FORTRAN Reference Manual

Revision 3

J. C. Adams
B. A. Domenico
P. A. Rotar
FOREWORD

This manual describes the FORTRAN language as interpreted by the NCAR compiler. Portions of this manual have been edited from Control Data copyrighted publications with the permission of the Control Data Corporation. New sections have been added and all of the examples have been checked using computer runs. Sample programs have been run on the NCAR system.

NCAR FORTRAN uses the American National Standards Institute standard FORTRAN X3.9-1966 as a guide in order to have the current standard a subset of the NCAR FORTRAN as far as possible. However, there are a number of cases where its usage is nonstandard. Some of these have been noted. Chapter 10 contains a statement list for FORTRAN X3.9-1966, the new standard (FORTRAN 77) currently in the final release process, NCAR Control Data 7600 FORTRAN, and CRAY FORTRAN.

Astrik Deirmendjian was the programmer responsible for testing the examples. David Kitts and Sandra Fuller, who are responsible for compilers at NCAR, were very helpful in interpretation of technical material. John Snyder was the final reader for this issue. Linda Besen and Sara Ladd did the layout work and prepared the final copy. Linda generated the index based on the execution of a KWIC index program using topical subtitles. Veronica Martinez assisted the editors.

This document is the third revision of Technical Note TN/IA-70 and supersedes the FORTRAN Reference Manual dated October 1975.

February 1978
# TABLE OF CONTENTS

## CHAPTER 1

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORTRAN CODING FORMS</td>
<td>1.1</td>
</tr>
<tr>
<td>Introduction</td>
<td>1.1</td>
</tr>
<tr>
<td>Coding Line</td>
<td>1.2</td>
</tr>
<tr>
<td>Statement</td>
<td>1.2</td>
</tr>
<tr>
<td>Statement Label</td>
<td>1.3</td>
</tr>
<tr>
<td>Identification Field</td>
<td>1.3</td>
</tr>
<tr>
<td>Comment Information</td>
<td>1.3</td>
</tr>
<tr>
<td>Special Comment Option</td>
<td>1.4</td>
</tr>
<tr>
<td>Punched Cards</td>
<td>1.4</td>
</tr>
<tr>
<td>Blank Cards</td>
<td>1.4</td>
</tr>
</tbody>
</table>

## CHAPTER 2

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORTRAN CONCEPTS</td>
<td>2.1</td>
</tr>
<tr>
<td>FORTRAN Character Set</td>
<td>2.1</td>
</tr>
<tr>
<td>Identifiers</td>
<td>2.2</td>
</tr>
<tr>
<td>Alphanumeric Identifier</td>
<td>2.2</td>
</tr>
<tr>
<td>Constants</td>
<td>2.4</td>
</tr>
<tr>
<td>Integer Constants</td>
<td>2.4</td>
</tr>
<tr>
<td>Octal Constants</td>
<td>2.6</td>
</tr>
<tr>
<td>Real Constants</td>
<td>2.7</td>
</tr>
<tr>
<td>Double-Precision Constants</td>
<td>2.8</td>
</tr>
<tr>
<td>Complex Constants</td>
<td>2.10</td>
</tr>
<tr>
<td>Hollerith Constants</td>
<td>2.11</td>
</tr>
<tr>
<td>Logical Constants</td>
<td>2.12</td>
</tr>
<tr>
<td>Variables</td>
<td>2.13</td>
</tr>
<tr>
<td>Types of Variables</td>
<td>2.14</td>
</tr>
<tr>
<td>Integer Variables</td>
<td>2.14</td>
</tr>
<tr>
<td>Real Variables</td>
<td>2.15</td>
</tr>
<tr>
<td>Double-Precision Variables</td>
<td>2.15</td>
</tr>
<tr>
<td>Complex Variables</td>
<td>2.15</td>
</tr>
<tr>
<td>Naming Octal Constants</td>
<td>2.15</td>
</tr>
<tr>
<td>Alphanumeric Names (Hollerith)</td>
<td>2.16</td>
</tr>
<tr>
<td>Logical Variables</td>
<td>2.16</td>
</tr>
<tr>
<td>Classes of Variables</td>
<td>2.16</td>
</tr>
<tr>
<td>Simple Variable</td>
<td>2.16</td>
</tr>
<tr>
<td>Subscripted Variable</td>
<td>2.17</td>
</tr>
<tr>
<td>Arrays</td>
<td>2.19</td>
</tr>
<tr>
<td>Array Structure</td>
<td>2.20</td>
</tr>
</tbody>
</table>
CHAPTER 3

EXPRESSIONS: ARITHMETIC, LOGICAL, MASKING, MIXED MODE . 3.1
Introduction .................................. 3.1
Expressions .................................... 3.2
  Arithmetic Expressions .......................... 3.2
    Standard Evaluation ............................ 3.3
    Evaluation with Optimization .................... 3.6
  Logical Expressions ................................ 3.11
    Relational Operators ............................ 3.11
    Logical Connectives .............................. 3.13
  Masking Expressions .................................. 3.14
  Mixed-Mode Expressions ............................ 3.17

CHAPTER 4

REPLACEMENT STATEMENTS .................................. 4.1
Introduction ....................................... 4.1
Mixed-Mode Replacement Statements ..................... 4.1
Multiple Replacement Statements ....................... 4.4

CHAPTER 5

TYPE DECLARATIONS AND STORAGE ALLOCATION .................. 5.1
Introduction ....................................... 5.1
Statement Ordering .................................. 5.1
Type Declaration .................................... 5.2
Implicit Typing ..................................... 5.4
  IMPLICIT Statement ................................ 5.4
Symbolic Name of a Constant ................................ 5.7
  PARAMETER Statement ................................ 5.7
Dimension Declaration .................................. 5.10
  Variable Dimensions .................................. 5.13
Common Declaration .................................... 5.14
  Local Variables ..................................... 5.14
  Common Variables .................................... 5.15
Equivalence Declaration .................................. 5.24
  Storage Allocation .................................. 5.28
Data Declaration ..................................... 5.35
  DATA Statement .................................... 5.35

CHAPTER 6

CONTROL STATEMENTS .................................. 6.1
Introduction ....................................... 6.1
GO TO Statements ..................................... 6.2
  Unconditional GO TO Statement ..................... 6.2
  Assigned GO TO Statement .......................... 6.2
  ASSIGN Statement .................................. 6.3
  Computed GO TO Statement ........................... 6.3
IF Statements ........................................ 6.5
  Three-Branch Arithmetic IF Statement ............. 6.5
  Two-Branch Relational IF Statement ............... 6.7
  One-Branch Relational IF Statement ............... 6.8
DO Statement ....................................... 6.9
  DO Loop Execution .................................. 6.13
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DO Nests</td>
<td>6.15</td>
</tr>
<tr>
<td>Subscripting within DO Loops</td>
<td>6.18</td>
</tr>
<tr>
<td>Extended Range</td>
<td>6.18</td>
</tr>
<tr>
<td>Optimization</td>
<td>6.21</td>
</tr>
<tr>
<td>*FORTRAN,1</td>
<td>6.21</td>
</tr>
<tr>
<td>*FORTRAN,2</td>
<td>6.22</td>
</tr>
<tr>
<td>*FORTRAN,3</td>
<td>6.23</td>
</tr>
<tr>
<td>*FORTRAN,4</td>
<td>6.23</td>
</tr>
<tr>
<td>*FORTRAN,13</td>
<td>6.24</td>
</tr>
<tr>
<td>*FORTRAN,18</td>
<td>6.24</td>
</tr>
<tr>
<td>*FORTRAN,25</td>
<td>6.24</td>
</tr>
<tr>
<td>*FORTRAN,60</td>
<td>6.24</td>
</tr>
<tr>
<td>CONTINUE Statement</td>
<td>6.26</td>
</tr>
<tr>
<td>PAUSE Statement</td>
<td>6.26</td>
</tr>
<tr>
<td>END Statement</td>
<td>6.26</td>
</tr>
<tr>
<td>Normal Termination Statements</td>
<td>6.27</td>
</tr>
<tr>
<td>IF/THEN/ELSE</td>
<td>6.28</td>
</tr>
<tr>
<td>General Structure</td>
<td>6.28</td>
</tr>
<tr>
<td>Nesting Levels</td>
<td>6.29</td>
</tr>
<tr>
<td>Transfer of Control</td>
<td>6.29</td>
</tr>
<tr>
<td>IF THEN Statement</td>
<td>6.30</td>
</tr>
<tr>
<td>Form</td>
<td>6.30</td>
</tr>
<tr>
<td>Block</td>
<td>6.30</td>
</tr>
<tr>
<td>Execution</td>
<td>6.30</td>
</tr>
<tr>
<td>ELSE IF Statement</td>
<td>6.31</td>
</tr>
<tr>
<td>Form</td>
<td>6.31</td>
</tr>
<tr>
<td>Block</td>
<td>6.31</td>
</tr>
<tr>
<td>Execution</td>
<td>6.31</td>
</tr>
<tr>
<td>ELSE Statement</td>
<td>6.32</td>
</tr>
<tr>
<td>Form</td>
<td>6.32</td>
</tr>
<tr>
<td>Block</td>
<td>6.32</td>
</tr>
<tr>
<td>Execution</td>
<td>6.32</td>
</tr>
<tr>
<td>END IF Statement</td>
<td>6.33</td>
</tr>
<tr>
<td>Form</td>
<td>6.33</td>
</tr>
<tr>
<td>Execution</td>
<td>6.33</td>
</tr>
<tr>
<td>DO-Loop (FORTRAN 77)</td>
<td>6.37</td>
</tr>
<tr>
<td>DO-Loop Structure</td>
<td>6.37</td>
</tr>
<tr>
<td>DO Statement</td>
<td>6.37</td>
</tr>
<tr>
<td>DO-Loop Range</td>
<td>6.37</td>
</tr>
<tr>
<td>Terminal Statement</td>
<td>6.38</td>
</tr>
<tr>
<td>DO-Loop Execution</td>
<td>6.39</td>
</tr>
<tr>
<td>DO Statement Execution</td>
<td>6.40</td>
</tr>
<tr>
<td>Loop Control Processing</td>
<td>6.41</td>
</tr>
<tr>
<td>Execution of the Range</td>
<td>6.41</td>
</tr>
<tr>
<td>Incrementation Processing</td>
<td>6.41</td>
</tr>
<tr>
<td>Active and Inactive DO-Loops</td>
<td>6.42</td>
</tr>
</tbody>
</table>
### CHAPTER 7

**PROGRAM, FUNCTION, AND SUBROUTINE** ........................................... 7.1

- **Introduction** .................................................. 7.1
- **Main Program** .................................................. 7.2
- **Subprograms** ..................................................
  - **Formal Parameters** ........................................... 7.3
  - **Actual Parameters** .......................................... 7.4
- **Subroutine Subprogram** ......................................... 7.7
- **CALL Statement** ............................................... 7.8
- **Function** ........................................................
  - **Function Definition** ......................................... 7.11
  - **Function Reference** ......................................... 7.13
- **Arithmetic Statement Function** ................................ 7.15
- **Library Functions** ............................................. 7.18
- **In-Line Algorithms** ........................................... 7.18
- **Mathematical Library Routines** ................................ 7.18
- **Auxiliary Library Routines** ................................... 7.25
- **Block Data** ..................................................... 7.26
- **RETURN and END Statements** .................................. 7.27
- **ENTRY Statement** ................................................ 7.28
- **EXTERNAL Statement** ........................................... 7.31
- **Variable Dimensions in Subprograms** .......................... 7.33
- **Arrangement of FORTRAN Decks** ................................ 7.35

### CHAPTER 8

**INPUT/OUTPUT FORMATS** ......................................................... 8.1

- **Introduction** .................................................... 8.1
- **Input/Output List** ............................................... 8.2
  - **Standard Subscripts** ......................................... 8.2
  - **Nonstandard Subscripts** ..................................... 8.3
  - **Array Transmission** .......................................... 8.3
- **FORMAT Declaration** ............................................. 8.7
- **FORMAT Specifications** .......................................... 8.8
- **Conversion Specifications** ....................................... 8.9

  - **Ew.d Input** ................................................... 8.12
  - **Fw.d Input** ................................................... 8.14
  - **Gw.d Input** ................................................... 8.16
  - **Dw.d Input** ................................................... 8.18
  - **Iw Input** ...................................................... 8.19
  - **Ow Input** ...................................................... 8.20
  - **Aw Input** ...................................................... 8.22
  - **Rw Input** ...................................................... 8.23
  - **Aw Output** ...................................................... 8.24
  - **Ow Output** ...................................................... 8.25
  - **Iw Output** ...................................................... 8.25
  - **Lw Input** ...................................................... 8.25
  - **Lw Output** ...................................................... 8.25
CHAPTER 9

INPUT/OUTPUT STATEMENTS .............................................. 9.1
Data Transfer ......................................................... 9.1
  Units .................................................................. 9.2
  Source Data ....................................................... 9.2
  The I/O List ....................................................... 9.3
  Rules ................................................................. 9.3
Input/Output Terminology .............................................. 9.4
  Synchronous and Asynchronous ................................. 9.4
    Core-to-Core Transfer ......................................... 9.4
  Access to Data .................................................. 9.5
  Keywords and Keylists ......................................... 9.5
  Formatted and Unformatted .................................. 9.6
Reading and Writing .................................................. 9.7
  Generalized Form of Synchronous Read/Write ................. 9.8
    Format ......................................................... 9.8
    Input Parameters ............................................. 9.9
    Output Parameters .......................................... 9.11
    Syntax Error Diagnostics .................................. 9.11
Specific Forms of Synchronous I/O ................................. 9.12
    Unformatted Input Statements .............................. 9.15
    Unformatted Output Statements ............................ 9.17
    Structure of Logical Records .............................. 9.18
Generalized Form of Asynchronous READ/WRITE ................. 9.19
  Format ......................................................... 9.19
  Input Parameters ............................................. 9.20
  Output Parameters .......................................... 9.21
  Syntax Error Diagnostics .................................. 9.22
Alternate Form for Asynchronous Statements ..................... 9.23
  Parameters ..................................................... 9.24
    Record Structure for BUFFER OUT in Binary or ......... 9.25
      Character Code ............................................ 9.25
    Status Statement .......................................... 9.25
Internal Files ....................................................... 9.27
ENCODE/DECODE Statements ........................................ 9.27
CHAPTER 9
Dataset Support Statements .......................... 9.38
The OPEN Statement .................................... 9.38
Form .................................................. 9.38
Ident Parameters ....................................... 9.39
Error Condition Statement Label .................... 9.40
Keylist Parameters ..................................... 9.40
Return Parameter ....................................... 9.42
Syntax Error Diagnostics ............................... 9.43
The CLOSE Statement .................................... 9.44
Form .................................................. 9.44
Input Parameters ....................................... 9.44
Error Condition Statement Label .................... 9.45
Return Parameter ....................................... 9.45
Syntax Error Diagnostics ............................... 9.45
The INQUIRE Statement ................................ 9.46
Form .................................................. 9.46
Ident Parameters ....................................... 9.46
Error Condition Statement Label .................... 9.47
Keylist Parameters ..................................... 9.48
Syntax Error Diagnostics ............................... 9.54
The SETPASS Statement ................................. 9.55
Form .................................................. 9.55
Parameters ............................................ 9.55
Syntax Error Diagnostics ............................... 9.57
File Positioning with Sequential Access .............. 9.58
Positioning Statements .................................. 9.60
Example Using Sequential Access ....................... 9.65
Comments ............................................. 9.66
Example Using Direct Access ........................... 9.67
Comments ............................................. 9.67
Example Using Direct and Sequential Access on the
Same Unit ............................................. 9.68
Comments ............................................. 9.68
Example of User Catalog Program ....................... 9.69
Comments ............................................. 9.69
Summary Table of FORTRAN Syntax ..................... 9.73

CHAPTER 10
FORTRAN STATEMENT LIST ............................ 10.1
Introduction ........................................... 10.1
Program and Subprogram Statements .................... 10.2
Subprogram Reference and Transfer ...................... 10.2
Arithmetic Statement Function ........................ 10.2
Type Declaration ...................................... 10.2
Storage Allocation .................................... 10.4
Replacement .......................................... 10.4
Branching and IF Logic ................................ 10.4
Looping and Control Statements ......................... 10.6
FORMAT ............................................. 10.6
Data Input ........................................... 10.6
FORTRAN CODING FORMS
FORTRAN CODING FORMS

INTRODUCTION

FORTRAN is a second-level programming language commonly used with computers for solving scientific problems. It consists of statements which may be associated easily with problems set down in mathematical notation. Programs written in a second-level language usually require a systems program called a compiler, which scans the statements presented and reformulates these statements in machine terms. The FORTRAN language compiler described in this manual is for the Control Data 7600 computer at the National Center for Atmospheric Research (NCAR).

A source program written in FORTRAN is an ordered set of statements in the FORTRAN language, from which machine instructions, as well as storage areas, are defined.

After the compiler finishes the reformulation of a FORTRAN program, the resulting machine program, or object program, is available to the computer for execution.
CODING LINE

A coding line contains 80 columns, in which characters are written one character per column. A FORTRAN line is a string of 72 characters from the NCAR FORTRAN character set.

