**REPORT DOCUMENTATION PAGE**

<table>
<thead>
<tr>
<th>REPORT NUMBER</th>
<th>RADC-TR-78-268, Vol III (of three)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TITTLE (and subtitle)</td>
<td>FORTRAN AUTOMATED VERIFICATION SYSTEM (FAVS). DMATRAN User's Guide</td>
</tr>
<tr>
<td>READ INSTRUCTIONS</td>
<td>BEFORE COMPLETING FORM</td>
</tr>
</tbody>
</table>

**READ INSTRUCTIONS**

<table>
<thead>
<tr>
<th>RECIPIENT'S CATALOG NUMBER</th>
<th></th>
</tr>
</thead>
</table>

**AUTHOR(S)**

- A. Melton
- M. Andrews

**PERFORMING ORGANIZATION NAME AND ADDRESS**

General Research Corporation
P.O. Box 6770
Santa Barbara CA 93111

**CONTRACT OR GRANT NUMBER(S)**

F30602-76-C-0636

**PERFORMING ORGANIZATION REPORT NUMBER**

N/A

**CONTROLLED OFFICE NAME AND ADDRESS**

Rome Air Development Center (ISIE)
Griffiss AFB NY 13441

**PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS**

63701B
32000320

**CONTRACT OR GRANT NUMBER(S)**

N/A

**MONITORING AGENCY NAME AND ADDRESS (IF DIFFERENT FROM CONTROLLING OFFICE)**

Same

**SECURITY CLASS. (OF THIS REPORT)**

UNCLASSIFIED

**SECURITY CLASS. (OF THIS PAGE (SCHD) (of this Report)**

UNCLASSIFIED

**DISTRIBUTION STATEMENT (of this Report)**

Approved for public release; distribution unlimited.

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

Same

**SUPPLEMENTARY NOTES**

RADC Project Engineer: Frank S. Lamonica (ISIE)

**KEY WORDS**

Computer Software
FABS
Software Testing
Automated Verification System
Software Verification
Software Documentation

**ABSTRACT**

DMATRAN is a structured programming language which provides the logical constructs that are necessary to write structured code in FORTRAN. The additional control constructs in DMATRAN are:

- A structured IF construct which allows execution of a group of statements if a logical expression is true (DO IF...END IF) or until a logical expression becomes true (DO UNTIL...END UNTIL), and
- A non-iterative CASE structure which begins with an integer expression which is
Item 20 (Cont'd)

evaluated and then compared with integers in a list of CASE statements that follow. Execution of a group of statements following the matching CASE statement is then initiated. The DMATRAN language also contains an INVOKE...BLOCK...END BLOCK construct which provides a form of internal subroutine capability as well as a way to reduce overhead costs by eliminating duplicate sections of code.

This DMATRAN User's Guide describes and illustrates each structured construct. It also explains how to use the DMATRAN precompiler which translates DMATRAN into compilable FORTRAN code. The DMATRAN precompiler provides additional source text editing and display features including page ejection, suppressing source listing, and changing the syntax of DMATRAN statements; the DMATRAN commands for these capabilities are also described.

The DMATRAN precompiler has been installed on the HIS 6180 GCOS and MULTICS computer systems at the Rome Air Development Center, Griffiss AFB, New York, and on the UNIVAC 1100/42 computer systems at the Defense Mapping Agency Aerospace Center in St. Louis, Missouri, and the Defense Mapping Agency Topographic Center in Washington, D.C.
## CONTENTS

<table>
<thead>
<tr>
<th>SECTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>INTRODUCTION</td>
<td>1-1</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>DMATRAN CONTROL CONSTRUCTS</td>
<td>2-1</td>
</tr>
<tr>
<td>2.1 IF...THEN...ELSE...END IF</td>
<td>2-3</td>
</tr>
<tr>
<td>2.2 DO WHILE...END WHILE</td>
<td>2-5</td>
</tr>
<tr>
<td>2.3 DO UNTIL...END UNTIL</td>
<td>2-5</td>
</tr>
<tr>
<td>2.4 CASE OF...CASE...CASE ELSE...END CASE</td>
<td>2-8</td>
</tr>
<tr>
<td>2.5 BLOCK...END BLOCK AND INVOKE</td>
<td>2-11</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>USING THE DMATRAN PRECOMPILER</td>
<td>3-1</td>
</tr>
<tr>
<td>3.1 DMATRAN Input</td>
<td>3-1</td>
</tr>
<tr>
<td>3.2 DMATRAN Indented Listing</td>
<td>3-2</td>
</tr>
<tr>
<td>3.3 FORTRAN Output</td>
<td>3-3</td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>DMATRAN CONSTRAINTS</td>
<td>4-1</td>
</tr>
<tr>
<td>4.1 SYNTAX</td>
<td>4-1</td>
</tr>
<tr>
<td>4.2 DO UNTIL</td>
<td>4-1</td>
</tr>
<tr>
<td>4.3 CASE</td>
<td>4-1</td>
</tr>
<tr>
<td>4.4 BLOCK Construct</td>
<td>4-1</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>DISPLAY COMMANDS</td>
<td>5-1</td>
</tr>
<tr>
<td>5.1 Command Form</td>
<td>5-1</td>
</tr>
<tr>
<td>5.2 Source Listing</td>
<td>5-2</td>
</tr>
<tr>
<td>5.3 Double-Space Around Comments</td>
<td>5-2</td>
</tr>
<tr>
<td>5.4 Keyword Recognition</td>
<td>5-2</td>
</tr>
<tr>
<td>5.5 Page Ejection</td>
<td>5-5</td>
</tr>
<tr>
<td>5.6 Extended Comment</td>
<td>5-5</td>
</tr>
</tbody>
</table>
## CONTENTS CONT.

