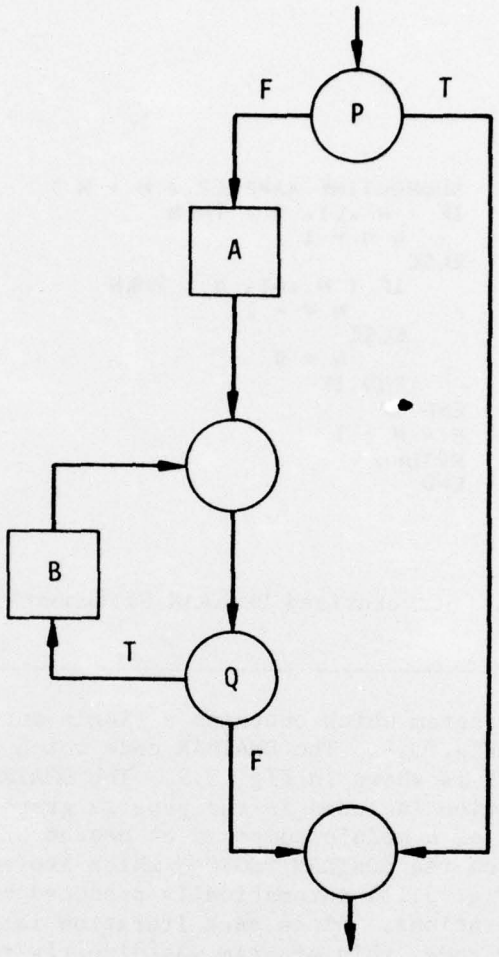

Figure 3.8. Well-Structured Graph with Nested Iteration

```
IF (.NOT. P) THEN
    A
    DO WHILE
        B
    END WHILE
END IF
```

Figure 3.9. DMATRAN Code Corresponding to Fig. 3.8

```
 1              SUBROUTINE XAMPLE3 ( K , L , M , N )
 2              N = 0
 3              IF ( M .LE. 0 ) GOTO 10
 4              I = 1
 5        50    IF ( I .GT. M ) GOTO 10
 6              IF ( I .LE. 0 ) GOTO 30
 7              J = 1
 8        40    IF ( J .GT. L ) GOTO 30
 9              N = N + K
10              J = J + 1
11              GOTO 40
12        30    N = N + K
13              I = I + 1
14              GOTO 50
15        10    CONTINUE
16              RETURN
17              END
```

Figure 3.10. Original FORTRAN Statements for XAMPLE3

```
SUBROUTINE XAMPLE3 ( K , L , M , N )                    1
N = 0                                                   2
IF ( M .GT. 0 ) THEN                                    3
    I = 1                                               4
    DO WHILE ( I .LE. M )                               5
        IF ( I .GT. 0 ) THEN                            6
            J = 1                                       7
            DO WHILE ( J .LE. L )                       8
                N = N + K                               9
                J = J + 1                              10
            END WHILE                                   0
        END IF                                          0
        N = N + K                                      12
        I = I - 1                                      13
    END WHILE                                           0
END IF                                                  0
RETURN                                                 16
END                                                    17
```