There are four types of coding lines:

<table>
<thead>
<tr>
<th>Type</th>
<th>Columns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statement</td>
<td>1-5</td>
<td>Statement label (optional)</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>Blank or zero</td>
</tr>
<tr>
<td></td>
<td>7-72</td>
<td>FORTRAN statement</td>
</tr>
<tr>
<td></td>
<td>73-80</td>
<td>Identification field (optional)</td>
</tr>
<tr>
<td>Continuation</td>
<td>1-5</td>
<td>Blank</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>FORTRAN character other than blank or zero</td>
</tr>
<tr>
<td></td>
<td>7-72</td>
<td>Continued FORTRAN statement</td>
</tr>
<tr>
<td></td>
<td>73-80</td>
<td>Identification field (optional)</td>
</tr>
<tr>
<td>Comment</td>
<td>1</td>
<td>Character C</td>
</tr>
<tr>
<td></td>
<td>2-80</td>
<td>Comments</td>
</tr>
<tr>
<td>Data</td>
<td>1-80</td>
<td>Data</td>
</tr>
</tbody>
</table>

STATEMENT

The FORTRAN statement is written in columns 7-72. Statements longer than 66 columns may be carried to the next line by using a continuation line. Blanks may be used freely in FORTRAN statements to provide readability; they are ignored by the FORTRAN compiler. The character $ may be used to write more than one statement on a coding line, but cannot be used as a separator with FORMAT or DATA statements. The $ indicates that the next column will be interpreted as column 7.
The first line of every statement is called an initial line and must have a blank or zero in column 6. If statements occupy more than one line, all subsequent lines are continuation lines and must have a number or letter other than blank or zero in column 6. A continuation line is not a comment.

**STATEMENT LABEL**

A statement label is a string of one to five digits occupying any column position 1 through 5 of the initial coding line of the statement. This label is used to identify a particular FORTRAN statement. Any statement may have a statement label, but only statements referred to elsewhere in the program require identifying statement labels. The same label may be given to only one statement within a program unit.

**IDENTIFICATION FIELD**

Columns 73 through 80 may be used for identification when the FORTRAN program is to be punched on cards. The identification columns are always ignored by the compiler. Usually these columns contain sequencing information provided by the programmer. They may be left blank.

**COMMENT INFORMATION**

Comment information is designated by a C in column 1 of a coding line. Comment information appears in the source program and the source program listing, but it is not translated by the compiler. Comment lines may contain any character in the set for the Control Data 7600 computer. Each line must have the character C in column 1. The continuation character in column 6 does not apply to comment cards. Comments do not affect the program in any way.
### SPECIAL COMMENT

A special comment card has a C in column 1 and a period in column 2. Only C. comment cards will be listed when the FORTRAN list output default option is selected, (see page 12.22 in Chapter 12). This option is included to convey special information to NCAR library routine users, even when FORTRAN cards are not listed.

### PUNCHED CARDS

Each line of the coding form corresponds to one 80-column card in the program deck. The terms *line* and *card* are often used interchangeably. Source programs and data can be read from cards into the computer; some output from the computer--relocatable binary programs or data--also can be punched directly onto cards.

### BLANK CARDS

If a blank card appears between two statements, it is ignored. When cards are being used for data input, all 80 columns may be used.
```
<table>
<thead>
<tr>
<th>Statement</th>
<th>FORTRAN STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PROGRAM SAMPLE</td>
</tr>
<tr>
<td>C. SAMPLE</td>
<td>PROGRAM TO SHOW CODING LINE FORMATS</td>
</tr>
<tr>
<td></td>
<td>DIMENSION, MAT(25, 25),</td>
</tr>
<tr>
<td></td>
<td>REAL MAT</td>
</tr>
<tr>
<td></td>
<td>PRINT 103</td>
</tr>
<tr>
<td>1.05</td>
<td>FORMAT(*1, THIS LINE SHOWS THAT CONTINUATION OF A FORTRAN SENTENCE)</td>
</tr>
<tr>
<td>5</td>
<td>READ((S, I, 0), N, (MAT(I, J), I = 1, N, J = 1, N))</td>
</tr>
<tr>
<td>1.00</td>
<td>FORMAT((L, I), I = (8, E10.0)),</td>
</tr>
<tr>
<td></td>
<td>L = 1, 1, 2, DET = 1.0,</td>
</tr>
<tr>
<td></td>
<td>D = 6, E = 2, E,</td>
</tr>
<tr>
<td></td>
<td>D = 10, L = K, N,</td>
</tr>
<tr>
<td></td>
<td>R = MAT(I, J) / MAT(J, J),</td>
</tr>
<tr>
<td></td>
<td>D = 10, J = K, N,</td>
</tr>
<tr>
<td>1.0</td>
<td>MAT(I, J) = MAT(I, J) - MAT(I, L) * MAT(L, J),</td>
</tr>
<tr>
<td></td>
<td>L = K,</td>
</tr>
<tr>
<td></td>
<td>CONTINUE,</td>
</tr>
<tr>
<td>1.0</td>
<td>WRITE((S, I, 0), (MAT(K, J), K = 1, N))</td>
</tr>
<tr>
<td>1.02</td>
<td>FORMAT((L, I), I = (12, E10.2)),</td>
</tr>
<tr>
<td></td>
<td>L = DET, DET, MAT(I, J),</td>
</tr>
<tr>
<td></td>
<td>WRITE((S, I, 0), DET),</td>
</tr>
<tr>
<td>1.01</td>
<td>FORMAT((I, zero DETERMINANT EQUALS E15.4),</td>
</tr>
<tr>
<td></td>
<td>GO TO 5</td>
</tr>
<tr>
<td>END</td>
<td></td>
</tr>
</tbody>
</table>
```
II

FORTRAN CONCEPTS

IDENTIFIERS
CONSTANTS
VARIABLES
The following characters may be used in writing a FORTRAN program at NCAR.

- Alphabetic
  
  A, B, C, ..., X, Y, Z

- Numeric
  
  0, 1, 2, ..., 7, 8, 9

- Special

  blank  (space)
  Equals   =
  Plus     +
  Minus    -
  Asterisk *
  Slash    /
  Left parenthesis (
  Right parenthesis )
  Comma    ,
  Decimal point .
  Dollar sign $
  Apostrophe '
  Colon    :

Other special characters are available but are not supported by the FORTRAN standard. The character blank is ignored by the compiler except in Hollerith fields and otherwise may be used freely to improve program readability.
IDENTIFIERS

There are two kinds of identifiers: alphanumeric and statement identifiers.

ALPHANUMERIC IDENTIFIER

Alphanumeric identifiers are symbolic names used in a FORTRAN statement to identify a member of a class. An identifier is a string of one to six alphanumeric characters. The first character of an alphanumeric identifier must be a letter. Embedded blanks within an identifier are ignored. For example, the name ATEST is the same as AT EST. Identifiers are names used in the following classes of names:

<table>
<thead>
<tr>
<th>Type</th>
<th>FORTRAN Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array element</td>
<td>A(I,J) = 1.0</td>
</tr>
<tr>
<td>Variables</td>
<td>A63 = XYZZZZ + S</td>
</tr>
<tr>
<td>Subroutines</td>
<td>SUBROUTINE RUNGE</td>
</tr>
<tr>
<td>Entry</td>
<td>ENTRY TEST</td>
</tr>
<tr>
<td>Main programs</td>
<td>PROGRAM MAIN6</td>
</tr>
<tr>
<td>Function names</td>
<td>FUNCTION POLY</td>
</tr>
<tr>
<td>Labeled common blocks</td>
<td>COMMON/BL14/A,B,C</td>
</tr>
<tr>
<td>Block data subprograms</td>
<td>BLOCK DATA XTEST</td>
</tr>
</tbody>
</table>
Statement labels are unsigned integer constants used to identify a particular statement in the program. A statement label is a string of one to five digits placed in columns 1 through 5 of the initial line of a statement. The range is from 1 to 99999. Leading zeros are ignored, and zero is not a legal statement identifier. Within any given program or subprogram each statement identifier must be unique.

Example of Statement Labels

<table>
<thead>
<tr>
<th>Statement Label</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, 7, , ,</td>
<td>ET = LAMDA</td>
</tr>
<tr>
<td>4, 3</td>
<td>A = COS(X) + ET</td>
</tr>
<tr>
<td>1, 2, 5, 9</td>
<td>B = 4.3, 9 1</td>
</tr>
</tbody>
</table>
Seven types of constants are used in FORTRAN:

- Integer
- Octal
- Real
- Double precision
- Complex
- Hollerith
- Logical

Complex and double-precision constants are formed from real constants. The type of a constant is determined by its form and each type has a different internal representation.

An integer constant is a string of up to 18 decimal digits in the range \(-(2^{59}-1) \prec N \prec (2^{59}-1)\). \(2^{59}-1\) equals 57646075230342348710 (377777777777777777778). An integer constant is an exact representation of a positive, negative, or zero integral value. Integer add and subtract may be performed on all integers in the above range; however, no trap for overflow ever occurs on integer add or subtract. In a replacement statement for a constant greater than this range, the diagnostic CONSTANT OUT OF RANGE will occur.

Due to hardware limitations, integers in the range \(2^{59}-1 \prec |I| \prec 2^{48}-1\) may not be used to do integer multiply and divide. Since there is no integer divide in the hardware, it is necessary in these cases for the compiler to convert the number to real so the division can be done in floating point unit; the result of the floating point division is then truncated to an integer.
There is a hardware integer multiply used by compiled programs. The integer result will be no larger than $2^{48} - 1$ in absolute value. Overflow and indefinite conditions will occur when an integer multiply exceeds $2^{48} - 1$ in absolute value.

$2^{48} - 1$ equals $281474976710655_{10} (000077777777777777778)_{2}$

For numbers greater than $2^{48} - 1$ in absolute value, the print routine will output an R (meaning range error) for an I format. The number stored in the computer, however, is good for integer add and subtract.

The maximum value of the result of integer division must be less than $2^{48} - 1$. A value larger than $2^{48} - 1$ will cause high-order bits to be lost in the operation. No diagnostic is provided for the latter case. -0 is an integer constant as well as 0.

0 is 000000000000000000000
-0 is 777777777777777777778

Examples

67
281474976710655
2
-314159269
576460752303423487

The word structure for an integer constant is as follows:
Octal numbers are integer numbers in the base 8 system. Octal is used because of its easy and explicit conversion to and from binary. An octal constant consists of 1 to 20 octal digits preceded by a sign. An octal number is followed by a B suffix. The form is nB where n is the octal number. If the constant exceeds 20 digits, the compiler diagnostic is AN OCTAL NUMBER HAS MORE THAN 20 DIGITS. If a non-octal digit appears, the compiler diagnostic is ILLEGAL CHARACTER APPEARS IN A CONSTANT.

In FORTRAN there is no interpretation of octal in the current standard. Note that octal is used as integer type.

**Examples**

Note that the B convention right-justifies the octal number in the computer word, and the word is zero-filled to the left. All octal constants should be type integer.

<table>
<thead>
<tr>
<th>FORTRAN Statement</th>
<th>60-Bit Word Defined</th>
</tr>
</thead>
<tbody>
<tr>
<td>I = 2374216B</td>
<td>00000000000002374216</td>
</tr>
<tr>
<td>KK = 777776B</td>
<td>00000000000000777776</td>
</tr>
<tr>
<td>MASK = 7770077700077B</td>
<td>000007770007770077</td>
</tr>
</tbody>
</table>

The word structure is
REAL CONSTANTS

A real constant is represented by a string of up to 15 decimal digits. A real constant contains a decimal point and may contain an exponent representing a power of 10. The largest real number defined in the computer is 1.26501408317069E+322, base 10, (377677777777777777778). A variable location may contain this number. If a constant is equal to or greater than 1.E+322, a compiler diagnostic is provided (CONSTANT OUT OF RANGE). However, the higher number will work in all floating point arithmetic units. When a number calculated in the computer exceeds this range, RRRRR, indicating a range error, is printed from a WRITE statement.

The smallest positive real constant is 3.13151306251401E-294, base 10, (000140000000000000008). If a real constant is less than this number, the constant is set to zero. No compiler diagnostic is provided.

A positive infinity in a computer word is 3777XXXXXXXXXXXXXXXXX8. A negative infinity is the 7’s complement of positive infinity 4000XXXXXXXXXXXXXXXXX8. An indefinite number is 1777XXXXXXXXXXXXXXXXX8 in the computer word. A negative indefinite is the complement 6000XXXXXXXXXXXXXXXXX8.

Examples

The number is n; s is the exponent to the base 10.

<table>
<thead>
<tr>
<th>n.Es</th>
<th>3.E1 (means 3.0 \times 10^1; i.e., 30.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>n.n</td>
<td>3.1415768</td>
</tr>
<tr>
<td>n.</td>
<td>314.</td>
</tr>
<tr>
<td>.n</td>
<td>.0749162</td>
</tr>
<tr>
<td>n.nE+s</td>
<td>-3.141592E+279</td>
</tr>
<tr>
<td>n.nE-s</td>
<td>31.41592E-01</td>
</tr>
<tr>
<td>.nEs</td>
<td>.31415E01</td>
</tr>
<tr>
<td>.nE+s</td>
<td>.31515E+01</td>
</tr>
<tr>
<td>nEs</td>
<td>52E6</td>
</tr>
</tbody>
</table>
REAL CONSTANTS
(continued)

The word structure of a floating point constant is

A constant of zero is the same as 0.0 in the computer word. However, do not mix mode in defining zeros. Use A = 0. (real) or IA = 0 (integer). The compiler will generate an extra instruction to pack or unpack the zero depending on whether or not the decimal point is present, even through its value is the same.

DOUBLE-PRECISION
CONSTANTS

A double-precision constant is represented by a string of up to 29 decimal digits. The forms are similar to real constants, where D is used to prefix the exponent.

.nDs n.nDs n.Ds

The number is n; s is the exponent to the base 10.

Double-precision constants are represented internally by two words. The D must always appear. The plus sign may be omitted for positive s.

The largest double-precision constant is
1.2650140831706913647030959170D+322 (base 10).

37767777777777777777777777
37167777777777777777777777

word 1
word 2

If a constant exceeds this range, the compiler diagnostic will be CONSTANT OUT OF RANGE. When a number calculated in the computer exceeds this range, RRRRR, indicating a range error, is printed from a WRITE statement.
The smallest double-precision constant is 
\[ 3.1315130625140199656189188302D-29 \] (base 10).

\[ \begin{align*}
000140000000000000000 & \text{ word 1} \\
000000000000000000000 & \text{ word 2}
\end{align*} \]

Numbers smaller than \(10^{-28}\) have word 2 of the double-precision constant set to zero. Numbers smaller than \(10^{-29}\) have both word 1 and word 2 set to zero. No compiler diagnostic is provided.

**Examples**

\[
\begin{align*}
n.nDs & \quad 3.14159254358979323D0 \\
n.nDs & \quad 2.1843D0 \\
n.nDs & \quad 1.0 \\
n.nD-s & \quad 3141.592D-03 \\
n.nDs & \quad 3141.592D3
\end{align*}
\]

Two computer words define a double-precision constant: word 1 is the most significant part, word 2 the least significant part.

The structure of the two computer words defining a double-precision constant is
A complex constant is an ordered pair of real constants separated by a comma and enclosed in parentheses \((R_1, R_2)\). \(R_1\) represents the real part of the complex number, and \(R_2\) represents the imaginary part. Either constant may be preceded by a minus sign. The two constants in the complex constant are real numbers and have the same range as real numbers. If the range of the real numbers comprising the constant is exceeded, the compiler diagnostic is CONSTANT OUT OF RANGE. Diagnostics also occur when the number pair consists of integer constants, including \((0,0)\).

There are two words in a complex constant: word 1 is the real part of the complex number, word 2 the imaginary part.

**Examples**

<table>
<thead>
<tr>
<th>FORTRAN Representation</th>
<th>Complex Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1., 6.55)</td>
<td>1. + 6.55i</td>
</tr>
<tr>
<td>(15., 16.7)</td>
<td>15. - 16.7i</td>
</tr>
<tr>
<td>(-14.09, 1.65E-04)</td>
<td>-14.09 + .000165i</td>
</tr>
<tr>
<td>(0., -1.)</td>
<td>-i</td>
</tr>
</tbody>
</table>

The structure of the two words defining a complex constant is
HOLLERITH CONSTANTS

A Hollerith constant (or alphanumeric constant) is a string of FORTRAN characters of the form nHf; n is an unsigned decimal integer representing the length of the field f. One computer word will hold 10 Hollerith characters. Spaces are significant in the field f. When n is not a multiple of 10, the last computer word is left-justified with spaces filling the remainder of the word. The alternate form is nRf, which is interpreted as a right-justified Hollerith constant with zero-fill for incomplete words. A Hollerith constant of the form L = (5HABCDE) is also permitted.

Hollerith constants may be used in arithmetic and DATA statements. They are stored internally in console display code.

Rules:

- In an arithmetic replacement statement, such as
  \[ L = 4HABCD, \] n may not be greater than 10.

- In a DATA statement, n may not exceed 150.

\[
\text{DIMENSION L(13)}
\]
\[
\text{DATA(L=130H ........}
\]
\[
1....................
\]
\[
2.............)
\]

When the H specification is greater than 10 characters, the array defined must be appropriately dimensioned; in this case L must be dimensioned 13.

- A Hollerith constant of the form \( K = (+5HABCDE) \) is also permitted.
Rules (continued)

- Hollerith constants in a replacement statement are assumed by the compiler to be type integer.

- In a CALL statement, a Hollerith constant may not exceed 150 characters.