<table>
<thead>
<tr>
<th>SECTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPENDIX A SUMMARY OF DMATRAN STATEMENTS AND COMMANDS</td>
<td>A-1</td>
</tr>
<tr>
<td>APPENDIX B FILE DESCRIPTIONS</td>
<td>B-1</td>
</tr>
<tr>
<td>APPENDIX C JOB STREAMS</td>
<td>C-1</td>
</tr>
<tr>
<td>INDEX</td>
<td>I-1</td>
</tr>
<tr>
<td>NO.</td>
<td>ILLUSTRATIONS</td>
</tr>
<tr>
<td>-----</td>
<td>---------------------------------------------------</td>
</tr>
<tr>
<td>1.1</td>
<td>DMATRAN Precompiler</td>
</tr>
<tr>
<td>2.1</td>
<td>DMATRAN Control Constructs</td>
</tr>
<tr>
<td>2.2</td>
<td>IF...THEN...END IF Construct</td>
</tr>
<tr>
<td>2.3</td>
<td>IF...THEN...ELSE...END IF Construct</td>
</tr>
<tr>
<td>2.4</td>
<td>DO WHILE...END WHILE Construct</td>
</tr>
<tr>
<td>2.5</td>
<td>DO UNTIL...END UNTIL</td>
</tr>
<tr>
<td>2.6</td>
<td>CASE OF..CASE..CASE ELSE...END CASE</td>
</tr>
<tr>
<td>2.7</td>
<td>DMATRAN Case Construct</td>
</tr>
<tr>
<td>2.8</td>
<td>BLOCK...END BLOCK and INVOKE Construct</td>
</tr>
<tr>
<td>2.9</td>
<td>BLOCK Cross-Reference Report</td>
</tr>
<tr>
<td>3.1</td>
<td>DMATRAN Source Input</td>
</tr>
<tr>
<td>3.2</td>
<td>DMATRAN Indented Listing</td>
</tr>
<tr>
<td>3.3</td>
<td>Translated FORTRAN</td>
</tr>
<tr>
<td>5.1</td>
<td>DMATRAN Keyword Syntax</td>
</tr>
<tr>
<td>5.2</td>
<td>Listing of DMATRAN Keyword Example</td>
</tr>
</tbody>
</table>
INTRODUCTION

DMATRAN is an extension to FORTRAN that simplifies structured programming by providing a convenient syntax for writing structured programming control constructs. The DMATRAN precompiler translates the DMATRAN statements into standard FORTRAN while passing all other statements unchanged to a file which can then be compiled by the FORTRAN compiler. This sequence is illustrated in Fig. 1.1. In addition to the translation, the precompiler checks the control structure for proper use of DMATRAN control structures and issues error messages if violations occur.

The precompiler provides the following additional features to improve code production:

1. Indented listing of the DMATRAN source code.

2. Editing functions which include in-line comments, double-spacing around comments, indentation control, selective page ejection, and selective suppression of the source listings.
Figure 1.1. DMATRAN Precompiler
DMATRAN replaces FORTRAN control statements with the following control statement constructs:

- **IF...THEN...ELSE...END IF** - provides block structuring of conditionally executable sequences of statements.
- **DO WHILE...END WHILE** - permits iteration of a code segment while a specified condition remains true.
- **CASE OF...CASE...CASE ELSE...END CASE** - allows multiple choices for program action selection.
- **DO UNTIL...END UNTIL** - permits iteration until a specified condition becomes true.
- **INVOKE<name>...BLOCK<name>...END BLOCK** - provides 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 which is processed by the DMATRAN precompiler. DMATRAN statements are converted by the precompiler to equivalent FORTRAN statements, and the resulting file can be compiled by the FORTRAN compiler in the normal manner. A description, flowchart, and example of each DMATRAN control construct is provided in this section. The DMATRAN constructs are shown in Fig. 2.1.
Figure 2.1. DMATRAN Control Constructs
2.1 IF...THEN...ELSE...END IF

The IF...THEN...ELSE...END IF construct provides block structuring of conditionally executable statements. The basic form of this construct is IF...THEN...END IF, illustrated in Fig. 2.2. If <expression> is true, control transfers to the first statement within the construct; otherwise, the statement immediately following the END IF will be executed. Use of the ELSE statement is optional. If the ELSE is present and <expression> is false, the statements following the ELSE are executed. This construct is illustrated in Fig. 2.3.

![Figure 2.2. IF...THEN...END IF Construct](image-url)
IF (\textit{<EXPRESSION>}) \textbf{THEN}

\hspace{1em} \textbf{: \textbf{STATEMENTS TO EXECUTE IF \textit{<EXPRESSION>} IS TRUE}}

\hspace{1em} \textbf{ELSE}

\hspace{2em} \textbf{STATEMENTS TO EXECUTE IF \textit{<EXPRESSION>} IS FALSE}

\hspace{1em} \textbf{END IF}

\hspace{1em}\hspace{9em} \begin{array}{c}
\text{IF}
\end{array}
\hspace{1em} \text{\textit{<EXPRESSION>}}

\hspace{1em}\hspace{9em} \begin{array}{c}
\text{\textbf{THEN}}
\end{array}
\hspace{1em} \text{\textbf{NOT}.}
\hspace{1em} \text{\textit{<EXPRESSION>}}

\hspace{1em}\hspace{9em} \begin{array}{c}
\text{\textbf{ELSE}}
\end{array}
\hspace{1em} \text{\textbf{STATEMENTS TO EXECUTE IF \textit{<EXPRESSION>} IS FALSE}}
\hspace{1em} \text{\textbf{STATEMENTS TO EXECUTE IF \textit{<EXPRESSION>} IS TRUE}}
\hspace{1em} \text{\textbf{END IF}}

\begin{figure}
\centering
\begin{tikzpicture}
\node (input) at (0,0) {IF \textit{<EXPRESSION>}};
\node (then) [below of=input] {\textbf{THEN}};
\node (not) [below of=then] {\textbf{NOT}. \textit{<EXPRESSION>}};
\node (else) [below of=not] {\textbf{ELSE}};
\node (false) [below of=else] {\textbf{STATEMENTS TO EXECUTE IF \textit{<EXPRESSION>} IS FALSE}};
\node (true) [right of=false] {\textbf{STATEMENTS TO EXECUTE IF \textit{<EXPRESSION>} IS TRUE}};
\node (end) [below of=true] {\textbf{END IF}};
\draw (input) -- (then);
\draw (then) -- (not);
\draw (not) -- (else);
\draw (else) -- (false) -- (false -- (true));
\draw (true) -- (end);
\end{tikzpicture}
\caption{IF...THEN...ELSE...END IF Construct}
\end{figure}

\textbf{FUNCTION SINC} (\textit{X})
\begin{align*}
\text{IF} \ (\textit{X} . \text{EQ} . \text{D}) \ \text{THEN} & \\ \text{\textbf{SINC}} = \ 1. \\
\text{ELSE} & \\ \textbf{SINC} = \ \text{SIN}(\textit{X}) / \textit{X} \\
\text{END IF} & \\
\text{RETURN} & \\
\text{END}
\end{align*}
2.2 DO WHILE...END WHILE

The DO WHILE...END WHILE construct indicates a repetitive operation which is to be performed zero or more times. Execution occurs in the following manner:

1. The value of <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 <expression> is checked again, with the same consequences as in (1).