Figure 3.11. Sturcturized DMATRAN Statements for XAMPLE3

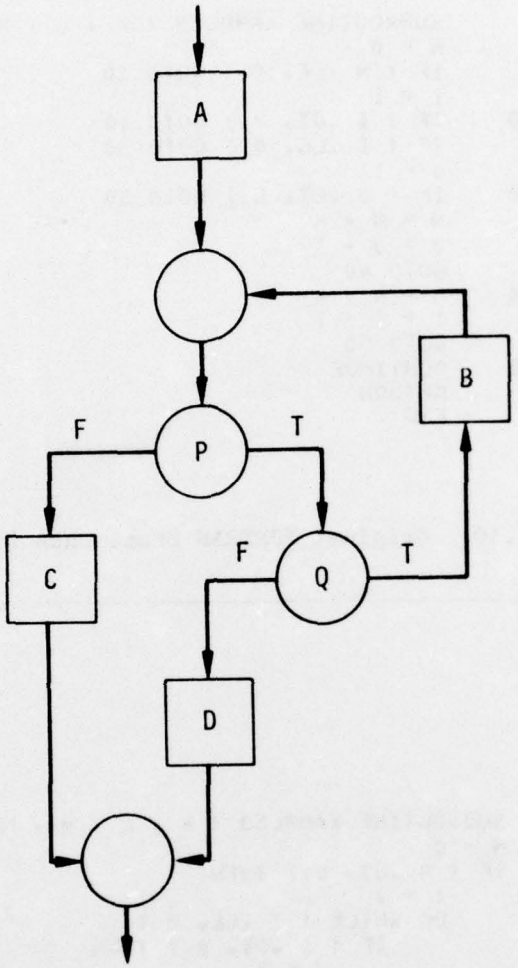

Figure 3.12. One Entry/Two Exit Iteration Expressed with GOTO's

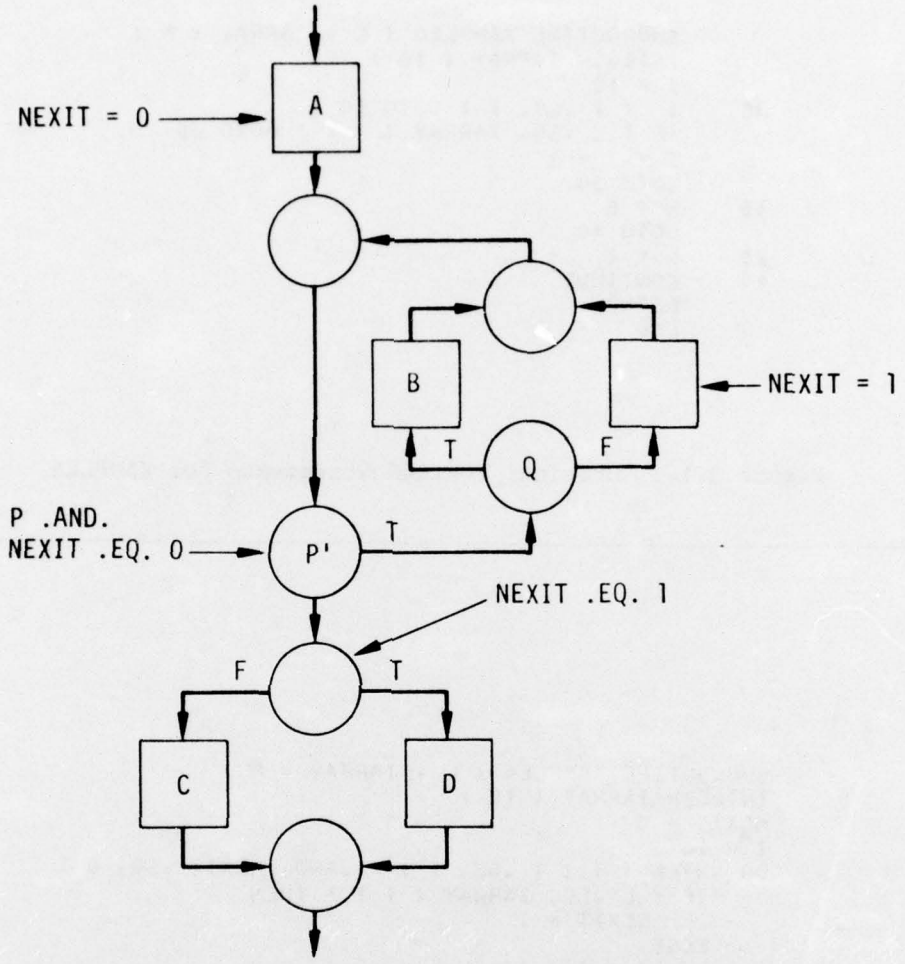

Figure 3.13. Structured Version with Variable and Predicates Added

```
 1                SUBROUTINE XAMPLE4 ( L , IARRAY , N )
 2                INTEGER IARRAY ( 10 )
 3                I = 10
 4          30    IF ( I .LT. 1 ) GOTO 10
 5                IF ( L .EQ. IARRAY ( I ) ) GOTO 20
 6                I = I - 1
 7                GOTO 30
 8          10    N = 0
 9                GOTO 40
10          20    N = 1
11          40    CONTINUE
12                RETURN
13                END
```