Examples in a Replacement Statement

<table>
<thead>
<tr>
<th>FORTRAN Statement</th>
<th>Word in Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>M = 6HABCDE</td>
<td>0102030405555555555555</td>
</tr>
<tr>
<td>K = 1HR</td>
<td>2255555555555555555555</td>
</tr>
<tr>
<td>L(1) = 10HTEMPERATUR</td>
<td>24051520052201242522</td>
</tr>
<tr>
<td>L(2) = 10HE</td>
<td>0555555555555555555555</td>
</tr>
<tr>
<td>K = 5H12345</td>
<td>3435363740555555555555</td>
</tr>
<tr>
<td>J = 5R12345</td>
<td>00000000003435363740</td>
</tr>
</tbody>
</table>

Examples in a DATA Statement

<table>
<thead>
<tr>
<th>FORTRAN Statement</th>
<th>Word in Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA M/3HSUM,4HMEAN/</td>
<td>2325155555555555555555</td>
</tr>
<tr>
<td></td>
<td>1505011655555555555555</td>
</tr>
<tr>
<td>DATA L/11HTEMPERATURE/</td>
<td>24051520052201242522</td>
</tr>
<tr>
<td></td>
<td>0555555555555555555555</td>
</tr>
</tbody>
</table>

Logical Constants

Logical constants are .TRUE. or .FALSE. .TRUE. is a word filled with 7's. This is -0. Any non-zero or minus-zero value is true. .FALSE. is a word all zero. This is +0.
A variable in FORTRAN is a symbolic name used to identify a quantity stored in the computer, which may assume different values. The value of a variable may change during execution of a program.

The symbolic representation, that is, the variable name, is a reference to the location of the quantity. The value of that quantity is contained in the location represented by the variable name.

Example

\[ AB = 10.5 \]

AB is the variable name or location of the number 10.5. 10.5 is the number in AB, and is referred to as the contents of AB.

Variable names may have from one to six alphanumeric characters, and the first character must be a letter.

These names may be characterized as one of the following types:

- Integer
- Real
- Double-precision
- Complex
- Logical
VARIABLES

Unless the type of a variable name is declared in a TYPE or IMPLICIT statement (see chapter 5), the compiler assumes the name to be integer or real according to the following convention:

Assumed from name

Variables are assumed to be integer if the first character of their symbolic name is one of the characters I, J, K, L, M, or N. All other variables are assumed to be real.

Typed

The programmer may override the rule and specify any of the five types by using the FORTRAN TYPE declaration.

<table>
<thead>
<tr>
<th>Type</th>
<th>Statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>INTEGER A, B, K</td>
</tr>
<tr>
<td>Floating point (real)</td>
<td>REAL I, J, T</td>
</tr>
<tr>
<td>Complex</td>
<td>COMPLEX A, C, C1, N</td>
</tr>
<tr>
<td>Double-precision</td>
<td>DOUBLE D, DD, S, L</td>
</tr>
<tr>
<td>Logical</td>
<td>LOGICAL L1, LT</td>
</tr>
</tbody>
</table>

TYPES OF VARIABLES

Integer Variables

An integer variable may be typed explicitly with a TYPE declaration, or assumed to be of integer type if the first letter in the name is I, J, K, L, M, or N. Each simple integer variable occupies one word in storage and assumes an exact integer value.

N
ITEM
M58A
ITEST
L2
Real Variables

Real variables may be typed explicitly or assumed to be real variables if the first letter is not I, J, K, L, M, or N. Each real variable occupies one word in storage and is stored in normalized floating point format. It is an approximation to a real number.

```
VECTOR
X
DERIV
A4S9
```

Double-Precision Variables

Double-precision variables must be typed as DOUBLE explicitly in a FORTRAN TYPE declaration. Each double-precision variable occupies two words of storage.

Complex Variables

Complex variables must be explicitly defined in a FORTRAN TYPE declaration. A complex variable occupies two words in storage. Each word contains a number in FORTRAN real variable format. The ordered pair of real variables represents the complex number \( x_1 + ix_2 \).

Naming Octal Constants

Octal constants should be defined using integer variable names to avoid having the compiler float the number if modes are mixed.

```
IA = 25B
IB = 0155B
MASK = 7777777B
```
2.16
FORTRAN Concepts

TYPES OF VARIABLES
(continued)

Alphanumeric Names (Hollerith)
Names of variables containing alphanumeric messages or labels should have integer type names.

\[
\begin{align*}
\text{NAME} &= 4\text{MARY} \\
\text{M2S(1)} &= 10\text{TEMPERATUR} \\
\text{M2S(2)} &= 1\text{HE}
\end{align*}
\]

Logical Variables
Logical variables may be logical or integer type.

CLASSES OF VARIABLES
The FORTRAN compiler recognizes two classes of variables, simple and subscripted. A simple variable is one that is single-valued; a subscripted variable is multiple-valued.

Simple Variable
A simple variable refers to a single storage location containing one quantity. The value specified by the name is always the current value stored in that location.

\[
\begin{align*}
\text{FRAN} \\
\text{PTEST} \\
\text{E9T} \\
\text{MAT} \\
\text{I3} \\
\text{K}
\end{align*}
\]

If A has been set equal to 4.5 and is used as a variable in an equation, the value of A (which is 4.5) is used in the equation.
A subscripted variable is an alphanumeric identifier with one, two, or three associated subscripts enclosed in parentheses. It is called an array. The value of the array element is the quantity stored in the name modified by the subscripts specified.

\[
\begin{align*}
A(1) \\
B(2,3) \\
LTV(2,1,4) \\
A5(1,10,8)
\end{align*}
\]

If more than three subscripts appear, the compiler diagnostic given is **GREATER THAN THREE DIMENSIONS IN DIMENSION STATEMENT**.

If a subscripted variable appears in a replacement statement with more subscripts than were declared in the DIMENSION statement, the diagnostic **TOO MANY SUBSCRIPT INDICES** is given.

A standard subscript has one of the following forms: \(a\) and \(b\) are unsigned integer constants and \(I\) is a simple integer variable. Use of standard subscripts produces the most efficient code for indices.

<table>
<thead>
<tr>
<th>General Form</th>
<th>Standard Subscripted Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a*I+b)</td>
<td>(A(I,J,K))</td>
</tr>
<tr>
<td>(I*a+b)</td>
<td>(R(I*3+2))</td>
</tr>
<tr>
<td>(b+I*a)</td>
<td>(R(3+I*2))</td>
</tr>
<tr>
<td>(b+a*I)</td>
<td>(R(3+4*K))</td>
</tr>
<tr>
<td>(a+I)</td>
<td>(R(5+L))</td>
</tr>
<tr>
<td>(I+b)</td>
<td>(B(I-1,J+1,3*K-1))</td>
</tr>
<tr>
<td>(a*I)</td>
<td>(D(9))</td>
</tr>
<tr>
<td>(I)</td>
<td>(XYZ(2*K+4))</td>
</tr>
<tr>
<td>(a)</td>
<td>(T(2,1))</td>
</tr>
</tbody>
</table>
CLASSES OF VARIABLES (continued)

Subscripted Variable (continued)

A nonstandard subscript is any arithmetic expression other than the standard forms used as a subscript. If the expression is floating point, the result is truncated and used as the integer index.

Examples

Nonstandard Subscripted Variables:

A(MAXF(I,J,M))
B(J,SINF(J))
MAT(3*K*LIM+3.5)
Q(1,-4,-2)
T(X,Y)

Limitation

- Subscripted subscripts are not allowed.

Nonstandard subscripts generate inefficient code from the compiler. Efficient code results from any of the standard forms listed above.
Arrays

An array is a named and ordered set of data stored in the computer under a symbolic name.

A single element of this array is referenced by the array name plus one, two, or three subscripts. The array name must appear in a DIMENSION statement. A TYPE or COMMON declaration may also be used, if it is appropriate.

The entire array may be referenced using the array name without subscripts in I/O lists where implied DO loop notation prints the entire array as specified by the DIMENSION declaration in a DIMENSION, TYPE, or COMMON statement. In an arithmetic expression, an array name without subscripts is taken to be the first element only.

There are no diagnostics if the subscript value is greater than the dimension bound declared for that name. Program execution may be in error as a result. (Any address in the program which tries to go outside the total core used by this program will cause an execution error. The program terminates with an SCM DIRECT RANGE diagnostic on the 7600.)
Array Structure

Array elements are stored by columns in consecutive sequential storage locations. The array dimensioned as A(3,3,3) appears as follows:

\[
\begin{array}{c c c}
A_{11} & A_{12} & A_{13} \\
A_{21} & A_{22} & A_{23} \\
A_{31} & A_{32} & A_{33} \\
\end{array}
\]

\[
\begin{array}{c c c}
A_{11} & A_{12} & A_{13} \\
A_{21} & A_{22} & A_{23} \\
A_{31} & A_{32} & A_{33} \\
\end{array}
\]

The planes are stored in order, starting with the first, as follows:

- \( A_{11} \rightarrow L \)
- \( A_{21} \rightarrow L+1 \)
- \( A_{31} \rightarrow L+2 \)
- \( A_{12} \rightarrow L+3 \)
- \( A_{22} \rightarrow L+4 \)
- \( A_{32} \rightarrow L+5 \)
- \( A_{13} \rightarrow L+6 \)
- \( A_{23} \rightarrow L+7 \)
- \( A_{33} \rightarrow L+8 \)

Array allocation is discussed under DIMENSION declaration in chapter 5. The location of an array element with respect to the first element is a function of the maximum array dimensions and the type of the array.

Given DIMENSION \( A(L,M,N) \), the location of \( A(i,j,k) \), with respect to the first element \( A \) of the array, is given by

\[
A + (i-1 + L*(j-1+M*(k-1)))\cdot E
\]
The quantity enclosed by the outer parentheses is the subscript expression. E is the element length; i.e., the number of storage words required for each element of the array. For real and integer arrays, E = 1. For complex and double-precision arrays, E = 2.

Note: A left shift rather than a multiply will be used in calculating a single element when the dimension is 64 or less and a power of 2. Use of shifts on higher powers of 2 may also occur.

Example

In an array defined by DIMENSION A(3,3,3) where A is real, the location of A(2,2,3) with respect to A(1,1,1) is

\[ A(2,2,3) = A(1,1,1) + (2-1+3(1+3(2))) \]
\[ = A(1,1,1) + 22 \]

FORTRAN permits the following relaxation of the representation of subscripted variables (where \( N_i \) are integer constants):

Given \( A(N_1,N_2,N_3) \),

then \( A(I,J,K) \) implies \( A(I,J,K) \)
\( A(I,J) \) implies \( A(I,J,1) \)
\( A(I) \) implies \( A(I,1,1) \)
\( A \) implies \( A(1,1,1) \)
CLASSES OF VARIABLES
(continued)

Array Structure
(continued)

Similarly, given \( A(N_1,N_2) \),
then \( A(I,J) \) implies \( A(I,J) \)
\( A(I) \) implies \( A(I,1) \)
\( A \) implies \( A(1,1) \)

and given \( A(N_1) \),
then \( A(I) \) implies \( A(I) \)
\( A \) implies \( A(1) \)

The elements of a single-dimensioned array \( A(N) \) may not be referred to as \( A(I,J,K) \) or \( A(I,J) \). Similarly, the elements of a double-dimensioned array \( A(M,N) \) may not be referred to as \( A(I,J,K) \). The diagnostic in such cases is TOO MANY SUBSCRIPT INDICES.
III

EXPRESSIONS
ARITHMETIC, LOGICAL
MASKING, MIXED MODE

EVALUATION
OPTIMIZATION

exp
Three kinds of expressions are recognized by the 7600 FORTRAN compiler: arithmetic expressions; logical expressions, which may be true or false; and masking expressions which, when evaluated by the computer, have numerical values. For each type of expression there is an associated group of operators and operands.
Expressions

An expression may contain constants, variables (simple or subscripted), functions, and combinations of these separated by operators and/or parenthesis.

Arithmetic Expressions

An arithmetic expression may contain the following operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>A + B</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>C - D + 1.324</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>C*D</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>C*D + A/D</td>
</tr>
<tr>
<td>**</td>
<td>Exponentiation†</td>
<td>A**B + 3.1415</td>
</tr>
</tbody>
</table>

An operator performs arithmetic using operands. An operand may be a constant, a variable (simple or subscripted), the result of the evaluation of a function or the result of the evaluation of an expression.

In the expression \((A**B + 3.1415)\), \(A\), \(B\), and 3.1415, as well as the result of \(A**B\) are operands, while \(**\) and \(+\) are operators. When all operations are completed as indicated in the expression, the expression is said to be evaluated and thus has a numerical value. Operands in an arithmetic expression may be separated by only one operator \((X op op Y\) is not valid). The only exception is unary negation, e.g. \(A*(-B)\).

Examples

\[
\begin{align*}
T & \quad (B - \sqrt{B^2 - (f*A*C)})/(2.0*A) \\
3.14159 & \quad \text{GROSS} - (\text{TAX} * 0.05) \\
B + 16.427 & \quad (\text{TEMP} + V(M, \text{MAXF}(A, B))^{**C})/(H - \text{FACT}(K + 3)) \\
(XBAR + (B(I,J+I,K)/3.)) & \quad A + (-B) \\
-(C + \text{DELTA*AERO}) & \\
\end{align*}
\]
The hierarchy of arithmetic evaluation is:

- ** Exponentiation (class 1)
- / Division
- * Multiplication
- + Addition
- - Subtraction
- Unary negation (class 3)

Expressions with no embedded parentheses which contain unlike classes of operators are evaluated in the above order beginning with class 1. Expressions which contain operators of like classes are evaluated from left to right. For example, A**B**C is evaluated as (A**B)**C.

Parenthetical and function expressions (the ** operator is a function expression) are evaluated first in a left-to-right scan of the entire statement. In parenthetical expressions within parenthetical expressions, evaluation begins with the innermost expression. Parenthetical expressions are evaluated as they are encountered in the left-to-right scanning process.

When writing an integer expression, it is important to remember not only the left-to-right scanning process, but also that dividing an integer quantity by an integer quantity always yields a truncated result; thus, when the expression 1/3 is evaluated, the result is 0.

The expression I*J/K will yield a result different from the expression J/K*I because of the order of operations and truncation in each case. For example; 4 * 3/2 means 4 times 3 (or 12) divided by two resulting in 6. 3/2 * 4 means 3/2 (truncated to 1) times 4 resulting in 4.
The NCAR compiler described in this manual will optimize code to produce more efficient machine language object code.

Arithmetic expression evaluation without optimization will apply only if the *FORTRAN card specifies that no optimization should be done by the compiler (see page 3.6).

**Example 1**

\[ A^{**}B/C+D^{*}E^{*}F-G \]

is evaluated:

\[ A^{**}B \rightarrow R_1 \]
\[ R_1/C \rightarrow R_2 \]
\[ D^{*}E \rightarrow R_3 \]
\[ R_3^{*}F \rightarrow R_4 \]
\[ R_4+R_2 \rightarrow R_5 \]
\[ R_5-G \rightarrow R_6 \] evaluation completed.

**Example 2**

\[ A^{**}B/(C+D)^{*}(E^{*}F-G) \]

is evaluated:

\[ A^{**}B \rightarrow R_1 \]
\[ C+D \rightarrow R_2 \]
\[ E^{*}F-G \rightarrow R_3 \]
\[ R_1/R_2 \rightarrow R_4 \]
\[ R_4^{*}R_3 \rightarrow R_5 \] evaluation completed.

**Example 3**

\[ H(I3)+C(I,J+2)^{*}(COS(Z))^{**2} \]

is evaluated:

\[ COS(Z) \rightarrow R_1 \]
\[ R_1^{**2} \rightarrow R_2 \]
\[ R_2^{*}C(I,J+2) \rightarrow R_3 \]
\[ R_3^{*}H(I3) \rightarrow R_4 \] evaluation completed.
The following are examples of expressions with embedded parentheses.

**Example 1**

\[ A^* (B + ((C/D) - E)) \] is evaluated:

- \( C/D \rightarrow R_1 \)
- \( R_1 - E \rightarrow R_2 \)
- \( R_2 + B \rightarrow R_3 \)
- \( R_3 ^* A \rightarrow R_4 \) evaluation completed.

**Example 2**

\[ A^* (\text{SIN}(X) + 1.) - Z / (C^*(D - E + F)) \] is evaluated:

- \( \text{SIN}(X) \rightarrow R_1 \)
- \( R_1 + 1. \rightarrow R_2 \)
- \( D - E \rightarrow R_3 \)
- \( R_3 + F \rightarrow R_4 \)
- \( C^* R_4 \rightarrow R_5 \)
- \( A^* R_2 \rightarrow R_6 \)
- \( -Z \rightarrow R_7 \)
- \( R_7 / R_5 \rightarrow R_8 \)
- \( R_8 + R_6 \rightarrow R_9 \) evaluation completed.
Optimization of arithmetic sequences involves changing the order of operations so that the sequence is performed with speed and efficiency. The most efficient code does not always conform to conventional rules for expression evaluation. The *FORTRAN Standard* states that processors may evaluate the mathematically equivalent expression. Where the previous rules are important to the results, it is necessary for the programmer to indicate to the compiler that no reordering is to take place.

When a *FORTRAN* program is compiled, all optimization techniques are applied. If the order of evaluation described in the previous section must be followed precisely, the programmer must turn off the optimization on the *FORTRAN,n* card, using the following codes for n. A type double or type complex turns off the optimizer for all expressions throughout the routine in which the *TYPE* declaration appears.