Figure 2.4 illustrates the form and meaning of this construct. It is important to note that no initialization or incrementing operations are caused by the DO WHILE...END WHILE construct. Initialization must be explicitly performed prior to entering the loop, and the iteration variables must be explicitly modified on each pass through the loop.

2.3 DO UNTIL...END UNTIL

The DO UNTIL...END UNTIL construct is like a FORTRAN DO-LOOP in that it is performed at least once and has a single exit at the bottom of the loop, and like a DO WHILE...END WHILE in that no initialization or incrementing operations are caused by this construct. Initialization must be performed prior to entering the loop, and iteration variables must be modified on each pass through the loop. Figure 2.5 illustrates this construct.

The statements enclosed within the DO UNTIL and the END UNTIL are executed at least once. Then <expression> is evaluated and, if false, iteration and evaluation of the expression continue until it is true. At that time execution of the statements following the END UNTIL begins.
INITIALIZATION STATEMENTS
DO WHILE (<EXPRESSION>)
  STATEMENTS TO EXECUTE IF <EXPRESSION> IS TRUE
END WHILE

FUNCTION SQRT( A )
X = A
DO WHILE( ABS(X-A/X) .GT. 1.E-6 )
  X = (X+A/X)/2
END WHILE
SQRT = X
RETURN
END

Figure 2.4. DO WHILE...END WHILE Construct
INITIALIZATION STATEMENTS

DO UNTIL (<EXPRESSION>)
  : STATEMENTS TO EXECUTE IF <EXPRESSION> IS FALSE
  :
END UNTIL

FUNCTION CONVRG(XINIT, EPS, F)
EXTERNAL F
X = XINIT
DO UNTIL (ABS(X - XOLD) .LE. EPS)
  XOLD = X
  X = F(X)
END UNTIL
CONVRG = X
RETURN
END

Figure 2.5. DO UNTIL...END UNTIL
2.4 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
CASE OF...CASE...CASE ELSE...END CASE.

Figure 2.6 illustrates the CASE construct. I, J, and N represent integers of positive value. They may be in any order, and there is no limit to how many integers may be listed.

The value of <integer expression> is computed, and if any of the specified integers in the CASE list are equal to the value of <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, then the block of statements following 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.

A listing containing an example of the CASE construct is shown in Fig. 2.7.
CASE OF (<INTEGER EXPRESSION>)
CASE (I)
  .  BLOCK OF STATEMENTS
CASE (J)
  .  BLOCK OF STATEMENTS
  ...
CASE (N)
  .  BLOCK OF STATEMENTS
CASE ELSE
  .  BLOCK OF STATEMENTS
END CASE

Figure 2.6. CASE OF..CASE..CASE ELSE...END CASE
SUBROUTINE XAMPL (ITYPE,NPARS)

::
CASE OF (ITYPE)
CASE (3)
   CALL GETCRD(ITYPE)
CASE (5)
   JTYPE = ITYPE + 3
   CALL STRUCT(JTYPE)
CASE (9)
   CALL IBALPR(ITYPE,NPARS)
CASE ELSE
   CALL ERROR
END CASE
RETURN
END

Figure 2.7. DMATRAN Case Construct
2.5 BLOCK...END BLOCK and INVOKE

The BLOCK...END BLOCK construct provides a form of internal subroutine capability in DMATRAN source programs. This construct is an internal procedure which has access to all variables in the routine which contains it. A BLOCK...END BLOCK is executed only if it is referred to with an INVOKE statement which specifies its name. The form for this construct is:

```
INVOKE (<name>)
...
BLOCK (<name>)
...
END BLOCK
```

where `<name>` is any string of characters (i.e., COMPUTE LENGTH, PRINT CURRENT STATUS, or COMPUTE NEW ARRAY ELEMENT). The name of a BLOCK may be arbitrarily long, so that the name can have mnemonic significance. All characters are significant after the first non-blank and before the last non-blank. The name of a BLOCK is known throughout the entire routine in which it is contained. Figure 2.8 illustrates this construct.

As the flowchart for this construct indicates, it is a single-entry (the BLOCK statement), single-exit (the END BLOCK statement) section of code. An INVOKE statement causes control to transfer to the named BLOCK statement, and the matching END BLOCK statement causes control to transfer back to the statement after the INVOKE. More than one INVOKE for a given BLOCK...END BLOCK construct is allowed. Though BLOCK...END BLOCK constructs can be nested, no recursion is allowed in the invoking of BLOCKS (i.e., a BLOCK cannot directly or indirectly invoke itself). BLOCKS cannot be invoked from an external routine, nor can they be passed as a parameter to another routine. BLOCK constructs may be placed before or after the RETURN statement.
**Figure 2.8. BLOCK...END BLOCK and INVOKE Construct**

```
SUBROUTINE MLTPLY(A,B,C,N)
DIMENSION A(10,10),B(10,10),C(10,10)
I = 1
DO WHILE ( I .LE. N )
   J = 1
   DO WHILE ( J .LE. N )
      INVOKE ( COMPUTE NEW ARRAY ELEMENT )
      J = J + 1
   END WHILE
   I = I + 1
END WHILE
BLOCK ( COMPUTE NEW ARRAY ELEMENT )
   S = 0.0
   K = 1
   DO WHILE ( K .LE. N )
      S = S + A(I,K) * B(K,J)
      K = K + 1
   END WHILE
   C(I,J) = S
END BLOCK
RETURN
END
```
When the BLOCK...END BLOCK construct is used, available space for storing additional BLOCK names is indicated in the BLOCK Cross-Reference Report (Fig. 2.9) which follows the DMATRAN source listing for each module. The total number of characters in all BLOCK names used in one subroutine cannot exceed 1,000. The number of INVOKEs and the number of BLOCKs varies with each module as it is dependent on the size of the names of the BLOCKs and the number of invocations. When the maximum has been reached, a message is printed on the DMATRAN listing indicating a BLOCK name table overflow.