Figure 3.14. Original FORTRAN Statements for XAMPLE4

```
SUBROUTINE XAMPLE4 ( L , IARRAY , N )                                  1
INTEGER IARRAY ( 10 )                                                  2
NEXIT = 0                                                              0
I = 10                                                                 3
DO WHILE ( ( ( I .GE. 1 ) ) .AND. NEXIT .EQ. 0 )                       4
    IF ( L .EQ. IARRAY ( I ) ) THEN                                    5
        NEXIT = 1                                                      0
    ELSE                                                               5
        I = I - 1                                                      6
    END IF                                                             0
END WHILE                                                              0
IF ( NEXIT .EQ. 1 ) THEN                                               4
    NEXIT = 0                                                          0
    N = I                                                             10
ELSE                                                                   4
    N = 0                                                              8
END IF                                                                 0
RETURN                                                                12
END                                                                   13
```

Figure 3.15. Structurized DMATRAN Statements for XAMPLE4

3.3 STRUCTRAN-2 GRAPHICAL ANALYSIS

The basic steps in restructuring FORTRAN modules are to identify statement types and symbols in the original text, extract the module's graph from this information, perform a sophisticated analysis of the program graph, and use the results of this analysis along with the other information known about the module to rewrite it in a structured form. The algorithm implemented by the original module will also be implemented by its structured version. The restructured program essentially executes the same statements in the same order as the original program. The advantage of the structured version is that it can be read from top to bottom. The restructured version is easier to understand, yet it performs exactly like the original version.

The graphical analysis performed by STRUCTRAN-2 is based on two properties of the three basic structured forms. Each of the basic forms can be characterized as a single entry/single exit subgraph. Also the graph of a structured program which uses only the basic forms is built up of well-nested single entry/single exit subgraphs. The natural way to graphically analyze structured programs is to rely on these properties. The program graph for structured modules is obtained in the same way as for unstructured modules. Basic forms which do not contain any other forms can be readily identified in the program graph. Since each basic form is a single entry/single exit subgraph, the complexity of the graph can be reduced by replacing an identified form with a single edge which goes from its single entry to its single exit. It is then possible to repeat this process and identify basic forms which previously were composed of more complex structures than the edges which have been inserted. The process terminates when the resulting graph consists of a single edge from the entry of the module to the exit from the module. As the reductions are being performed, it is essential to maintain a data structure which indicates the basic form identified and the single entry/single exit subgraphs of which it is composed. The natural data structure for this information is a hierarchy of SE/SE subgraphs. The structure of this hierarchy corresponds directly with the well-nested, indented representation of the text as a structured program.

STRUCTRAN-2 extends the identification of forms to include occurrences of borrowed code, iterations with multiple entries or multiple exits, and iterations which will be executed at least once. These constructs are represented in the hierarchy describing the graph as composed of the basic structured forms. When this process is applied to an unstructured FORTRAN program, the resulting hierarchy of SE/SE subgraphs represents the original program in a structured way. It is this representation which is used by STRUCTRAN-2 to translate the program to a structured form.

Experience with STRUCTRAN-2 indicates that the most commonly occurring forms in unstructured FORTRAN programs are the three basic forms. Most unstructured FORTRAN programs do, however, contain instances of borrowing code with GOTOs. Approximately 50% of FORTRAN programs contain one or more single entry/multiple exit iterations. Fewer than 10% of FORTRAN modules contain multiple entry iterations.

REFERENCES

1. C. Boehm and G. Jacopini, "Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules," CACM 9, 5, May 1966.

2. E. F. Miller, Jr., "Extensions to FORTRAN to Support Structured Programming," SIGPLAN Notices 8, 6, 1973.

3. E. Ashcroft and Z. Manna, "The Translation of GOTO Programs to WHILE Programs," Information Processing 71.

4. J. E. Sullivan, "Measuring the Complexity of Computer Software," MITRE Corp. MTR-2648, June 1973.