- n=14 Do not optimize divide operators (this option automatically turns off 15 as well)
- 15 Do not optimize multiply operators
- 16 Do not optimize add and subtract operators
- 0 Apply 14, 15, and 16 to prevent reordering of the arithmetic sequence

**Expression:**  \( X = \frac{A}{B/C/D/E} \)

### With Optimization (Code 14)

<table>
<thead>
<tr>
<th>Expression</th>
<th>Without Optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>( B \times C \rightarrow R_1 )</td>
<td>( A/B \rightarrow R_1 )</td>
</tr>
<tr>
<td>( R_1 \times D \rightarrow R_2 )</td>
<td>( R_1/C \rightarrow R_2 )</td>
</tr>
<tr>
<td>( R_2 \times E \rightarrow R_3 )</td>
<td>( R_2/D \rightarrow R_3 )</td>
</tr>
<tr>
<td>( A/R_3 \rightarrow R_4 )</td>
<td>( R_3/E \rightarrow R_4 )</td>
</tr>
</tbody>
</table>

To optimize, a series of divides is replaced by multiplies which are faster.
Expression:  \[ X = A^B C D^E \]

With Optimization
\[
\begin{align*}
A^B & \rightarrow R_1 \\
C^D & \rightarrow R_2 \\
R_1^R_2 & \rightarrow R_3 \\
R_3^E & \rightarrow R_4 \\
\end{align*}
\]

Without Optimization (Code 15)
\[
\begin{align*}
A^B & \rightarrow R_1 \\
R_1^C & \rightarrow R_2 \\
R_2^D & \rightarrow R_3 \\
R_3^E & \rightarrow R_4 \\
\end{align*}
\]

NCAR's compiler will optimize because of the pipeline characteristics of the functional units in the Control Data 7600 computer. Again, optimization is done to take advantage of the pipeline characteristics.

Expression:  \[ X = A + B - C + D \]

With Optimization
\[
\begin{align*}
A+B & \rightarrow R_1 \\
D-C & \rightarrow R_2 \\
R_1+R_2 & \rightarrow R_3 \\
\end{align*}
\]

Without Optimization (Code 16)
\[
\begin{align*}
A+B & \rightarrow R_1 \\
R_1-C & \rightarrow R_2 \\
R_2+D & \rightarrow R_3 \\
\end{align*}
\]

Expression:  \[ X = A^B C^D E^F G^H \]

With Optimization
\[
\begin{align*}
A^B & \rightarrow R_1 \\
C^D & \rightarrow R_2 \\
E^F & \rightarrow R_3 \\
G^H & \rightarrow R_4 \\
R_1^R_2 & \rightarrow R_5 \\
R_3^R_4 & \rightarrow R_6 \\
R_6^R_5 & \rightarrow R_7 \\
\end{align*}
\]

Without Optimization (Code 15)
\[
\begin{align*}
A^B & \rightarrow R_1 \\
R_1^C & \rightarrow R_2 \\
R_2^D & \rightarrow R_3 \\
R_3^E & \rightarrow R_4 \\
R_4^F & \rightarrow R_5 \\
R_5^G & \rightarrow R_6 \\
R_6^H & \rightarrow R_7 \\
\end{align*}
\]

Notice that resultants are also optimized under code 16.
Expressions

EVALUATION WITH OPTIMIZATION
(continued)

Expression:
F = A*B*X/Y

<table>
<thead>
<tr>
<th>With Optimization</th>
<th>Without Optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>X/Y → R₁</td>
<td>A*B → R₁</td>
</tr>
<tr>
<td>A*B → R₂</td>
<td>R₁*X → R₂</td>
</tr>
<tr>
<td>R₂*R₁ → R₃</td>
<td>R₂/Y → R₃</td>
</tr>
</tbody>
</table>

To optimize, the slower divide operation is moved up in the sequence so that R₁ and R₂ are done simultaneously with the divide covering the multiply.

Expression:
X = A+B+C+D+E

<table>
<thead>
<tr>
<th>With Optimization</th>
<th>Without Optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>A+B → R₁</td>
<td>A+B → R₁</td>
</tr>
<tr>
<td>C+D → R₂</td>
<td>R₁+C → R₂</td>
</tr>
<tr>
<td>R₁+R₂ → R₃</td>
<td>R₂+D → R₃</td>
</tr>
<tr>
<td>R₃+E → R₄</td>
<td>R₃+E → R₄</td>
</tr>
</tbody>
</table>

This is done to optimize the use of the pipelined functional units.

Expression:
B = U*R + U*R + U*R

<table>
<thead>
<tr>
<th>With Optimization</th>
<th>Without Optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>U*R → R₁</td>
<td>U*R → R₁</td>
</tr>
<tr>
<td>R₁+R₁ → R₂</td>
<td>U*R → R₂</td>
</tr>
<tr>
<td>R₂+R₁ → R₃</td>
<td>R₁+R₂ → R₃</td>
</tr>
<tr>
<td></td>
<td>U*R → R₄</td>
</tr>
<tr>
<td></td>
<td>R₄+R₃ → R₅</td>
</tr>
</tbody>
</table>

To optimize, U*R is calculated only once.
Expression:  \( U^T U^* + \frac{A}{R} \)

*With Optimization*

\[
\begin{align*}
    A/R & \rightarrow R_1 \\
    U^T & \rightarrow R_2 \\
    R_2^* R_2 & \rightarrow R_3 \\
    R_3 + R_1 & \rightarrow R_4
\end{align*}
\]

In this case \( U^T \) is calculated only once. The slow divide operation is moved forward in the string.

*Without Optimisation as on *FORTRAN, 14, 15*

or *FORTRAN, 14:*

\[
\begin{align*}
    U^T & \rightarrow R_1 \\
    R_1 U & \rightarrow R_2 \\
    R_2^* T & \rightarrow R_3 \\
    A/R & \rightarrow R_4 \\
    R_3 + R_4 & \rightarrow R_5
\end{align*}
\]

Here, no divide or multiply is optimized. The divide option turns off the multiply as well.

*Without Optimisation as on *FORTRAN, 15:*

\[
\begin{align*}
    A/R & \rightarrow R_1 \\
    U^T & \rightarrow R_2 \\
    R_2^* U & \rightarrow R_3 \\
    R_3^* T & \rightarrow R_4 \\
    R_4 + R_1 & \rightarrow R_5
\end{align*}
\]

In this case, the multiply option is turned off with a 15 on the *FORTRAN* card, so the divide option (14) which is not turned off applies, and the divide is brought forward.
3.10 Expressions

**EVALUATION WITH OPTIMIZATION (continued)**

**Expression:** \( C = A^aB - E/R \)

<table>
<thead>
<tr>
<th>With Full Optimization</th>
<th>Without Optimization (Codes 14, 15 or Code 14)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( E/R \rightarrow R_1 )</td>
<td>( A^aB \rightarrow R_1 )</td>
</tr>
<tr>
<td>( A^aB \rightarrow R_2 )</td>
<td>( -E \rightarrow R_2 )</td>
</tr>
<tr>
<td>( R_2 - R_1 \rightarrow R_3 )</td>
<td>( R_2/R \rightarrow R_3 )</td>
</tr>
<tr>
<td></td>
<td>( R_3 + R_1 \rightarrow R_4 )</td>
</tr>
</tbody>
</table>

The 14 option turns off both 14 and 15.

**Expression:** \( X = A^aB^cC^d + G/H^aP/Y - R + S \)

<table>
<thead>
<tr>
<th>With Optimization</th>
<th>Without Optimization (Code 0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C^aP \rightarrow R_1 )</td>
<td>( A^aB \rightarrow R_1 )</td>
</tr>
<tr>
<td>( H^aY \rightarrow R_2 )</td>
<td>( R_1^aC \rightarrow R_2 )</td>
</tr>
<tr>
<td>( R_1/R_2 \rightarrow R_3 )</td>
<td>( R_2^aD \rightarrow R_3 )</td>
</tr>
<tr>
<td>( A^aB \rightarrow R_4 )</td>
<td>( G/H \rightarrow R_4 )</td>
</tr>
<tr>
<td>( C^aD \rightarrow R_5 )</td>
<td>( R_4^aP \rightarrow R_5 )</td>
</tr>
<tr>
<td>( S-R \rightarrow R_6 )</td>
<td>( R_5/Y \rightarrow R_6 )</td>
</tr>
<tr>
<td>( R_4^aR_5 \rightarrow R_7 )</td>
<td>( R_6^aR_3 \rightarrow R_7 )</td>
</tr>
<tr>
<td>( R_7 + R_3 \rightarrow R_8 )</td>
<td>( R_7 - R \rightarrow R_8 )</td>
</tr>
<tr>
<td>( R_9 + R_6 \rightarrow R_9 )</td>
<td>( R_8 + S \rightarrow R_9 )</td>
</tr>
</tbody>
</table>
LOGICAL EXPRESSIONS

A logical expression has the form:

```
exp1 rop exp2
```

RELATIONAL OPERATORS

The exps are arithmetic expressions, evaluated according to rules set down in the previous section; rop is one of the following relational operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>.EQ.</td>
<td>Equal to</td>
</tr>
<tr>
<td>.NE.</td>
<td>Not equal to</td>
</tr>
<tr>
<td>.GT.</td>
<td>Greater than</td>
</tr>
<tr>
<td>.GE</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>.LT.</td>
<td>Less than</td>
</tr>
<tr>
<td>.LE.</td>
<td>Less than or equal to</td>
</tr>
</tbody>
</table>

The logical expression is true if the arithmetic expressions satisfy the relation specified by rop; otherwise, it is false.

```
C = B.LT.(I*COS(X-1.0))
```

Logical expressions are evaluated as illustrated in the example, A.EQ.B. This is equivalent to the question: does \(A - B = 0\)?

The following table shows how the 7600 compiler treats the logical expressions. In the following expressions, \(A0 = 0\), \(A1 > 0\).

<table>
<thead>
<tr>
<th>Computer Calculation</th>
<th>Test</th>
<th>Evaluated Expression (in octal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0.EQ.A1</td>
<td>A0 - A1</td>
<td>Not zero</td>
</tr>
<tr>
<td>A0.NE.A1</td>
<td>A0 - A1</td>
<td>Zero</td>
</tr>
<tr>
<td>A0.GT.A1</td>
<td>A1 - A0</td>
<td>Zero or plus</td>
</tr>
<tr>
<td>A0.GE.A1</td>
<td>A0 - A1</td>
<td>Minus (other than -0)</td>
</tr>
<tr>
<td>A0.LT.A1</td>
<td>A0 - A1</td>
<td>Zero or plus</td>
</tr>
<tr>
<td>A0.LE.A1</td>
<td>A1 - A0</td>
<td>Minus (other than -0)</td>
</tr>
</tbody>
</table>
The difference is computed and tested. If the test is satisfied, the relation is true and the expression is assigned the value -0 or .TRUE. (77777777777777777777). If the relation is false, the expression is set to 0 or .FALSE.

If A is true, A is set to 77777777777777777777B
If A is false, A is set to 00000000000000000000B

Logical expressions of the following forms are allowed:

I.LT.R , I.LT.D , I.LT.C

where I=integer, R=real, D=double, and C=complex. An expression of the form I.GE.0 is treated as being true if I assumes the value -0.

Logical expressions are converted internally to arithmetic expressions according to the rules of mixed-mode arithmetic (see section below on mixed mode expressions). When complex expressions are tested, only the real part is used in the comparison.

- A₁ rop A₂ rop A₃ ... is not a valid logical expression (see logical connectives, p. 3.13).
- A logical expression of the form A₁ rop A₂ is evaluated from left to right. The logical expressions A₁ rop A₂, A₁ rop (A₂), and (A₁) rop (A₂) are equivalent.
- A false logical expression is assigned the value +0; a true logical expression is assigned the value -0.
- Relational operators may not be used in multiple replacement statements.
- In a logical replacement statement, the mode of the resultant is ignored; the result is true or false (i.e., -0 or +0).
Examples

A.LT.16.
T - Q(I)*Z.LE.3.141592
B - C.NE.D + E
R(I).GE.R(I-1)
K.LT.16
I.EQ.J(K)
(I).EQ.(J(K))

LOGICAL CONNECTIVES

Logical connectives may be used with relational operators in the parenthetical expressions used in an IF statement. A logical expression may be of the form

\[ \text{IF}(\text{lexp}_1 \text{ con } \text{lexp}_2 \ldots)_{n_1,n_2} \]

The terms lexp\(_i\) are logical expressions as shown in the examples in the previous section, where lexp\(_i\) = A\(_i\) rop A\(_2\) and con is a connective belonging to the following set:

<table>
<thead>
<tr>
<th>Connective</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>.NOT.</td>
<td>Negation (this does not mean complement)</td>
</tr>
<tr>
<td>.AND.</td>
<td>Conjunction</td>
</tr>
<tr>
<td>.OR.</td>
<td>Disjunction</td>
</tr>
</tbody>
</table>

Connectives are defined as follows:

- .NOT.REL\(_1\) Is false if and only if REL\(_1\) is true
- REL\(_1\).AND.REL\(_2\) Is true if and only if REL\(_1\) and REL\(_2\) are both true
- REL\(_1\).OR.REL\(_2\) Is false if and only if REL\(_1\) and REL\(_2\) are both false (this is not an exclusion or)
Expressions

LOGICAL CONNECTIVES (continued)

- Results of logical expressions in an IF statement are either true or false:

  True  77777777777777777777
  False  00000000000000000000

- The scan is based on the connectives as they appear from left to right. Evaluation terminates as soon as the truth values can be established for an expression.

- Where .NOT. is used alone, as in (.NOT.A.EQ.B), no negation is involved. The compiler reverses the test (i.e., (A.NE.B)) in order to set truth values.

- Logical variables may be set true or false in a replacement statement by declaring:

  L1 = .TRUE.
  L2 = .FALSE.

- REL con1 con2 REL is never legal where con1 is either .AND. or .OR.

- In a combined logical expression with connectives, the resultant is set to true or false regardless of mode

  \[ B - C \leq A < B + C \]

  is written as

  \[ B - C \text{ LE} A \text{ AND} A \text{ LT} B + C \]

MASKING EXPRESSIONS

In a FORTRAN masking expression, 60-bit logical arithmetic is performed bit by bit on the operands within the expression. The operands may be real- or integer-type variables or constants. No mode conversion is performed during evaluation. Although the form of masking operators is the same as logical connectives, the meanings are very different. They are defined below, listed according to hierarchy of operation.
Expressions

.NOT. Complement the operand bit by bit

.AND. Form the bit-by-bit logical product of two operands:

\[ \begin{array}{c|c|c|c|c}
   p & v & p \text{ .AND. } v \\
   \hline
   1 & 1 & 1 & 1 & 0 \\
   1 & 0 & 0 & 1 & 0 \\
   0 & 1 & 0 & 1 & 1 \\
   0 & 0 & 0 & 0 & 1 \\
\end{array} \]

.OR. Form the bit-by-bit logical sum of two operands:

\[ \begin{array}{c|c|c|c|c}
   p & v & p \text{ .OR. } v \\
   \hline
   0101 & 1101 & 1101 (this is not an exclusive or) \\
\end{array} \]

The operations are described below:

\[ \begin{array}{c|c|c|c|c|c|c}
   p & v & p \text{ .AND. } v & p \text{ .OR. } v & \text{ .NOT. } p \\
   \hline
   1 & 1 & 1 & 1 & 0 \\
   1 & 0 & 0 & 1 & 0 \\
   0 & 1 & 0 & 1 & 1 \\
   0 & 0 & 0 & 0 & 1 \\
\end{array} \]

Example 1

Let \( \text{MASK1} = 77007700000000000000B \)
\( \text{MASK2} = 7777B \)
\( \text{MASK3} = 77770000000000000000B \)
\( \text{IB} = 10\text{HAAAAAMARY} \) \( (0101010101015012231B \text{ in octal}) \)
\( \text{CI} = 1.0 \) \( (17204000000000000000B \text{ in octal}) \)

Then \( \text{.NOT.} \text{MASK1} \) is 00770077777777777777B
\( \text{MASK3} \text{.AND.} \text{CI} \) is 17200000000000000000B
\( \text{MASK2} \text{.OR.} \text{MASK3} \) is 77770000000000007777B
\( \text{MASK2} \text{.AND.} \text{IB} \) is 000000000000002231B

Example 2

Let \( \text{MASK1} = 77770000777700007777B \)
\( \text{MASK2} = 10\text{HABCDABCDAB} \) \( (01020304010203040102B \text{ in octal}) \)
\( \text{A} = 1.0 \) \( (17204000000000000000B \text{ in octal}) \)
\( \text{I} = 00000000001111111111B \)

Then the resultant for

\( \text{R} = \text{MASK1} \text{.AND.} \text{MASK2} \) is 010200000010200000102B
\( \text{IJ} = \text{MASK2} \text{.OR.} \text{I} \) is 01020304011313151113B
\( \text{KI} = \text{A} \text{.OR.} \text{.NOT.} \text{I} \) is 77777777776666666666B
The user should note that these masking operations, plus the exclusive OR, can also be performed using the in-line functions AND, OR, COMPL, and XR described in the Library Routines Manual (NCAR Technical Note TN-IA-67). In general programs, using the masking functions rather than masking operators will be more portable.