<table>
<thead>
<tr>
<th>BLOCK NAME</th>
<th>DEFINED</th>
<th>INVOKEd</th>
</tr>
</thead>
<tbody>
<tr>
<td>INITIALIZE BOUNDARIES</td>
<td>36</td>
<td>27</td>
</tr>
<tr>
<td>SORT INCOMING TABLE IN SEGMENTS</td>
<td>54</td>
<td>29</td>
</tr>
<tr>
<td>PARTITION SORT SEGMENTED TABLE</td>
<td>80</td>
<td>30</td>
</tr>
<tr>
<td>SORT ENTIRE TABLE</td>
<td>114</td>
<td>32</td>
</tr>
<tr>
<td>SWITCH INCOMING TABLE BLOCKS TO SORTED</td>
<td>127</td>
<td>34</td>
</tr>
<tr>
<td>TRANSFER FROM TABLE FOR FIRST SORT</td>
<td>133</td>
<td>59 116</td>
</tr>
<tr>
<td>SLT UP SORT BY SEGMENTS</td>
<td>151</td>
<td>63</td>
</tr>
<tr>
<td>STORE SALLEST IN TEMPORARY TABLE</td>
<td>164</td>
<td>90 96 110 164</td>
</tr>
<tr>
<td>MOVE FROM A SEGMENT TO ARRAY</td>
<td>189</td>
<td>92 157</td>
</tr>
<tr>
<td>FIND WHICH SEGMENT</td>
<td>167</td>
<td>95 102 163</td>
</tr>
</tbody>
</table>

645 WORDS LEFT

Figure 2.9. BLOCK Cross-Reference Report
There are many reasons for using the BLOCK construct. For example, the overhead in calling subroutines is often very high; and, in addition, variables in FORTRAN and DMATRAN programs must be passed as parameters or placed in COMMON to be accessible to both the calling routine and the subroutine which is called. Often a subroutine references only variables which are already in the calling routine. Using a BLOCK structure as an internal subroutine eliminates the need to provide a means of accessing these variables.

Another way the BLOCK construct can improve overhead costs is by elimination of duplicate sections of code. Since the same BLOCK can be invoked by more than one INVOKE statement, code can be made more efficient by putting identical sections of code into BLOCK structures. The following example illustrates the use of BLOCKs to avoid code duplication.
\( S_1 \) and \( S_2 \) in the following code represent two sets of statements. The use of a BLOCK in Method 2 below eliminates the need for duplicating code.

**Method 1:**

\[
\text{IF}(A) \text{ THEN }
\text{IF}(C) \text{ THEN }
\quad S_1
\text{ ELSE }
\quad S_2
\text{ END IF }
\text{ELSE }
\text{IF}(D) \text{ THEN }
\quad S_2
\text{ ELSE }
\quad S_1
\text{ END IF }
\text{END IF }
\]

**Method 2:**

\[
\text{IF}(A) \text{ THEN }
\text{IF}(C) \text{ THEN }
\quad \text{INVOKE}(\text{BLOCK-A})
\text{ ELSE }
\quad \text{INVOKE}(\text{BLOCK-B})
\text{ END IF }
\text{ELSE }
\text{IF}(D) \text{ THEN }
\quad \text{INVOKE}(\text{BLOCK-B})
\text{ ELSE }
\quad \text{INVOKE}(\text{BLOCK-A})
\text{ END IF }
\text{END IF }
\]

where the BLOCKs are defined as:

\[
\text{BLOCK}(\text{BLOCK-A})
\quad S_1
\text{ END BLOCK }
\]

and

\[
\text{BLOCK}(\text{BLOCK-B})
\quad S_2
\text{ END BLOCK }
\]

2-15
3 USING THE DMATRAN PREPROCESSOR

3.1 DMATRAN INPUT

Figure 3.1 illustrates a DMATRAN source program with embedded FORTRAN statements ready for input to the DMATRAN precompiler. The DMATRAN source code begins in column 7 and is not indented. More than one module may be processed in each DMATRAN run.

```
SUBROUTINE EXAM1 (INFO,LENGTH)
    ! ILLUSTRATION OF DMATRAN SYNTAX
    IF (INFO.LE.10 .AND. LENGTH.GT.0) THEN
        CALL CALLER (INFO)
    ELSE
        LENGTH=50
        END IF
        CASE OF (INFO+6)
        CASE (14)
            LENGTH=LENGTH-INFO
        CASE (17)
            DO WHILL (INFO.LT.20)
                DO UNTIL (LENGTH.LE.INFO)
                    INVOKE (COMPUTE LENGTH)
                    IF (LENGTH.GE.10) THEN
                        INVOKE (PRINT-RESULTS)
                    END IF
                    END UNTIL
                    INFO=INFO+1
                    END WHILE
                END CASE
                END CASE
            END BLOCK (PRINT-RESULTS)
            WRITE (6,1)INFO,LENGTH
            1 FORMAT (10X,15,20X,15)
        END BLOCK
    END CASE
END BLOCK (COMPUTE LENGTH)
LENGTH = LENGTH -10
END BLOCK
RETURN
END
```

Figure 3.1. DMATRAN Source Input
3.2 DMATRAN INDENTED LISTING

Figure 3.2 illustrates the automatically indented DMATRAN listing which resulted from processing the input shown in Fig. 3.1. The heading contains information from the first card of the routine being processed, as well as a page number. The leftmost column of numbers refers to the successive statements of the DMATRAN source deck. The nesting depth of each indented statement is indicated next to the statement number. Structural visibility is enhanced by connecting related DMATRAN statements with vertical dots. The dots assist in tracing paths through the program, identifying the statement number for a given line, and debugging improperly formed DMATRAN control constructs. Structural errors are indicated by error diagnostics in the DMATRAN listing. Sequence information following column 72 of the DMATRAN source cards is included on the right side of the DMATRAN listing.