**Rules**

- Let \( B \) be masking expressions, variables, or constants of any type. The following are masking expressions:
  
  \[
  .NOT. B_1 \quad B_1 .AND. B_2 \quad B_1 .OR. B_2
  \]

- \( .NOT. \) may appear with \( .AND. \) or \( .OR. \) only as follows:
  
  \[
  \begin{align*}
  .AND. &. NOT. \\
  .OR. &. NOT. \\
  .AND. & (. NOT. . . . ) \\
  .OR. & (. NOT. . . . )
  \end{align*}
  \]

- Masking expressions of the following forms are evaluated from right to left:
  
  \[
  A .AND. B .AND. C \quad \text{is evaluated as follows:}
  \]
  
  \[
  \begin{align*}
  B \quad .AND. \quad C+R_1 \\
  A \quad .AND. \quad R+R_2
  \end{align*}
  \]

- Masking expressions must not contain parenthetical arithmetic expressions, but any function subprogram call is allowed. If \( L5 = \text{MASK}.AND.X*I \) is written, the computer diagnostic is ILLEGAL USE OF A REPLACEMENT STATEMENT.

- Only real- and integer-type variables may be used. Other variable types result in the diagnostic MASKING OPNDS MUST BE REAL OR INTEGER.

- The masking forms of \( .AND. \), \( .OR. \), and \( .NOT. \) may not be used in an IF statement.

- Masking operations do not produce truth values.

- The mode of the variables in a masking statement is ignored.
Mixed-mode expressions are permitted using the NCAR FORTRAN compiler. In discussing mixed mode it is helpful to distinguish between simple and combined expressions. A simple expression does not contain parenthetical delimiters within the expression. $A^R/COS(X)$ and $C+D*I$ are simple expressions. A combined expression is an expression consisting of more than one simple expression including nests. $A^R(SIN(X)+1) - Z/(C^R(D-E+F))$ is an example of a combined expression.

A precedence of expressions and operators can be established for mixed expressions. This order is shown below.

<table>
<thead>
<tr>
<th>Expression Type</th>
<th>Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>** / * + -</td>
</tr>
<tr>
<td>Relational</td>
<td>.GT. , .GE. , .LT. , .LE. , .EQ. , .NE.</td>
</tr>
<tr>
<td>Logical connectives</td>
<td>.NOT. , .AND. , .OR.</td>
</tr>
<tr>
<td>Masking</td>
<td>.NOT. , .AND. , .OR.</td>
</tr>
</tbody>
</table>

In the following expression

$A+B.LT.C-D$

the arithmetic expressions are evaluated first, then the relational operators.

$A+B + R_1$
$C-D + R_2$
$R_1.LT.R_2 + R_3$

$R_3$ will have a truth value.
A table of permissible combinations of expression types with the mode of result may be helpful in interpreting mixed expressions.

**Rules**

- The order of dominance of the operand types within a simple expression from the highest to the lowest is
  
  Complex  
  Double  
  Real  
  Integer  
  Relational

  The dominant type is used for all arithmetic operations within a simple expression.

- All simple expressions in a combined expression are evaluated first, starting with the innermost nest of parentheses. The final scan uses the mode of the dominant resultant of all simple expressions or variables remaining after the initial scan.

- Double-precision and complex arithmetic are evaluated by in-line arithmetic functions.

- A simple arithmetic expression is evaluated in the mode of the dominant operand type.

- Expressions which contain Hollerith or octal constants are ordered as type integer in the order of dominance list above. If these are in a mixed expression of higher dominance, they will be converted to the appropriate type. (Using Hollerith constants in a mixed-mode expression is usually a mistake.)

- Expression optimization is never done by the compiler when there is mixed mode.
The operator ** combines constants, variables, expressions, and subscripted variables. The following list contains the combinations of mode that are allowed. Any combination other than these will result in the execution diagnostic ILLEGAL EXPONENT.

<table>
<thead>
<tr>
<th>Base</th>
<th>Exponent</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td>Real</td>
<td>Integer</td>
<td>Real</td>
</tr>
<tr>
<td>Real</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>Double</td>
<td>Integer</td>
<td>Double</td>
</tr>
<tr>
<td>Double</td>
<td>Real</td>
<td>Double</td>
</tr>
<tr>
<td>Double</td>
<td>Double</td>
<td>Double</td>
</tr>
<tr>
<td>Complex</td>
<td>Integer</td>
<td>Complex</td>
</tr>
</tbody>
</table>

Some illustrations of simple and combined expressions that may be used with the ** operator are given below:

- T**I
- I**(J - 1)
- RE(K,J + 1)**(2.1*COS(X))
- A**(I + K-1)
- (4.23D0 + 1)**(XTEMP)
- C**(I*K + 3)
- (4.23D0 + 1)**(1.D0 + 3)

Example 1

Given real A,B; integer I,J. The simple expression A*B - I + J is real because the dominant operand type is real. The expression is evaluated:

A*B → R₁ real
Convert I to real
Convert J to real
R₁ - I → R₂ real
R₂ + J → R₃ real evaluation completed.
Example 2

The use of parentheses can change the evaluation. Given $A*B-(I+J)$, the simple expression $(I+J)$ is evaluated first in integer. The evaluated expression $(I+J)$ is then converted to real. $A*B-(I+J)$ is evaluated:

$$I+J \rightarrow R_1 \text{ integer}$$

Convert $R_1$ to real

$$A*B \rightarrow R_2 \text{ real}$$

$R_2-R_1 \rightarrow R_3$ real evaluation completed.

Example 3

Given $A*B-J*K/J$. This simple expression is evaluated as follows where the dominant mode is real:

$$A*B \rightarrow R_1$$

Convert $J$ to real $\rightarrow R_2$

Convert $K$ to real

$$R_2 \# K$$

Bring in $J$ again and convert to real

$$R_2 \# K/J \rightarrow R_3$$

$R_3+R_1 \rightarrow R_4$ evaluation completed.

Example 4

Given $A*B-(J*K/J)$. This combined expression is evaluated as follows where the simple expression $(J*K/J)$ is evaluated as type integer:

$$J*K \rightarrow R_1$$

$R_1/J \rightarrow R_2$ (where result is type integer)

Convert $R_2$ to real $\rightarrow R_3$

$$A*B \rightarrow R_4$$

$R_4-R_3 \rightarrow R_5$ evaluation completed.
Example 5

Given complex C1, C2; real A, B, F. The combined expression A*(C1/C2) + B*F is complex. The expression is evaluated:

\[ \frac{C1}{C2} \rightarrow R_1 \text{ complex} \]
Convert A to complex
\[ A \cdot R_1 \rightarrow R_2 \text{ complex} \]
Convert B to complex
Convert F to complex
\[ B \cdot F \rightarrow R_3 \text{ complex} \]
\[ R_2 + R_3 \rightarrow R_4 \text{ complex evaluation completed.} \]

Example 6

Consider the expression C1/C2 + (A-B) where the operands are defined as the example above. The expression is evaluated:

\[ A-B \rightarrow R_1 \text{ real} \]
Convert R1 to complex
\[ \frac{C1}{C2} \rightarrow R_2 \text{ complex} \]
\[ R_1 + R_2 \rightarrow R_3 \text{ complex evaluation completed.} \]
3.22
Expressions

Rules
(continued)

Example 7

Given Cl*D + R / I where C1 is complex; D is double precision; R is real; I is integer. In this simple expression all variables are raised to the dominant operand type, and complex arithmetic is used in the entire expression.

Convert D to complex†
Cl*D + R1 complex
Convert R to complex
Convert I to complex
R / I → R complex
R1 + R2 → R3 complex evaluation completed.

Example 8

Given Cl*D + (R/I) where the variables are the same as in the example above. In this combined expression the simple expression (R/I) is evaluated first in its dominant mode which is real. The order and mode of the evaluation is as follows:

Convert I to real
R/I → R1 real
Convert R1 to complex
Convert D to complex
Cl*D → R2 complex
R1 + R2 → R3 complex evaluation complete.

† Truncate D to the most significant portion, which is used as the real part of a complex number; the imaginary part is set to zero.
IV

REPLACEMENT STATEMENTS

EVALUATION
MIXED MODE
MULTIPLE

\[ v = \exp \]
The general form of a replacement statement is $A = exp$, where $exp$ is an expression and $A$ is a variable name, simple or subscripted. The operator, $=$, means that the value of $A$ is replaced by the value of the evaluated expression, $exp$, with conversion for mode, if necessary. If $A$ is a logical variable, $exp$ must be a logical expression.

Examples

\[
A = -A \\
B(3,4) = \text{CALC}(I+1) \times \text{BETA} + 2.3478 \\
\text{XTHETA} = 7.4 \times \text{DELTA} + A(I,J,K) \times \text{BETA} \\
\text{RESPNS} = \sin(abar.inv + 2,jbar)) / \alpha(j) \\
\text{JMAX} = 19 \\
\text{AREA} = \text{SIDE1} \times \text{SIDE2} \\
\text{PERIM} = 2 \times (\text{SIDE1} + \text{SIDE2})
\]

The type of an evaluated expression is determined by the type of the dominant operand (see Chapter 3). This, however, does not restrict the types that identifier $A$ may assume in the replacement statement where $A = expression$. The converted result of a complex expression may replace $A$ when $A$ is real. The following table shows the $A = exp$ relationship for all the standard modes. The mode of $A$ determines the mode of the result; the dominant operand in the expression determines the mode of $exp$. 
4.2
Replacement Statements

**MIXED-MODE REPLACEMENT STATEMENTS**

*Given:* \( C_1, A_C \) complex

\( D_1, A_D \) double

\( R_1, A_R \) real

\( I_1, A_I \) integer

**Example 1**

\[ A_C = C_1 \times C_2 - C_3 / C_4 \]

\((6.90525, 15.39287) = (4.4, 2.1)(3.0, 2.0) - (3.3, 6.8)/(1.1, 3.4)\)

The expression is complex. Therefore, the result of the expression evaluation is a two-word, floating point quantity which represents a complex number. This complex quantity is stored in \( A_C \) as a two-word complex result, since \( A_C \) is type complex.

**Example 2**

\[ A_R = C_1 \]

\[ 4.4000 = (4.4, 2.1) \]

The expression is complex. \( A_R \) is real; therefore the real part of \( C_1 \) replaces \( A_R \).

**Example 3**

\[ A_R = C_1 \times (0., -1.) \]

\[ 2.100 = (4.4, 2.1) \times (0., -1.) \]

The expression is complex. \( A_R \) is real; the real part of the result of the complex multiplication replaces \( A_R \).

**Example 4**

\[ A_I = R_1 / R_2 \times (R_3 - R_4) + I_1 - (I_2 \times R_5) \]

\[ 13 = 8.4 / 4.2 \times (3.1 - 2.1) + 14 - (1 \times 2.3) \]

The expression is real. \( A_I \) is integer; the result of the expression evaluation, a real, is converted to an integer replacing \( A_I \).
### Table 4-1. Replacement Statement A = exp

- **A** is an Identifier
- **exp** is an Expression
- **¢(f)** is the Evaluated Expression

<table>
<thead>
<tr>
<th>Mode of W(f) of A</th>
<th>Complex</th>
<th>Double</th>
<th>Real</th>
<th>Integer</th>
<th>Logical</th>
<th>Masking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Store real and imaginary parts of W(f) in real and imaginary parts of A.</td>
<td>Round W(f) to real. Store in real part of A. Store zero in imaginary part of A.</td>
<td>Store W(f) in real part of A. Store zero in imaginary part of A.</td>
<td>Convert W(f) to real and store in real part of A. Store zero in imaginary part of A.</td>
<td>Store W(f) in real part of A. Store zero in imaginary part of A.</td>
<td>Illegal</td>
<td></td>
</tr>
<tr>
<td>Complex</td>
<td>Store real part of W(f) in A and replace the least significant part with plus zero.</td>
<td>Store W(f) (most and least significant parts) in A. Store in A (most and least significant parts).</td>
<td>If W(f) is ±, affix ±0 as least significant part. Store in A (most and least significant parts).</td>
<td>Convert W(f) to real. Fill least significant part with plus zeros. Store in A (most and least significant parts).</td>
<td>Illegal</td>
<td></td>
</tr>
<tr>
<td>Double</td>
<td>Store real part of W(f) in A. Imaginary part is lost.</td>
<td>Round W(f) to real and store in A. Least significant part of W(f) is lost.</td>
<td>Store W(f) in A.</td>
<td>Convert W(f) to real. Store in A.</td>
<td>Store W(f) in A.</td>
<td>Store W(f) in A.</td>
</tr>
<tr>
<td>Real</td>
<td>Truncate real part of W(f) to integer. Store in A. Imaginary part is lost.</td>
<td>Truncate W(f) to integer and store in A. The least significant part of W(f) is lost.</td>
<td>Truncate W(f) to integer. Store in A.</td>
<td>Store W(f) in A.</td>
<td>Store W(f) in A.</td>
<td>Store W(f) in A.</td>
</tr>
<tr>
<td>Integer</td>
<td>Store real part of W(f) in A. Imaginary part is lost.</td>
<td>Store most significant part of W(f) in A. Least significant part is lost.</td>
<td>Store W(f) in A.</td>
<td>Store W(f) in A.</td>
<td>Store W(f) in A.</td>
<td>Store W(f) in A.</td>
</tr>
<tr>
<td>Logical</td>
<td>Store real part of W(f) in A. Imaginary part is lost.</td>
<td>Store most significant part of W(f) in A. Least significant part is lost.</td>
<td>Store W(f) in A.</td>
<td>Store W(f) in A.</td>
<td>Store W(f) in A.</td>
<td>Store W(f) in A.</td>
</tr>
</tbody>
</table>

† Hollerith constants and octal constants are assumed to be type integer in an expression.
Example 5

\[ A_D = D_1 \times (D_2 + D_3) + D_2 \]

4.96800000000000D+01 = 2.0D**'2* (3.2D+(4.1D*1.0D))
+ (3.2D**2.0D*3.2D)

The expression is double precision. \( A_D \) is double precision; the result of the expression evaluation, a double-precision floating quantity, replaces \( A_D \).

Example 6

\[ A_I = C_I \times R_1 - R_2 + I_1 \]

33 = (4.4,2.1)**8.4 - 4.2 + 0.4

The expression is complex. Since \( A_I \) is integer, the truncated real part of the evaluated expression replaces \( A_I \).

**Multiple Replacement Statements**

The multiple replacement statement is a generalization of the replacement statements discussed earlier in this chapter, and its form is

\[ \psi_n = \psi_{n-1} = \ldots = \psi_2 = \psi_1 = \text{expression} \]

For example, \( A = B = C = 4.4\times\cos(x) + 32 \). The \( \psi_i \) are variables of any type, and the multiple replacement statement replaces each of the variables \( \psi_1, \ldots, \psi_n \) with the value of the expression following the last equals. This is done with the standard conventions used in mixed-mode arithmetic statements, as shown in the following examples, where each successive replacement \( \psi_i = \psi_{i-1} \) is done based only on the modes of \( \psi_i \) and \( \psi_{i-1} \) (see example 5 below).
Given that $A$ is real; $C_1,C_2$ is complex; $D$ is double; and $I$ is integer:

**Example 1**

\[
A = D = 3.1415926535897932384626D + D = 3.1415926535897932384626D + 3.141592654 + A
\]

Note: In octal representation, $A$ is not rounded; in this case, conversion of $A$ from octal to decimal causes $A$ to appear as if it had been rounded.

**Example 2**

\[
I = A = 4.6
\]

\[
4.6 \to A
\]

\[
4. \to I
\]

**Example 3**

\[
A = I = 4.6
\]

\[
4 \to I
\]

\[
4.0 \to A
\]

**Example 4**

\[
I = A = C_1 = (10.2, 3.0)
\]

\[
10.2 \to C_1 \text{ real}
\]

\[
3.0 \to C_1 \text{ imaginary}
\]

\[
10.2 \to A
\]

\[
10 \to I
\]

**Example 5**

\[
C_2 = A = I = C_1 = (13.4, 16.2)
\]

\[
13.4 \to C_1 \text{ real}
\]

\[
16.2 \to C_1 \text{ imaginary}
\]

\[
13 \to I
\]

\[
13.0 \to A
\]

\[
13.0 \to C_2 \text{ real}
\]

\[
0.0 \to C_2 \text{ imaginary}
\]

Only one expression is permissible and it must be the last replacement in the set of replacements. If an expression is introduced in the middle of a string, the diagnostic ILLEGAL USE OF A REPLACEMENT STATEMENT will be printed.
TYPE DECLARATIONS AND STORAGE ALLOCATION

COMPLEX nlist
DOUBLE PRECISION nlist
REAL nlist
INTEGER nlist
LOGICAL nlist

IMPLICIT typ(let[,let]...)[,typ(let[,let]...)]...
PARAMETER (param=exp[,param=exp]...)
DIMENSION v(n1 [,n2]][,n3)][v(n1) [,n2]][,n3)]...
COMMON/blk1/nlist[,blk2/nlist...]
EQUIVALENCE(nlist[,nlist])...
DATA nlist1/clist1/[,nlist2/clist2/[,nlistn/clistn/]


The FORTRAN code discussed in this chapter determines the types of variables that are used and where they will be stored in the program. Since this information must be established before the program is executed, the TYPE and storage allocation instructions are nonexecutable and are handled at compile time.

**STATEMENT ORDERING**

TYPE, PARAMETER, IMPLICIT, DIMENSION, COMMON and EQUIVALENCE are specification statements. They must appear ahead of the first executable statement in the program; otherwise, the diagnostic is DECLARATIVE STATEMENTS MUST PRECEDE ALL ARITHMETIC STATEMENTS. The IMPLICIT statement must appear before any other specification statement, except PARAMETER. The PARAMETER statement must precede any specification statement that contains a parameter reference. DATA declarations may appear anywhere in a program unit following the specification statements.
TYPE DECLARATION

The TYPE declaration statement provides the compiler with information on the internal representation and names of variables and functions. Both the name and the datum identified by the name have a type. There are five variable types declared by one of the following statements:

<table>
<thead>
<tr>
<th>Statement</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPLEX nlist</td>
<td>2 words/element; floating point</td>
</tr>
<tr>
<td>DOUBLE PRECISION nlist</td>
<td>2 words/element; floating point</td>
</tr>
<tr>
<td>REAL nlist</td>
<td>1 word/element; floating point</td>
</tr>
<tr>
<td>INTEGER nlist</td>
<td>1 word/element; integer</td>
</tr>
<tr>
<td>LOGICAL nlist</td>
<td>1 word/element; integer</td>
</tr>
</tbody>
</table>

nlist is a string of identifiers separated by commas; positive integer constant subscripts are permitted. An example of nlist is:

A, B1, CAT, D36F, GAR(1,2,3)

Rules

- The TYPE declaration is nonexecutable and must precede the first executable statement.

- A TYPE declaration for type double or complex turns off the optimizer for expressions in the entire routine in which the type declaration appears.

- If an identifier is declared in two or more TYPE declarations, the diagnostic ATTEMPT TO DOUBLY TYPE A VARIABLE will appear.
• An identifier not declared in a TYPE declaration or in IMPLICIT typing has a default type of real or integer. Type integer is assumed if the first letter of the name is I, J, K, L, M or N; for any other letter, it is type real.

• The TYPE statement may also provide dimension information. When subscripts appear in nlist, the associated identifier is the name of an array, and the product of the subscripts determines the amount of storage to be reserved for that array. Where the amount of storage is specified in more than one declarative statement, a diagnostic appears.

• Type logical is essentially type integer. However, any expression containing a logical variable may ignore the mixed-mode conventions. Results from the expression are unpredictable; for example, L + X, where L is logical and X is real, generates a simple unnormalized add of the two variables, resulting in meaningless code.

Examples

COMPLEX ALPHA,B21,INTERIM
DOUBLE PRECISION POST(10,10),NO,BILL
REAL K(2,3),RE,IN2F
INTEGER X,Y,GAR(1,2,3)
DOUBLE PRECISION RL,MASS(10,10)

For a discussion of typing a function name, see Function Definition in Chapter 7.
Implicit typing is used to identify a type associated with single letters that appear as the first letter of names of variables, constants, arrays or functions. It is used primarily to modify the typing conventions that are used in FORTRAN as default types which apply when no explicit typing appears among the specification statements of a program unit. Explicit typing overrides both the default type and the implicit type of a name.

The form of the IMPLICIT statement is:

```
IMPLICIT typ(let1[,let2]...)[,typ(let3[,let4]...)]...
```

where `typ` may be `COMPLEX`, `DOUBLE PRECISION`, `REAL`, `INTEGER` or `LOGICAL`; and `let` may be a list of single letters, `a1[,a2]...`, or a range of letters, `a1-a2`, or a combination of single letters and ranges of letters.

**Examples**

```
IMPLICIT REAL (I,N)
IMPLICIT INTEGER (A,C,X)
IMPLICIT COMPLEX (C,V-Z)
IMPLICIT REAL (I),INTEGER (A)
IMPLICIT LOGICAL (A-C),INTEGER (D-F,R-T,Z)
IMPLICIT INTEGER (A-J),REAL (K-Z)
```
5.5
Type Declarations and
Storage Allocation

Rules

- Any letter appearing as a single letter or within a range of letters in an IMPLICIT statement determines the type of those symbolic names beginning with that letter.

- The symbolic names affected may be names of constants, variables, arrays or external functions in the program unit.

- The type applies to all variables that are used within a main program, or within a particular subprogram. The implicit typing does not apply outside the range of a program unit.

- Implicit typing may not be used to modify the type of in-line functions.

- Implicit typing may not be used to specify a type for processor-supplied external routines such as SIN, TAN, etc.

- Explicit type statements take precedence over any implicit typing specified in a program unit.

- IMPLICIT statements must appear before any other specification statements except the PARAMETER statement.

- A particular letter may be used only once in any IMPLICIT statement or statements appearing in a given program unit.

- IMPLICIT statements refer to letters and may not contain explicit symbolic names of more than one letter. Therefore no dimension information may be supplied.
Example 1

```fortran
*FORTRANjFL
CARD APPROXIMATE
NUMBER PROGRAM LOCATION
1 0 PROGRAM TEST
2 0 IMPLICIT INTEGER (A,C), REAL (I-K), DOUBLE (X-Z)
3 0 IMPLICIT COMPLEX (E-G)
4 0 ABC = 2.3
5 0 IVAR = 1.5
6 0 JVAR = 3.6
7 0 CB = JVAR
8 0 XE = 10*252525205
9 0 ZA = -XE
10 0 EBAR = (3.2E2,-6.666)
11 WRITE (6,102)
12 23 102 FORMAT(1HO5X"ABC=",&I3/)
13 23 WRITE (6,104) IVAR,JVAR
14 34 104 FORMAT(1HO5X,"IVAR=",&F4.1 5X,JVAR=",&F4.1/)
15 34 WRITE (6,105) CB
16 43 105 FORMAT(1HO5X,"CB=",&I3/)
17 43 WRITE (6,106) ZA
18 52 106 FORMAT(1HO5X,"ZA=",&D16.8/)
19 END
```

LENGTH OF ROUTINE TEST 111

VARIABLE ASSIGNMENTS
ABC = 2
IVAR = 1.5
JVAR = 3.6
CB = 3
ZA = -1.052525250+06

SUBROUTINES CALLED
OUTPTC EXIT

COMPILE TIME = 21 MILLISECS

PROGRAM SPACE IS 1662

ORIGIN ENTRY POINTS AND LOCATIONS
4 TEST
115 GEGERR 115
130 EXIT 133 END 143 STOP 133
150 KODER 150
1203 OUTPTC 1203 OPTPER 1636

OVERALL CORE USE STATISTICS (DECIMAL)
THE MAXIMUM SCM IS 54077
THE PROGRAM CURRENTLY USES 946
THE LOADING PROCESS USED 8635
THE MAXIMUM LCM IS 449602
THE PROGRAM CURRENTLY USES 4785

LCM AREA MAP BY BUFFER TYPE
MISC 185 SYSIDU 6 PR 2049 RANFO 64 SYSSAT 512
SYSDIR 1024 SYSSCM 946

ABC = 2
IVAR = 1.5
JVAR = 3.6
CB = 3
ZA = -1.052525250+06

TERMINATION DATE = 06/27/77
TERMINATION TIME = 12/15/19
TOTAL CPU TIME IN MILLISECONDS = 89
PPU TIME IN MILLISECONDS = 400
PAGES PRINTED = 3
TOTAL RESOURCES USED = 14

DISK BLOCK USAGE SUMMARY
ODD UNITS EVEN UNITS
DISK 0 DISK 1
LIMIT ON BLOCKS ALLOWED 4095 4095
MAXIMUM BLOCKS USED 3 0
PLIB BLOCKS FOR THIS PROJECT 1
The PARAMETER statement is used to give a symbolic name to a constant. The name is used subsequently in specifications or in executable statements within a program unit. Thus, the symbolic name of the constant replaces the constant itself in FORTRAN statements. A useful application might be in a program unit containing many references to variables in a DIMENSION statement having the same dimension specification. Changing the value of the constant is simplified using the symbolic name of a constant and fewer programming errors occur at the time of the program change.

The form of the PARAMETER statement is

```
PARAMETER (param_1=exp_1[,param_2=exp_2]...)
```

where param is the symbolic name of a constant; and exp is an expression containing constants and/or symbolic names of constants previously defined.

**Examples**

- `PARAMETER (PI=3.1412)`
- `PARAMETER (X=4.265E-9)`
- `PARAMETER (Y=PI^2.,I=300)`
- `PARAMETER (K=I+100,Z=X*Y)`
Rules

- exp must be a constant expression.

- Type of constants is determined by the name according to the same rules that govern the type of symbolic names of variables, arrays and functions.

- The value is defined by the constant expression, exp. Operands within exp may be constants or symbolic names of constants.

- The value of the constant replaces the symbolic name. (String substitution in a statement is not used.)

- The symbolic name of a constant may not be used subsequently as a variable, an array or a function.

- A parameter must be defined before it is used. The definition may occur in the same or a preceding parameter statement within that program unit.

- A parameter may be defined only once in a PARAMETER statement within a program unit.

- Parameters, that is, symbolic names of constants, may not appear in a FORMAT statement.

- Parameters may not have their type changed in type statements following the PARAMETER statement.
Example 2

```fortran
!FORTRANiFL
CARD APPROXIMATE
NUMBER PROGRAM LOCATION
1 0 PROGRAM TEST
2 0 PARAMETER (A=5,2,JK=2)
3 0 PARAMETER (P=A+JK+3,JP=5)
4 0 DIMENSION L(IP,JP)
5 0 II = IP
6 31 JJ = JP
7 31 WRITE (6,102) II, JJ
8 44 102 FORMAT(1X,13,5X,II=13,5X,II=13/)
9 44 END
```

LENGTH OF ROUTINE TEST 53
VARIABLE ASSIGNMENTS
L - 0 II - 46 JJ - 45
SUBROUTINES CALLED
OUTPTC EXIT
COMPILE TIME = 7 MILLISECS
PROGRAM SPACE IS 1604
ORIGIN ENTRY POINTS AND LOCATIONS
6 TEST 35
77 ORDER 57
72 EXIT 105 END 105 STOP 75
112 KODER 112
1145 OUTPTC 1145 DIPTER 1600
OVERALL CORE USE STATISTICS (DECIMAL)
The maximum SCM is 4077
The program currently uses 916
The loading process used 8805
The maximum LCM is 46965
The program currently uses 4755
LCM AREA MAP BY BUFFER TYPE
 MISC 185 SYSTOU 6 PN 2048 RANFD 64 SYSSAT 512
STOUT 1024 SYSC# 916
II = 10 JJ = 5
TERMINATION DATE = 06/27/77
TERMINATION TIME = 13/15/29
TOTAL CPU TIME IN MILLISECONDS = 204
PPU TIME IN MILLISECONDS = 324
PAGES PRINTED = 2
TOTAL RESOURCES USED = .15

DISK BLOCK USAGE SUMMARY
ODD UNITS EVEN UNITS
DISK 0 DISK 1
LIMIT ON BLOCKS ALLOWED 4095 4095
MAXIMUM BLOCKS USED 3 0
PLIB BLOCKS FOR THIS PROJECT 1
A subscripted variable represents an element of an array. For example, A(8) in an expression refers to the eighth element of the array A. Storage is reserved for arrays in the nonexecutable statements DIMENSION, COMMON, or TYPE. DIMENSION A(8) means save eight elements to be referenced by the array name A.

```
DIMENSION v1(n1[,n2][,n3])[v2(n1[,n2][,n3])...]
```

The variable names \(v_1\) may have one, two, or three integer constant subscripts separated by commas, as in \(X(2,50,10)\). If more than three subscripts appear after an array name in a DIMENSION statement, the compiler diagnostic is GREATER THAN THREE DIMENSIONS IN DIMENSION STATEMENT. The DIMENSION declaration is nonexecutable and must precede the first executable statement.

The number of computer words reserved for an array is determined by the product of the subscripts in the subscript string and the type of the variable.

\(ARRA(2,4,3)\) would have \(2 \times 4 \times 3\) elements, or 24.

A maximum of 65,535 elements may be reserved in any one array. If the maximum is exceeded, the diagnostic is DIMENSIONED ARRAY TOO LARGE FOR CORE. Practically speaking, room for instructions and system requirements is needed for any program. In the declarations

```
COMPLEX ATOM
DIMENSION ATOM(10,20),
```

the number of elements in the array ATOM is 200. Two words are used to store a complex element; therefore, the number of computer words reserved is 400. This is also true for double-precision arrays. The maximum array size for arrays with
two-word elements is 32,767. For real and integer arrays, the number of words in an array equals the number of elements in the array.

If an array is dimensioned in more than one declaration statement, the diagnostic is AN ARRAY NAME PREVIOUSLY USED.

Examples

```
DIMENSION A(20,2,5)
DIMENSION MATRIX(10,10,10),VECTOR(100),ARRAY(16,27)
```

No diagnostic is provided during execution for index values that exceed the DIMENSION specification but stay within the program field length, for example:

```
DIMENSION A(10),B(15)
A(14) = 42.
B(02) = 91.
```

In the above example, no index check is made. A(14) is actually B(4). The replacement statement is executed and B(4) is destroyed. In order to add a check on dimensions, many more instructions would need to be added to a compiled program; the check instructions would be required every time an index is calculated. This is too costly in terms of run time and overall efficiency of the program. (When needed for program debugging, an index-checking facility can be activated through the use of a program preprocessor such as NCAR's FRED).
Negative subscripts may be used to access dimensioned variables during program execution. For example, in the statement \texttt{DIMENSION A(10),B(15)}, the following correspondences apply. Element \(B(-2)\) is the same as element \(A(8)\). The 0th element precedes \(B(1)\), \(B(-1)\) precedes \(B(0)\), etc.

If a subscript exceeds the program field in either direction, the execution diagnostic \texttt{SCM DIRECT RANGE} will be printed on the Control Data 7600.

A general formula for calculating a single element of an array with more than one subscript is included under "Array Structure" in chapter 2.
When an array identifier and its dimensions appear as formal parameters in a function or subroutine, the dimensions may be assigned through the actual parameter list accompanying the function reference or subroutine call. Notice that the array name, as well as the DIMENSION statement identifiers M and N, must appear in the parameter list. The dimensions must not exceed the maximum array size specified by the DIMENSION declaration in the calling program. However, no compiler diagnostic is provided and storage will be overwritten in the array area during execution. (See Variable Dimensions in Subprograms in Chapter 7.) For example:

```
SUBROUTINE VAR(A,M,N)
  DIMENSION A(M,N)
```

If either A, M, or N does not appear in the formal parameter list or if they appear in COMMON, the compiler diagnostic is VARIABLE DIMENSIONED IDENTIFIER NOT IN FORMAL PARAMETER LIST.

Variable dimensions may not be used in a main program; the diagnostic will be VARIABLE DIMENSIONED IDENTIFIER NOT IN FORMAL PARAMETER LIST.
When a programmer uses variable names in a program or declares arrays in a DIMENSION statement, the compiler provides storage within the program or subprogram for these variables and arrays. A place is saved for them within the program and the name of the variable or array is sometimes called a "place holder." Since these variables are located only within the program or subprogram using them, they are often called "local variables."