<table>
<thead>
<tr>
<th>SLG NEST SOURCE</th>
<th>SUBROUTINE EXAPL (INFO,LENGTH)</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SUBROUTINE EXAPL (INFO,LENGTH)</td>
<td>EXAPL1</td>
</tr>
<tr>
<td>2</td>
<td>ILLUSTRATION OF DMATRAN SYNTAX</td>
<td>EXAPL2</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>EXAPL3</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>EXAPL4</td>
</tr>
<tr>
<td>5</td>
<td>IF (INFO.LE.10 .AND. LENGTH.GT.6) THEN</td>
<td>EXAPL5</td>
</tr>
<tr>
<td>6</td>
<td>CALL CALLER (INFO) ELSE</td>
<td>EXAPL6</td>
</tr>
<tr>
<td>7</td>
<td>LENGTH=50 END IF</td>
<td>EXAPL7</td>
</tr>
<tr>
<td>8</td>
<td>CASE (INFO+6) CASE (14)</td>
<td>EXAPL8</td>
</tr>
<tr>
<td>9</td>
<td>CASE (17)</td>
<td>EXAPL9</td>
</tr>
<tr>
<td>10</td>
<td>LENGTH=LENGTH+INFO</td>
<td>EXAPL10</td>
</tr>
<tr>
<td>11</td>
<td>IF (LENGTH.GT.INFO) THEN</td>
<td>EXAPL11</td>
</tr>
<tr>
<td>12</td>
<td>END IF</td>
<td>EXAPL12</td>
</tr>
<tr>
<td>13</td>
<td>DO CASE (17)</td>
<td>EXAPL13</td>
</tr>
<tr>
<td>14</td>
<td>DO WHILE (INFO.LT.20)</td>
<td>EXAPL14</td>
</tr>
<tr>
<td>15</td>
<td>DO UNTIL (LENGTH.LE.INFO)</td>
<td>EXAPL15</td>
</tr>
<tr>
<td>16</td>
<td>CASE (LENGTH+6) THEN</td>
<td>EXAPL16</td>
</tr>
<tr>
<td>17</td>
<td>CASE (LENGTH+30) THEN</td>
<td>EXAPL17</td>
</tr>
<tr>
<td>18</td>
<td>CASE (COMPLETE RESULTS)</td>
<td>EXAPL18</td>
</tr>
<tr>
<td>19</td>
<td>END IF</td>
<td>EXAPL19</td>
</tr>
<tr>
<td>20</td>
<td>CASE (INFO+1)</td>
<td>EXAPL20</td>
</tr>
<tr>
<td>21</td>
<td>END WHILE</td>
<td>EXAPL21</td>
</tr>
<tr>
<td>22</td>
<td>CASE ELSE</td>
<td>EXAPL22</td>
</tr>
<tr>
<td>23</td>
<td>DO WHILE (LENGTH.GT.6)</td>
<td>EXAPL23</td>
</tr>
<tr>
<td>24</td>
<td>DO UNTIL (COMPLETE LENGTH)</td>
<td>EXAPL24</td>
</tr>
<tr>
<td>25</td>
<td>END WHILE</td>
<td>EXAPL25</td>
</tr>
<tr>
<td>26</td>
<td>END CASE</td>
<td>EXAPL26</td>
</tr>
<tr>
<td>27</td>
<td>BLOCK (COMPLETE RESULTS)</td>
<td>EXAPL27</td>
</tr>
<tr>
<td>28</td>
<td>WRITE (6,1)(INFO,LENGTH)</td>
<td>EXAPL28</td>
</tr>
<tr>
<td>29</td>
<td>FORMAT (10X,15.20X,15)</td>
<td>EXAPL29</td>
</tr>
<tr>
<td>30</td>
<td>END BLOCK</td>
<td>EXAPL30</td>
</tr>
<tr>
<td>31</td>
<td>BLOCK (COMPLETE LENGTH)</td>
<td>EXAPL31</td>
</tr>
<tr>
<td>32</td>
<td>LENGTH = LENGTH -10</td>
<td>EXAPL32</td>
</tr>
<tr>
<td>33</td>
<td>END BLOCK</td>
<td>EXAPL33</td>
</tr>
<tr>
<td>34</td>
<td>RETURN</td>
<td>EXAPL34</td>
</tr>
<tr>
<td>35</td>
<td>END</td>
<td>EXAPL35</td>
</tr>
</tbody>
</table>

Figure 3.2. DMATRAN Indented Listing
3.3 FORTRAN OUTPUT

Figure 3.3 illustrates a portion of the FORTRAN translation produced by the DMATRAN precompiler. The input was the SUBROUTINE EXAMPL from Fig. 3.2. The information in columns 73 to 80 of the translated FORTRAN is useful for tracing translated FORTRAN statements back to the original DMATRAN source statements. If the original source statement was a FORTRAN statement, columns 73 and 74 will contain "FO"; if it was a DMATRAN statement, column 1 is a "C" and columns 73 and 74 contain "DM." In either case columns 77 through 80 contain the original DMATRAN source statement number (the leftmost column of numbers on the DMATRAN listing) and columns 75 and 76 contain the depth of nesting of the original statement.
SUBROUTINE EAPPL (INFO,LENGTH)

C ILLUSTRATION OF DMATAN SYNTAX

C IF (INFO.LE.10 .AND. LENGTH.GT.0) THEN
  IF (INFO.LE.10 .AND. LENGTH.GT.0) GO TO 19997
  GO TO 19998

19997 CONTINUE
  CALL CALLER (INFO)
  ELSE
  GO TO 19999

19998 CONTINUE
  LENGTH=50
  ENC IF

19999 CONTINUE
  CASE (INFO+6)
  IF (INFO.LE.10 .AND. LENGTH.GT.0) THEN
    GO TO 19997
  END IF

  GO TO 19995

19993 CONTINUE
  LENGTH=LENGTH-INFO

19992 CONTINUE
  CASE (17)
  GO TO 19995

19990 CONTINUE
  GO UNTIL (LENGTH.LE.INFO)
  GO TO 19997

19989 CONTINUE
  IF (LENGTH.LE.INFO) GO TO 19988

19987 CONTINUE
  CASE (12)
  GO WHILE (INFO.LT.20)

19986 CONTINUE
  IF (LENGTH.GE.30) THEN
    GO TO 19992

19982 CONTINUE
  CASE (1)
  END IF

19963 CONTINUE
  ENC UNTIL
  GO TO 19999

19958 CONTINUE
  INFO=INFO+1

19991 CONTINUE
  CASE ELSE
  DO WHILE (LENGTH.LE.0)

19979 CONTINUE
  END WORK

19978 CONTINUE

Figure 3.3. Translated FORTRAN
4 DMA TRAN CONSTRAINTS

4.1 SYNTAX

- A maximum of 20 cards per statement
- Statement labels between 10000 and 19999 should not be used because the DMA TRAN preprocessor adds statement labels, beginning with label 19999 counting backwards, to the FORTRAN source code (Fig. 3.3).
- Don't transfer to labeled DMA TRAN statements with FORTRAN GO TO's.
- Comments may not be interspersed within DMA TRAN statements
- All two-word DMA TRAN directives may be written as two separate words or merged into one; i.e., DO UNTIL or DOUNTIL.

4.2 DO UNTIL

When the DO UNTIL...END UNTIL construct is used for iteration, it is important to note that the statements contained within the construct will be executed once before the logical expression is evaluated.

4.3 CASE

The value of <integer-expression> in CASE statements must be positive.

4.4 BLOCK CONSTRUCT

- Each BLOCK...END BLOCK construct should occur after all INVOKE statements which refer to the block name, but may be before or after the RETURN statement.
- Blocks can only be entered through INVOKE statements. Sequential control transfers around BLOCK...END BLOCK constructs. Do not use a GO TO enter the middle of a BLOCK.. END BLOCK construct from outside the block.
- The maximum number of INVOKEs and BLOCKs depends on the lengths of the BLOCK names and number of invocations, see Sec. 2.5.
5 DISPLAY COMMANDS

The DMATR.AN precompiler supports a variety of commands for controlling the format of DMATRAN source listings. The capabilities supported include:

- Suppressing source listing
- Double-spacing around comments
- Keyword recognition to allow a more simple DMATRAN syntax
- Page ejection
- Extended comments

5.1 COMMAND FORM

The DMATRAN command statement has two basic forms, both are FORTRAN comment statements and begin with a C in column 1. The first command statement form is

```
C<command>
```

<command> must start in column 2 and may be any of the following:

- LIST
- NOLIST
- DSOK
- NODS
- KWOK
- NOKW
- EJECT

The second form of the DMATRAN command statement is

```
CKCOM <value>
```

<value> must be a single character in column 7; column 6 must be blank.
5.2 SOURCE LISTING

The DMATRAN source listing may be turned off and on with the two commands

\begin{itemize}
  \item \texttt{CNOLIST}
  \item \texttt{CLIST} (default)
\end{itemize}

This feature is useful when the DMATRAN source code is large and only parts of the code are being modified.

5.3 DOUBLE-SPACE AROUND COMMENTS

Optional double-spacing around comments is obtained by the command

\texttt{CDSOOK}

and is turned off with the command

\texttt{CNODS} (default)

5.4 KEYWORD RECOGNITION

The commands for control of display format are:

\begin{itemize}
  \item \texttt{CKWOK}
  \item \texttt{CNOKW} (default)
\end{itemize}

The \texttt{CKWOK} command (meaning "key-word OK") allows DMATRAN statements to be written in a simpler syntax. When the DMATRAN precompiler is in keyword recognition mode, it recognizes DMATRAN statements which do not have parentheses surrounding clauses (as well as all statements normally recognized). In this mode, DMATRAN keywords are recognized if they begin a statement, contain no blanks, and are immediately followed by a blank.

A problem may arise if FORTRAN is the embedded language. FORTRAN IF statements will be interpreted as DMATRAN IF statements when a blank precedes the left parenthesis of the statement. Figure 5.1 contains an example of the simpler DMATRAN syntax with English as the embedded language. Figure 5.2 is the listing that results from processing the example in Fig. 5.1 which uses the DMATRAN display commands.
SUBROUTINE TO FIND SQUARE ROOTS

FOR EACH INPUT VALUE DETERMINE THE SQUARE ROOT
INITIAL VALUE IS POSITIVE, REAL
DO UNTIL AN EOF IS ENCOUNTERED
INVOKED REAC INPUT VALUE
DETERMINE INITIAL ESTIMATE OF SQUARE ROOT
DO WHILE ESTIMATE HAS CONVERGED TO A
IF CURRENT ESTIMATE IS WORSE THAN PREVIOUS ESTIMATE THEN
DETERMINE NEW ESTIMATE
END IF
END WHILE

CURRENT EPSILON IS 1.0E-05
IF ESTIMATE**2 IS WITHIN EPSILON OF A THEN
INVOKED PRINT SQUARE ROOT
ELSE
INVOKED PRINT ERROR MESSAGE
END IF
END UNTIL

THOSE BLOCKS WILL BE CCED LATER

BLOCK REAC INPUT VALUE
END BLOCK

BLOCK PRINT ERROR MESSAGE
END BLOCK

BLOCK WRITTEN 6/77

BLOCK (PRINT SQUARE ROOT)
PRINT 1000, ROOT
1000 FORMAT (* ROOT = *E12.5)
END BLOCK
RETURN
END

Figure 5.1. DMATRAN Keyword Syntax
SEQ NEST SOURCE

1 C KWOK
2 SUBROUTINE TO FIND SQUARE ROOTS
3 COSOK FOR EACH INPUT VALUE DETERMINE THE SQUARE ROOT
4 C INITIAL VALUE IS POSITIVE, REAL
5
6 DOUNTIL AN EOF IS ENCOUNTERED
7 1 INVoke READ INPUT VALUE
8 1 DETERMINE INITIAL ESTIMATE OF SQUARE ROOT
9 1 DOWNWhile INITIAL ESTIMATE HAS CONVERGED TO A
10 2 . . IF CURRENT ESTIMATE IS WORSE THAN PREVIOUS ESTIMATE THEN
11 3 . . DETERMINE NEW ESTIMATE
12 2 . . END IF
13 1 . END WHILE
14 1 C CURRENT EPSILON IS 1.0E-05
15 1 . . IF ESTIMATE**2 IS WITHIN EPSILON OF A THEN
16 2 . . INVoke PRINT SQUARE Root
17 1 . ELSE
18 2 . . INVoke PRINT ERROR MESSAGE
19 1 . END IF
20 . END UNTIL
21 C