**Example**

System Reference to Start of User's Program:

<table>
<thead>
<tr>
<th>0</th>
<th>Monitor Communications Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>A (array)</td>
</tr>
<tr>
<td></td>
<td>Main</td>
</tr>
<tr>
<td></td>
<td>Subprogram</td>
</tr>
<tr>
<td></td>
<td>Library Programs</td>
</tr>
</tbody>
</table>

In this example, the array A and the variables B and C are local variables. They are stored only within the program or subprogram referring to them.
The COMMON declaration may establish blocks of storage for variables and arrays that are available generally to the main program, to various subprograms, or to both. A particular common block area is available to any program unit which declares this block in a declaration statement and only to those program units with this declaration.
The program shown in Example 3 is loaded in the following locations:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Monitor Communications Area</td>
</tr>
<tr>
<td>48</td>
<td>Program TEST</td>
</tr>
<tr>
<td>128</td>
<td>T1 Common</td>
</tr>
<tr>
<td>14</td>
<td>Blank Common</td>
</tr>
<tr>
<td>17</td>
<td>Subprogram TT</td>
</tr>
<tr>
<td>26</td>
<td>T2 Common</td>
</tr>
<tr>
<td>30</td>
<td>.41 Common</td>
</tr>
<tr>
<td>32</td>
<td>BLKA Common</td>
</tr>
</tbody>
</table>

The Variable Assignments table specifies a relative address for each variable. If the variable is in common, Cxx appears after the address. (xx is merely a count of the blocks in any one routine. This count does not necessarily agree from program to subprogram.) D and E are in common T1; A, B, and C are in blank common; X and Y are local variables where no common block is specified. Common blocks are loaded as they appear in the program; T1 is first, then blank common. Thus T1 is C 0, blank common is C 1. Notice that both of these common blocks are loaded following the program, which is loaded into 48. The subroutine has common blocks not declared in the main program, T2, .41, BLKA. These are loaded following the subprogram TT which first declares them to be common blocks. The order of common blocks in the subroutine is

C 0 blank common
C 1 T2
C 2 .41
C 3 BLKA
Example 3

```fortran
PROGRAM TEST
  COMMON /T/ A, B
  COMMON C
  X = 4.0
  Y = 5.0
END
```

```
LENGTH OF ROUTINE TEST
VARIABLE ASSIGNMENTS
A - 0
B - 1
C - 2
X - 5
Y - 4

SUBROUTINES CALLED
EXIT

COMMON BLOCKS AND LENGTHS
T1 - 2 - 3

COMPILE TIME = 5 MILLISECS
```

```fortran
SUBROUTINE TT
  COMMON A, B, C
  COMMON /T2/ F, G
  COMMON /T1/ H, I
  COMMON /BLKA/ U, V
END
```

```
LENGTH OF ROUTINE TT
VARIABLE ASSIGNMENTS
A - 0
B - 1
C - 2
F - 1
G - 1
H - 1
I - 1
U - 1
V - 1

COMMON BLOCKS AND LENGTHS
T1 - 2 - 3
T2 - 2
41 - 2
BLKA - 2

COMPILE TIME = 3 MILLISECS
```

PROGRAM SPACE IS 54

ORIGIN ENTRY POINTS AND LOCATIONS
4 TEST
17 TT
34 FAIT
47 END
47 STOP

COMMON BLOCKS LOCATION
T1 - 12
T2 - 14
T1 - 15
T2 - 16

OVERALL CORE USE STATISTICS (DECIMAL)
THE MAXIMUM SCM IS 54101
THE PROGRAM CURRENTLY USES 60
THE LOADING PROCESS USED 7949
THE MAXIMUM LCM IS 449602
THE PROGRAM CURRENTLY USES 3899

LCM AREA MAP BY BUFFER TYPE
MISC 185 SYSDU 6 PR 2048 RANFO 64 SYSSAT 512
SYSGM 1024 SYSCM 60

TERMINATION DATE = 06/21/77
TERMINATION TIME = 11/36/24

TOTAL CPU TIME IN MILLISECONDS = 82
PPU TIME IN MILLISECONDS = 209
PAGES PRINTED = 3
TOTAL RESOURCES USED = 11

DISK BLOCK USAGE SUMMARY
ODD UNITS
DISK 0
LIMIT ON BLOCKS ALLOWED 4095
MAXIMUM BLOCKS USED 0
PLIB BLOCKS FOR THIS PROJECT 1

EVEN UNITS
DISK 1

The program shown in Example 4 is loaded as follows:

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Monitor Communications Area</td>
</tr>
<tr>
<td>48</td>
<td>Blank Common</td>
</tr>
<tr>
<td>78</td>
<td>Program TEST</td>
</tr>
<tr>
<td>158</td>
<td>T1 Common</td>
</tr>
<tr>
<td>178</td>
<td>Subprogram TT</td>
</tr>
<tr>
<td>268</td>
<td>T2 Common</td>
</tr>
<tr>
<td>308</td>
<td>.41 Common</td>
</tr>
<tr>
<td>328</td>
<td>BLKA Common</td>
</tr>
<tr>
<td>FL</td>
<td></td>
</tr>
</tbody>
</table>

When the reference to blank common appears in the first program loaded, and it is the first common block specified, then blank common is loaded into 48. (For programs that are approaching the maximum core size, some saving of core is made at load time by having blank common at 48. The loader, which is about 10,000\_h, will then use the same space as the blank common storage area; after this, it is set to zero.) Blank common must appear as the first common block or it will follow the program, as in Example 3.

Areas of common information may be specified by the declaration:

```plaintext
COMMON nlist
COMMON /blk1/nlist[/blk2/nlist ...]
```
Example 4

```fortran
  PROGRAM TEST
  COMMON A,B
  COMMON C
  COMMON/T1/ D,E
  X=4.0
  Y=5.0
  END
  
  LENGTH OF ROUTINE TEST
  VARIABLE ASSIGNMENTS
  A - O C 0 B - 1 C O C 2 C 0 D - OC 1 E - 1 C 1 X - 5
  Y - 4
  SUBROUTINES CALLED
  EXIT
  COMMON BLOCKS AND LENGTHS
  - 3 T1 - 2
  COMPILe TIME = 5 MILLISECS

  PROGRAM TEST
  COMMON A,B,C
  COMMON /T2/ F,G
  COMMON /41/ H,I
  COMMON /BLKA/ U,V
  END
  
  LENGTH OF ROUTINE TT
  VARIABLE ASSIGNMENTS
  A - O C 0 B - 1 C 0 C 2 C 0 F - OC 1 G - 1 C 1 H - OC 2
  I - 1 C 2 U - OC 3 V - 1 C 3
  COMMON BLOCKS AND LENGTHS
  - 3 T2 - 2 *41 - 2 BLKA - 2
  COMPILe TIME = 3 MILLISECS
  
  PROGRAM SPACE IS 54
  ORIGIN ENTRY POINTS AND LOCATIONS
  7 TEST 7
  17 TT 17
  34 EXIT 47 END 37
  STOP
  
  COMMON BLOCKS LOCATION
  4 T1 15 T2 26 *41 30 BLKA 32
  
  OVERALL CORE USE STATISTICS (DECIMAL)
  THE MAXIMUM SCM IS 54101
  THE PROGRAM CURRENTLY USES 60
  THE LOADING PROCESS USED 7965
  THE MAXIMUM LCM IS 449602
  THE PROGRAM CURRENTLY USES 3899
  LCM AREA MAP BY BUFFER TYPE
  MISC 765 SYSSIU 6 PR 2048 RANFO 64 SYSSAT 512
  SYSDM1 1024 SYSSCM 60
  
  TERMINATION DATE = 06/21/77
  TERMINATION TIME = 11/37/02

  TOTAL CPU TIME IN MILLISECONDS = 49
  PPU TIME IN MILLISECONDS = 248
  PAGES PRINTED = 3
  TOTAL RESOURCES USED = 410

  DISK BLOCK USAGE SUMMARY
  ODD UNITS EVEN UNITS
  DISK 3 DISK 1
  LIMIT ON BLOCKS ALLOWED 6795 4795
  MAXIMUM BLOCKS USED 9 3
  PLIB BLOCKS FOR THIS PROJECT 1
```


COMMON
VARIABLES
(continued)

blk_i is a common block identifier up to six characters long; it is the common block name. An alphanumeric identifier must start with a letter. A numeric identifier starting with a number must contain only numbers. Leading zeros in identifiers are ignored. Zero by itself is an acceptable common block identifier. The following are COMMON identifiers:

/1Z13/ 1/
/1MAXMUS/ 146/
/1Z/ 6600/
/1XRAY/ 0/

nlist is a string of identifiers representing simple and subscripted variables. If a nonsubscripted array name appears in nlist, the dimensions must be defined by a TYPE or DIMENSION declaration in that program. If an array is dimensioned in more than one declaration, a compiler diagnostic is given.

The common block identifier is omitted for blank common.
Common || nlist and common nlist both denote blank common.

Examples

COMMON A,B,C
COMMON /BLOCKA/A(15),B,C/123/DL(5,2),ECHO,X
COMMON B,A /VECTOR/VECTOR(15),VECTORA,VECTORB

Rules

• COMMON is nonexecutable and must precede the first executable statement. Any number of COMMON declarations may appear in a program.

• A COMMON declaration with a given name specified, /NAME/, may appear only once in any given program or subprogram. If one card is not long enough to hold all variable names in the list, continuation cards may be used.

    COMMON/Al/A,B,C,...,
    1E,Q,R(10,100)
• If a repeated name such as COMMON/A1/E,Q,R(10,100) is used instead of a continuation card, the diagnostic will be DUPLICATE BLOCK NAME.

• Blank common is the only declaration that may be used more than once in the same program and it must appear consecutively, such as

  COMMON A,B
  COMMON C,D,E

• If DIMENSION or COMMON or TYPE declarations appear together, the order is immaterial.

• Common block identifiers are used only for block identification within the compiler; they may be used elsewhere in the program as other kinds of identifiers except as subroutine and program names in the same job.

• An identifier in one common block may not appear as an identifier in another common block. (If it does, the name is doubly defined.)

• The order of array storage within a common block is determined by the list declaration.

• At the beginning of program execution the contents of all common areas are zero except those areas in named COMMON which were specified in a DATA declaration. (This can be changed by using *RUN,I; then at the beginning of program execution the contents of all common areas are negative indefinite (see chapter 2) except those areas in named COMMON which were specified in a DATA declaration.)

• Blank common may not be entered with data using a DATA statement. A diagnostic will be given that blank common may not be preset.
Rules (continued)

- The length of a common block in computer words is determined from the number and type of the list variables. In the following statements, the length of common block A is 12 computer words. The origin of the common block is Q(1).

```
COMMON/A/Q(4), R(4),S(2)
REAL Q,R
COMPLEX S
```

Block A

<table>
<thead>
<tr>
<th>Origin</th>
<th>Q(1)</th>
<th>Q(2)</th>
<th>Q(3)</th>
<th>Q(4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q(1)</td>
<td>Real part</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q(2)</td>
<td>Imaginary part</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q(3)</td>
<td>Real part</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q(4)</td>
<td>Imaginary part</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- If a subprogram does not use all of the locations reserved in a common block, unused variables may be necessary in the COMMON declaration to insure proper correspondence of common areas.

```
Main program          COMMON/SUM/A,B,C,D
Subprogram            COMMON/SUM/DUMMY(2),C
```

In the above example, only the variable C is used in the subprogram. The unused variable DUMMY is necessary to space over the area reserved by A and B.

- The first appearance of a common block establishes its length. If a longer block with the same name appears in a subprogram later, the diagnostic ATTEMPT TO CHANGE LENGTH OF COMMON BLOCK FROM XXXXXX TO XXXXXX is given.
The length of a common block must not be extended by sub-
programs using the block. However, the length may be
* shortened by subprograms. The symbolic names used within
the block may differ as shown below.

Each subprogram using a common block assigns the allocation
of words in the block. The identifiers used within the
block may differ as to name, type, and number of elements,
although the block identifier itself must remain the same.

The loader will overlay any blank common when it appears
* first in a program, regardless of the size of the common
length declared (larger or smaller). Therefore, when core
is short, as in a large program, and every location is
needed, blank common must appear first so that the loader
can overlay it. If blank common does not appear first,
the program may not load.

Example 5

```plaintext
PROGRAM MAIN
  COMPLEX C
  COMMON/TEST/C(20)/36/A,B,Z
  :
  END'
```

The length of TEST is 40 computer words. The subprogram may
rearrange the allocation of words as in:

```plaintext
SUBROUTINE ONE
  COMMON/TEST/A(10),G(10),K(10)
  COMPLEX A
  :
  END
```

The length of this TEST is also 40 words. The first 10
elements (20 words) of the block represented by A are complex
elements. Array G is the next 10 words and array K is the
last 10 words. Within the subprogram, elements of G are
treated as real floating point quantities and elements of K
are treated as integer quantities.
The EQUIVALENCE declaration permits variables to share the same storage locations. It is not a replacement statement and thus does not equate variables mathematically. The equivalence variables share the memory location(s) of the leftmost variable identifier. The variable on the left is called a referenced identifier in this program. Any name in the equivalence list calls out this memory location; thus, more than one identifier for a cell is established. The general form is EQUIVALENCE (nlist[, (nlist)]...)

EQUIVALENCE (A,B,C,...),(X(1),Y(4),Z(2)...)  

(A,B,C,...) is an EQUIVALENCE group of two or more simple or single-subscripted variable names. The equivalence group is defined as C shares the location of B, B shares locations of A; thus, C shares locations of A. A is the referenced identifier and appears on the left. The equivalence group is order-dependent. If one or more variables in the group are in COMMON, one of the COMMON variables must be the referenced identifier.

A multiply-subscripted variable can be represented only by a singly-subscripted variable. The correspondence is:

A(i,j,k) is the same as A((i+(j-1)*I + (k-1)*I*J))

where i,j,k are integer constants; I and J are the integer constants appearing in DIMENSION A(I,J,K). For example, in DIMENSION A(2,3,4), the element A(1,1,2) is represented by A(7).
EQUIVALENCE is most commonly used when two or more arrays can share the same storage locations. The lengths may be different or equal.

**Example 6**

```plaintext
DIMENSION A(10,10), I(100)
EQUIVALENCE (A, I)
8 READ 10, A
... 
9 READ 20, I
... 
END
```

The EQUIVALENCE declaration assigns the first element of array A and array I to the same storage location. The READ statement 8 stores the A array in consecutive locations. All operations using A should be completed before statement 9 is executed, since the values of array I are read into the storage locations previously occupied by A.
EQUIVALENCE DECLARATION
(continued) *

Example 7

**DECLARATION**

**CARD**

<table>
<thead>
<tr>
<th>NUMBER</th>
<th>APPROXIMATE PROGRAM LOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 PROGRAM TEST</td>
</tr>
<tr>
<td>2</td>
<td>0 DIMENSION A(10), B(2,5)</td>
</tr>
<tr>
<td>3</td>
<td>0 EQUIVALENCE (A(5), B(5))</td>
</tr>
<tr>
<td>4</td>
<td>0 EQUIVALENCE (X, )</td>
</tr>
<tr>
<td>5</td>
<td>1 I=172000000000000000000000</td>
</tr>
<tr>
<td>6</td>
<td>15 WRITE (A(5)) X</td>
</tr>
<tr>
<td>7</td>
<td>26 FORMAT (I0**F5.1)</td>
</tr>
<tr>
<td>8</td>
<td>26 X(7)=99.</td>
</tr>
<tr>
<td>9</td>
<td>26 WRITE (A(10)) A(5)</td>
</tr>
<tr>
<td>10</td>
<td>37 FORMAT (A(5)=F5.2)</td>
</tr>
<tr>
<td>11</td>
<td>37 END</td>
</tr>
</tbody>
</table>

LENGTH OF ROUTINE TEST = 47

VARIABLE ASSIGNMENTS
A = 0 B = 2 X = 14 I = 14

SUBROUTINES CALLED
OUTPTC
EXIT

COMPILE TIME = 10 MILLISECS

PROGRAM SPACE IS 1577

ORIGIN ENTRY POINTS AND LOCATIONS
4 TEST 21
53 Q00ERR 55
65 EXIT 100 END 100 STOP 70
1140 OUTPTC 1140 OTPTER 1573

OVERALL CORE USE STATISTICS (DECIMAL)
The maximum SCM is 54101
The program currently uses 911
The loading process used 8600
The maximum LCM is 449602
The program currently uses 4750

LCM AREA MAP BY BUFFER TYPE
MISC 105 SYSGIU 6 PR 2048 RANFO 64 SYSSAT 512
SYSDM1 1024 SYSSCM 911

X = 1.0
A(5)=99.00

TERMINATION DATE = 06/21/77
TERMINATION TIME = 11/37/39

TOTAL CPU TIME IN MILLISECONDS = 61
PPU TIME IN MILLISECONDS = 191
PAGES PRINTED = 2
TOTAL RESOURCES USED = 8

DISK BLOCK USAGE SUMMARY

<table>
<thead>
<tr>
<th>ODD UNITS</th>
<th>EVEN UNITS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISK 0</td>
<td>DISK 1</td>
</tr>
<tr>
<td>LIMIT ON BLOCKS ALLOWED</td>
<td>4095</td>
</tr>
<tr>
<td>MAXIMUM BLOCKS USED</td>
<td>3</td>
</tr>
<tr>
<td>PLIB BLOCKS FOR THIS PROJECT</td>
<td>1</td>
</tr>
</tbody>
</table>
In example 7, the EQUIVALENCE declaration places A(5) in the same memory cell as B(3). Apply the formula above to attain a linear subscript for B. With dimensions at (2,5), B(1,2) is the same array element as B(3). Storage is arranged as in the following table:

<table>
<thead>
<tr>
<th>Origin A(1)</th>
<th>Double</th>
<th>Single</th>
</tr>
</thead>
<tbody>
<tr>
<td>A(2)</td>
<td>B(1,1) or B(1)</td>
<td></td>
</tr>
<tr>
<td>A(3)</td>
<td>B(2,1)</td>
<td>B(2)</td>
</tr>
<tr>
<td>A(4)</td>
<td>B(1,2)</td>
<td>B(3)</td>
</tr>
<tr>
<td>A(5)</td>
<td>B(2,2)</td>
<td>B(4)</td>
</tr>
<tr>
<td>A(6)</td>
<td>B(1,3)</td>
<td>B(5)</td>
</tr>
<tr>
<td>A(7)</td>
<td>B(2,3)</td>
<td>B(6)</td>
</tr>
<tr>
<td>A(8)</td>
<td>B(1,4)</td>
<td>B(7)</td>
</tr>
<tr>
<td>A(9)</td>
<td>B(2,4)</td>
<td>B(8)</td>
</tr>
<tr>
<td>A(10)</td>
<td>B(1,5)</td>
<td>B(9)</td>
</tr>
</tbody>
</table>

A and B share eight memory cells. In order to provide the correct equivalence specified, A requires two cells ahead of B for A(1) and A(2); B must be extended by two cells to include B(1,5) and B(2,5). The total block reserved for A and B together is 12 memory locations.

In the statement EQUIVALENCE (X,I) a floating point name is declared equivalent to an integer name. To avoid mixed mode when it is not applicable, choose the appropriate name. In Example 7, the octal constant is stored in I, but printed as a floating point number using X. The same address is used for both X and I.

Rules

- EQUIVALENCE is nonexecutable and must precede the first executable statement.
- DIMENSION, COMMON, EQUIVALENCE, DATA, or TYPE declarations may appear together in any order.
Rules
(continued)

- Any full or multiword variable may be made equivalent to any other full or multiword variable. The variables may be with or without subscripts.