SEQ NEST SOURCE

28 C
29 CNODS
30 C KWOK
31 C BLOCK WRITTEN 6/77
32 C BLOCK (PRINT SQUARE Root)
33 1 PRINT 1000, ROOT
34 1 1000 FORMAT(* Root = *,E12.5)
35 END BLOCK
36 RETURN
37 END

Figure 5.2. Listing of DMATRAN Keyword Example
5.5 PAGE EJECTION

Specification of page ejection at any point in the DMATRAN listing is especially useful in delineating BLOCK structures; it is obtained by the command

CEJECT

5.6 EXTENDED COMMENT

Comments may follow a statement by using the command

CXCOM <value>

For example, in the command

CXCOM ;

the extended comment character is considered to be ";". The statement

T = T + DT ; ADVANCE TIME

would appear on the DMATRAN listing unmodified, but the text "ADVANCE TIME" would be changed into a FORTRAN comment on the translated FORTRAN output.

C ADVANCE TIME

T = T + DT
APPENDIX A

SUMMARY OF DMATRAN STATEMENTS AND COMMANDS.
### SUMMARY OF DMATRAN STATEMENTS AND COMMANDS

<table>
<thead>
<tr>
<th>PAGE</th>
<th>DMATRAN STATEMENT</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-3</td>
<td>IF...THEN...ELSE...END IF</td>
<td>Selection construct</td>
</tr>
<tr>
<td>2-5</td>
<td>DO WHILE...END WHILE</td>
<td>Iteration with test at top</td>
</tr>
<tr>
<td>2-5</td>
<td>DO UNTIL...END UNTIL</td>
<td>Iteration with test at bottom</td>
</tr>
<tr>
<td>2-8</td>
<td>CASE OF...CASE...CASE ELSE...END CASE</td>
<td>Selection construct</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PAGE</th>
<th>DMATRAN COMMAND</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>(defaults underlined)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5-2</td>
<td>LIST</td>
<td>Resume listing of DMATRAN source statements</td>
</tr>
<tr>
<td>5-2</td>
<td>NOLIST</td>
<td>Suppress listing of DMATRAN source statements</td>
</tr>
<tr>
<td>5-2</td>
<td>DSOK</td>
<td>Double-space around comments</td>
</tr>
<tr>
<td>5-2</td>
<td>NODS</td>
<td>Suppress double-spacing around comments</td>
</tr>
<tr>
<td>5-2</td>
<td>KWOK</td>
<td>Enter keyword recognition mode</td>
</tr>
<tr>
<td>5-2</td>
<td>NOKW</td>
<td>Leave keyword recognition mode</td>
</tr>
<tr>
<td>5-5</td>
<td>EJECT</td>
<td>New page</td>
</tr>
<tr>
<td>5-5</td>
<td>XCOM &lt;char&gt;</td>
<td>Set the extended comment character</td>
</tr>
</tbody>
</table>
**FILES USED AT RADC INSTALLATION**

<table>
<thead>
<tr>
<th>UNIT</th>
<th>FILE NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INPUT</td>
<td>DMATRAN source input</td>
</tr>
<tr>
<td>2</td>
<td>PRINT</td>
<td>Indented listing of DMATRAN source</td>
</tr>
<tr>
<td>3</td>
<td>COMPILE</td>
<td>Compilable FORTRAN output</td>
</tr>
</tbody>
</table>

---

**FILES USED AT DMA INSTALLATIONS**

<table>
<thead>
<tr>
<th>UNIT</th>
<th>FILE NAME</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>INPUT</td>
<td>DMATRAN source input</td>
</tr>
<tr>
<td>6</td>
<td>PRINT</td>
<td>Indented listing of DMATRAN source</td>
</tr>
<tr>
<td>3</td>
<td>COMPILE</td>
<td>Compilable FORTRAN output</td>
</tr>
</tbody>
</table>
APPENDIX C

JOB STREAMS

<table>
<thead>
<tr>
<th>Command</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEQ</td>
<td>Start sequence</td>
</tr>
<tr>
<td>STOR</td>
<td>Store the result</td>
</tr>
<tr>
<td>END</td>
<td>End the sequence</td>
</tr>
</tbody>
</table>

The top stream is the flowchart template used for the processes.
The job stream in the following example can be used to execute the DMATRAN precompiler.

1. $ IDENT
2. $ SELECT BFCBGRS4/DMATRAN/EXECUTE
3. $ PRMFL 01,R,S,(BCD dmatran source file)
4. $ PRMFL 03,W,S, (BCD translated FORTRAN source file)
5. $ ENDJOB

The BCD DMATRAN Source File may have been generated by a programmer or by the FAVS Restructure Option (See FAVS User's Guide, General Research Corporation CR-1-754).
In order to use the DMATRAN precompiler, using source code written in DMATRAN generated by a programmer or by FAVS restructurer, the job stream shown in the following example can be used.

1. $ snumb (number)
2. $ ident
3. $ program rlhs
4. $ limits (CP time limit),32k,,(print line limit)
5. $ prmfl h*,r,r,>udd>3201c0320>Urban>dmatran>hstar
6. $ select >udd>3201c0320>Urban>dmatran>filedefs -ascii
7. $ prmfl 01,r,s,>udd>(BCD dmatran source file)
8. $ prmfl 03,w,s,>udd>(BCD Translated FORTRAN source file)
9. $ endjob
DMS UNIVAC 1100/42
SAMPLE DMATRAN JOB STREAM

The job stream in the following example can be used to execute the DMATRAN precompiler.

@ASG, A YOURSOURCE.
@USE Y., YOURSOURCE.
@ASG, A DBM*FAVS-DMA.
@USE DMA., DBM*FAVS-DMA.
@QT DMA.TRAN
@ADD Y.ELEMENTS

The UNIVAC 1100/42 installation of the DMATRAN precompiler supports an additional command (see Sec. 5.1) to assist in compiling translated DMATRAN. This command contains CFOR in columns 1 thru 4, followed by any desired information in columns 5 thru 80. The DMATRAN precompiler changes the C in column 1 of all CFOR commands to an @ character as the CFOR command is written to the FORTRAN output file. When the DMATRAN precompiler automatically adds the FORTRAN output file to the runstream, the translated CFOR cards direct the FORTRAN V compiler. Note that to compile a DMATRAN source element, the first line in the element should be a CFOR command. Indented listings without FORTRAN V compilations may be obtained by omitting CFOR commands.
## INDEX

<table>
<thead>
<tr>
<th>Topic</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic Indentation</td>
<td>3-2</td>
</tr>
<tr>
<td>BLOCK Construct</td>
<td>2-11, 4-1</td>
</tr>
<tr>
<td>BLOCK, Examples</td>
<td>2-12, 2-15</td>
</tr>
<tr>
<td>BLOCKs, Maximum Number of BLOCK Name</td>
<td>2-13</td>
</tr>
<tr>
<td>Card, Maximum Number per Statement</td>
<td>4-1</td>
</tr>
<tr>
<td>CASE Construct</td>
<td>2-8, 4-1</td>
</tr>
<tr>
<td>CASE, Examples of</td>
<td>2-9, 2-10</td>
</tr>
<tr>
<td>COMMENTS</td>
<td>5-1, 5-2</td>
</tr>
<tr>
<td>Depth, Nesting</td>
<td>3-2, 3-3</td>
</tr>
<tr>
<td>Display Commands</td>
<td>5-1</td>
</tr>
<tr>
<td>DMA Files</td>
<td>B-2</td>
</tr>
<tr>
<td>DMA Job Streams</td>
<td>C-4</td>
</tr>
<tr>
<td>DMATRAN Commands</td>
<td>5-1</td>
</tr>
<tr>
<td>DMATRAN Constructs</td>
<td>2-1</td>
</tr>
<tr>
<td>DMATRAN Guidelines</td>
<td>4-1</td>
</tr>
<tr>
<td>DMATRAN Input</td>
<td>3-1</td>
</tr>
<tr>
<td>DMATRAN Output</td>
<td>3-2, 3-3</td>
</tr>
<tr>
<td>DO UNTIL Construct</td>
<td>2-5, 4-1</td>
</tr>
<tr>
<td>DO UNTIL, Examples of</td>
<td>2-7</td>
</tr>
<tr>
<td>DO WHILE Construct</td>
<td>2-5</td>
</tr>
<tr>
<td>DO WHILE, Examples of</td>
<td>2-6</td>
</tr>
<tr>
<td>DSOK Command</td>
<td>5-2, 5-3, 5-4</td>
</tr>
<tr>
<td>ELSE</td>
<td>2-3</td>
</tr>
<tr>
<td>EJECT Command</td>
<td>5-5</td>
</tr>
<tr>
<td>Files</td>
<td></td>
</tr>
<tr>
<td>DMA</td>
<td>B-2</td>
</tr>
<tr>
<td>RADC</td>
<td>B-2</td>
</tr>
<tr>
<td>IF...THEN construct</td>
<td>2-3</td>
</tr>
<tr>
<td>IF( )THEN, Examples of</td>
<td>2-3, 2-4</td>
</tr>
<tr>
<td>INVOKE, Examples of</td>
<td>2-12, 2-15</td>
</tr>
<tr>
<td>INVOKE, Number of</td>
<td>2-13</td>
</tr>
<tr>
<td>INVOKE Statement</td>
<td>2-11, 4-1</td>
</tr>
<tr>
<td>JOB Streams</td>
<td></td>
</tr>
<tr>
<td>DMA UNIVAC 1100/42</td>
<td>C-4</td>
</tr>
<tr>
<td>RADC HONEYWELL 6180/GCOS</td>
<td>C-2</td>
</tr>
<tr>
<td>RADC HONEYWELL 6180/MULTICS</td>
<td>C-3</td>
</tr>
<tr>
<td>INDEX CONT.</td>
<td>PAGES</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>KWOK Command</td>
<td>5-2, 5-3, 5-4</td>
</tr>
<tr>
<td>Label, Initial Generation of Level, Nesting</td>
<td>4-1</td>
</tr>
<tr>
<td>LIST Command</td>
<td>3-2, 3-3</td>
</tr>
<tr>
<td>Listing, FORTRAN</td>
<td>5-2, 5-3, 5-4</td>
</tr>
<tr>
<td>Listing, DMATRAN</td>
<td>3-3</td>
</tr>
<tr>
<td>Listing Unit, DMATRAN</td>
<td>3-2</td>
</tr>
<tr>
<td>Nesting Depth</td>
<td>B-2</td>
</tr>
<tr>
<td>NODS Command</td>
<td>3-2, 3-3</td>
</tr>
<tr>
<td>NOKW Command</td>
<td>5-2, 5-3, 5-4</td>
</tr>
<tr>
<td>Nolist Command</td>
<td>5-2, 5-3, 5-4</td>
</tr>
<tr>
<td>Number of BLOCKs</td>
<td>2-13</td>
</tr>
<tr>
<td>Number of INVOKEs</td>
<td>2-13</td>
</tr>
<tr>
<td>RADC Files</td>
<td>B-2</td>
</tr>
<tr>
<td>RADC Job Streams</td>
<td>C-2, C-3</td>
</tr>
<tr>
<td>Structured Programming</td>
<td>1-1</td>
</tr>
<tr>
<td>XCOM Command</td>
<td>5-5</td>
</tr>
</tbody>
</table>
DMATRAN CONTROL CONSTRUCTS

CONDITIONAL
IF (logical expression) THEN
CASE OF (integer expression)
{CASE (index {, index})}
END IF
END CASE

LOOP
DO WHILE (logical expression)
END WHILE
END LOOP
DO UNTIL (logical expression)
END UNTIL

SEQUENTIAL
INVOKE (block name)
END INVOKE
BLOCK (block name)
END BLOCK

[ ] = optional
{ } = optional an arbitrary number of times

DMATRAN COMMANDS (Defaults Underlined)

LIST
NOLIST
DSOK
NDOS
KWOK
NOKW
EJECT
XCOM <char>

List source statements
Suppress listing of source statements
Double-space around comments
Single-space around comments
Reserved keywords identify DMATRAN statements
No reserved keywords
New page
Set the extended comment character