- No diagnostic is provided if a variable with a subscript appears in an EQUIVALENCE statement if the variable has not been dimensioned. The equivalences in the storage block will be correct and include enough space for all variables that are implied in equivalence lists. EQUIVALENCE (A(3),B) where A is not dimensioned leaves two cells for A(1),A(2) ahead of B.

- The EQUIVALENCE declaration does not rearrange COMMON, but arrays may be defined as equivalent so that the length of the common block is changed. The origin of the common block must not be changed by the EQUIVALENCE declaration. The elements of COMMON arrays may be equivalenced to other COMMON variables in the same COMMON blocks or to the elements of arrays not in COMMON.

- No element of a formal parameter list may appear in an EQUIVALENCE statement within a subroutine. If it does, the compiler diagnostic is FORMAL PARAMETER ERROR IN EQUIVALENCE.

Storage Allocation

Storage is allocated differently depending on whether the storage array is in COMMON. The following simple examples illustrate changes in block lengths caused by the EQUIVALENCE declaration.
Example 8

No variables are in COMMON.

The origin of the storage block is A with five locations followed by B with two locations. C is the same as A+1. There are seven cells in the block. The arrangement is as follows in core:

<table>
<thead>
<tr>
<th>LOC origin</th>
<th>A(1)</th>
<th>A(2)</th>
<th>A(3) ↔ C(2)</th>
<th>A(3) ↔ C(2)</th>
<th>C(3)</th>
<th>C(4)</th>
<th>B(1)</th>
<th>B(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC+1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOC+2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOC+3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOC+4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOC+5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOC+6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Example 9

Two arrays in COMMON are equivalenced.

<table>
<thead>
<tr>
<th>CARD NUMBER</th>
<th>APPROXIMATE PROGRAM LOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PROGRAM TEST</td>
</tr>
<tr>
<td>2</td>
<td>COMMON A(3),B(2),C(4)</td>
</tr>
<tr>
<td>3</td>
<td>EQUIVALENCE (A(3),C(2))</td>
</tr>
<tr>
<td>4</td>
<td>END</td>
</tr>
</tbody>
</table>

```fortran
PROGRAM TEST
  COMMON A, B, C
  EQUIVALENCE (A(3), C(2))
END
```

The origin of the common block is A with three locations. B follows after A with two locations. C is defined as A+1. The total block length is five locations. The arrangement in core is:

- LOC origin: A(1)
- LOC+1: A(2), C(1)
- LOC+2: A(3), C(2)
- LOC+3: B(1), C(3)
- LOC+4: B(2), C(4)
Example 10

A is in COMMON, B is not in COMMON. Notice that A must appear as the referenced identifier. Sa and Sb are the subscript of A and B:

- $S_b \leq S_a$ is a permissible subscript arrangement
- $S_b > S_a$ is not permissible

Block 2 in COMMON is six cells long. A is the origin; B is now in COMMON; B(1) is equivalent to A(2). Locations are arranged as follows:

- LOC origin: A(1)
- LOC+1: A(2) <-> B(1)
- LOC+2: A(3) <-> B(2)
- LOC+3: A(4) <-> B(3)
- LOC+4: B(4)
- LOC+5: B(5)
B is in COMMON, A is not in COMMON. Notice that B must appear as the referenced identifier. $S_a$ and $S_b$ are the subscripts of A and B:

$S_a \leq S_b$ is a permissible subscript
$S_a > S_b$ is not permissible

The array in COMMON should appear first in the EQUIVALENCE statement. Common block 3 is six cells long with B as the origin; A(1) is the same cell as B(2). The common block was extended two cells because of the dimension of A.

```plaintext
LOC origin   B(1)
LOC+1        B(2) <-> A(1)
LOC+2        B(3)    A(2)
LOC+3        B(4)    A(3)
LOC+4        A(4)
LOC+5        A(5)
```
Example 12

A, B are both in Common. If A appears before B in the COMMON statement and Sa and Sb are the subscripts of A and B,

Sa ≥ Sb is permissible
Sa < Sb is not permissible
STORAGE ALLOCATION (continued)

The block is 8 (108) words long; the origin is at A; B(1) is the same as A(2), extending the block by three words.

<table>
<thead>
<tr>
<th>LOC</th>
<th>origin</th>
<th>A(1)</th>
<th>B(1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC+1</td>
<td>A(2)</td>
<td></td>
<td>B(2)</td>
</tr>
<tr>
<td>LOC+2</td>
<td>A(3)</td>
<td></td>
<td>B(3)</td>
</tr>
<tr>
<td>LOC+3</td>
<td>A(4)</td>
<td>←→</td>
<td>B(3)</td>
</tr>
<tr>
<td>LOC+4</td>
<td>A(5)</td>
<td></td>
<td>B(4)</td>
</tr>
<tr>
<td>LOC+5</td>
<td></td>
<td></td>
<td>B(5)</td>
</tr>
<tr>
<td>LOC+6</td>
<td></td>
<td></td>
<td>B(6)</td>
</tr>
<tr>
<td>LOC+7</td>
<td></td>
<td></td>
<td>B(7)</td>
</tr>
</tbody>
</table>

If B appears before A in the Common statement,

Sa ≤ Sb is permissible
Sa > Sb is not permissible

EQUIVALENCE (A(3), B(4)) would change the origin of the common block; thus, it is not allowed.
The DATA statement provides initial values for variables, arrays and array elements. The DATA statement is non-executable and may appear in a routine anywhere after the specification statements (DIMENSION, COMMON, EQUIVALENCE, TYPE, IMPLICIT, PARAMETER and EXTERNAL).

The form of the DATA statement is:

```
DATA klist1/clist1/[[,]klist2/clist2]/...
```

where \( klist_i \) is a list of names of variables, arrays and array elements. \( clist_i \) is a list of constants of the form \( c \) or \( r^c \), where \( c \) is a constant or the symbolic name of a constant and \( r \), the repetition count, is a non-zero positive integer constant or the symbolic name of such a constant. The form \( r^c \) is equivalent to \( r \) successive appearances of the constant \( c \).

The commas between successive \( klist/clist/ \) structures are optional.

When the program begins execution, each element of \( klist_i \) will have the value of the constant in the corresponding position of \( clist_i \).
5.36
Type Declarations and
Storage Allocation

**Rules**

- Dummy arguments (formal parameters), elements of blank COMMON and function names cannot be assigned initial values in a DATA statement. An entity may have an initial value assigned only once in a program.

- There must be a one-to-one correspondence between the elements of $klist_i$ and $clist_i$. If an array name without subscripts appears in $klist_i$, there must be one constant in $clist_i$ for each element of the array. The ordering of array elements is determined by the array element subscript value (see chapter 2).

- The type of each entry in $klist_i$ must be the same as the corresponding entry in $clist_i$ if either is type LOGICAL. Otherwise the $clist_i$ constant will be converted to the type of the corresponding element in $klist_i$ according to the rules for mixed mode replacement statements (see chapter 3).

- Each subscript in $klist$ must be an integer constant expression except for implied DO variables.
The implied DO loop in a DATA statement has the general form:

\[ \text{dlist, j=m1, m2[, m3]} \]

where dlist is a list of array element names and implied DO lists; \( j \) is an integer variable, the implied DO variable; and \( m_1, m_2 \) and \( m_3 \) are each an integer constant expression except that the expression may contain the implied DO variables of other implied DO loops that have this loop in their ranges.

The iteration count and values of the implied DO variables are established by \( m_1, m_2 \) and \( m_3 \) exactly as they are for the explicit DO loop (see chapter 6) except that the iteration count must be positive. If the third parameter, \( m_3 \) is omitted, a default value of 1 is used.
Rules
(continued)

Example

PARAMETER (KK=4,L=3,M=5)
DOUBLE PRECISION D
DIMENSION A(6), B(5,5), C(10), ID(M,M,M)
DATA (A(I),I=1,6)/1.,2.,KK*5./,(C(I),I=1,10)/5*1.5*2./
DATA ((B(J,I),I=1,J),J=1,5)/15*1/
DATA (((ID(I,J,K),I=1,M),J=1,M),K=1,M)/125*2./
DATA D/4.5/
DATA ICH/'ABCD'/

The above DATA statements cause initial values to be assigned so that when program execution begins, the variables and array elements will be defined as follows:

- Array elements A(1) and A(2) will contain the floating point values 1. and 2., respectively. Elements A(3) through A(6) will contain the floating point value 5.

- Array elements C(1) through C(5) will contain the floating point value 1.. Elements C(6) through C(10) will contain the floating point value 2..

- The diagonal and lower triangular elements of the array B will contain the floating point value 1..

- Each element of the three-dimensional array ID will contain an integer value of 2..

- The variable D will contain the double precision value 4.5D0.

- The variable ICH will contain the character string 'ABCD'.

Type Declarations and Storage Allocation Rules

Example

PARAMETER (KK=4,L=3,M=5)
DOUBLE PRECISION D
DIMENSION A(6), B(5,5), C(10), ID(M,M,M)
DATA (A(I),I=1,6)/1.,2.,KK*5./,(C(I),I=1,10)/5*1.5*2./
DATA ((B(J,I),I=1,J),J=1,5)/15*1/
DATA (((ID(I,J,K),I=1,M),J=1,M),K=1,M)/125*2./
DATA D/4.5/
DATA ICH/'ABCD'/

The above DATA statements cause initial values to be assigned so that when program execution begins, the variables and array elements will be defined as follows:

- Array elements A(1) and A(2) will contain the floating point values 1. and 2., respectively. Elements A(3) through A(6) will contain the floating point value 5.

- Array elements C(1) through C(5) will contain the floating point value 1.. Elements C(6) through C(10) will contain the floating point value 2..

- The diagonal and lower triangular elements of the array B will contain the floating point value 1..

- Each element of the three-dimensional array ID will contain an integer value of 2..

- The variable D will contain the double precision value 4.5D0.

- The variable ICH will contain the character string 'ABCD'.
CONTROL STATEMENTS

GO TO s
GO TO lab[(s_1[,s_2]...)]
ASSIGN s TO lab
GO TO (s_1[,s_2]...)[,,i]
IF (exp) s_1,s_2,s_3
IF (exp) s_1,s_2
IF (exp) st
IF (exp) THEN
ELSE IF (exp) THEN
ELSE
END IF
DO s[,i] = exp_1,exp_2[,exp_3]
CONTINUE
PAUSE [string]
END
CALL EXIT
STOP [string]
INTRODUCTION

Program execution normally proceeds from each statement to the statement immediately following it in the program. The normal execution sequence begins with the first executable statement of the main program. Control statements may be used to alter this sequence or to cause a number of iterations of the same program section.

Control may be transferred to an executable statement only; a transfer to a nonexecutable statement results in a program error.

Users of the 7600 may currently choose one of two DO-loop interpretations on the NCAR compiler. The default version of the compiler accepts a syntax which is closer to the FORTRAN 66 standard. However, the ",25" modifier on the *FORTRAN control card invokes a version of the compiler which accepts a DO-loop syntax which conforms to the FORTRAN 77 standard. A separate section of this chapter is devoted to a description of each syntax.
6.2
Control Statements

GO TO STATEMENTS

UNCONDITIONAL
GO TO STATEMENT

GO TO s

This statement causes an unconditional transfer to the statement labeled s.

- If there is no statement labeled s, the diagnostic THESE STATEMENT LABELS ARE MISSING will be printed followed by a list of the missing statement labels.

- If s is a variable instead of an integer constant, no diagnostic is provided. The compiler assumes it is an assigned GO TO.

ASSIGNED GO TO STATEMENT

GO TO lab[, (s_1[, s_2]...)]

This statement acts as a many-branch GO TO; lab is a simple integer variable assigned a label value s_i in a preceding ASSIGN statement. The s_i are statement labels. As shown, the parenthetical statement label list need not be present; however, if it is not, DO loop optimization will not take place where the GO TO is in a loop.

When the list is omitted, the comma after lab must be omitted. A simple integer variable, lab cannot be defined as the result of a computation. (For such a capability, see the computed GO TO statement). No compiler diagnostic is given if lab is computed or not defined, but the object code generated by the compiler is incorrect. If the address lab is not in the program, a termination with no comment may occur.
ASSIGN STATEMENT

ASSIGN s to lab

This statement is used to assign statement labels that will appear with the assigned GO TO statement; s is a statement label and lab is a simple integer variable. If s is a variable name instead of a statement number, the diagnostic is UNRECOGNIZED STATEMENT.

ASSIGN 10 TO LSWITCH
;
GO TO LSWITCH,(5,10,14,20)

Control is transferred to statement 10.

COMPUTED GO TO STATEMENT

GO TO (s_1[,s_2]...)[,]i

This statement acts as a many-branch GO TO where i is preset or computed prior to its use in the GO TO. The branch will go to s_i, depending on the current value of i. That is, the branch will go to the i^{th} statement in the list.

The s_i are statement labels and i is an integer variable or expression. If i < 1 or if i > m, a transfer is not executed, and the next statement following the GO TO is executed.

Limitations

- For proper operations, i must not be specified by an ASSIGN statement. No compilation diagnostic is given for this error, but the object code generated by the compiler is incorrect and the program may stop without comment.
6.4
Control Statements

COMPUTED GO TO STATEMENT (continued)

Limitations (continued)

- If i is a floating point expression, a nonfatal diagnostic will be printed, NON-INTEGER EXPRESSION NOT ALLOWED IN FORTRAN STANDARD EXECUTION MAY CONTINUE. The value of the floating expression is truncated to an integer and execution continues.

Example

```fortran
ISWITCH = 1
GO TO (10,20,30),ISWITCH :
10 JSWITCH = ISWITCH + 1
GO TO (11,21,31),JSWITCH
```

Control is transferred first to statement 10 and from there to statement 21.
IF STATEMENTS

IF statements may be three-, two-, or one-branch statements.

THREE-BRANCH ARITHMETIC IF STATEMENT

IF(exp) s₁,s₂,s₃

exp is an arithmetic expression, and the sᵢ are statement labels. This statement tests the evaluated expression exp and jumps as follows:

- If exp = 0, jump to statement s₂
- If exp > 0, jump to statement s₃
- Otherwise, exp < 0, go to s₁

Example 1

C    JUMP - ,0, + depending on value of A
     IF(A) 1,2,3

Whether A is 0 or -0, the jump to 2 is taken; if A is greater than 0, the jump is to 3, if A is less than 0, the jump is to 1.

Example 2

COMPLEX C
     IF(C-SIN(X)) 1,4,5

The mode of the evaluated expression is complex. Only the real part is tested.

Example 3

LOGICAL L
     IF(L) 1,2,3

Both true (-0) and false (+0) jump to 2. Branches to 1 and 3 will be taken only on a nonzero value of L. This is not a standard way of using a logical variable, but no diagnostic is provided. L is treated here as type integer.
6.6
Control Statements

THREE-BRANCH
ARITHMETIC
IF STATEMENT
(continued)

Example 4

IF(A/B**2) 3,6,6

The expression is evaluated and if it is zero or plus, a jump to 6 is taken.

Example 5

F = .1
E = 0.0
A = 2.
DO 40 J=1,20
40  E = E+F
41  IF(E-A) 1,2,3

At statement 41, A = 2.0 (17214000000000000000B). After 20 iterations of statement 40, E = 1.999 (17207777777777777767B); since .1 cannot be expressed exactly as a 48 bit binary number, E does not become exactly 2.0. If the floating-point expression E-A, no zero will ever be reached; therefore, the branch to 2 will never be taken. It is recommended that the zero branch for an IF test with floating point variables not be relied on.

Example 6

F = 0.1
A = 2.0
E = 0.0
DO 40 J=1,20
40  E = E+F
    IF((A-E)+1.E-10) 1,3,3

In this example, no exact zero is expected and a very small increment is added to the expression to insure that truncation or round-off does not send the branch to 1 for values of E very close to A.
IF (exp) s₁, s₂

exp is an expression. The s₁ are statement labels. The evaluated expression is tested for true (nonzero, including -0) or false (+0) condition. If exp is true, the jump is to statement s₁. If exp is false, the jump is to statement s₂.

Example 1

C    TWO BRANCH JUMP    T,F
     IF(A.GT.B.AND.A.LT.C) 5,6

(A.GT.B) is evaluated. If it is false, the branch to 6 is immediately taken. If (A.GT.B) is true, then (A.LT.C) is evaluated, and a branch to 5 is taken if it is true; otherwise a branch to 6 is taken.

Example 2

IF(A-B) 1,2

Since (A-B) is arithmetic, no -0 could result from this expression since a subtract is generated. However, the test is (false) for any nonzero value (as true) with a branch to 1; a zero will branch to 2.

Example 3

A = 0.0
B = -A
20 IF(B) 19,29

In this case B is equal to -0. A negative test is done first where -0 goes to the 19 (true) branch. (-0 in this test is true.) A zero test then branches to 29 (false), all non-zero numbers branching to 19. (The compiler might also generate a zero test first, where both +0 and -0 would be true.)
ONE-BRANCH
RELATIONAL
IF STATEMENT

IF(exp) st

exp is an expression and st is a statement. If exp is true (nonzero including -0), execute statement st. If exp is false (+0), do not execute statement st.

In an IF(exp) st statement, st may not be an END statement, nor may it be another conditional or a DO. If a simple branch is required, use the full GO TO s statement.

Examples

C IF EXP TRUE, A=2.0
   IF(A.LE.2.5) A=2.0
   IF(VALUE*4.73.GT.ATEM.OR.VALUE.LT.150.0)STEP=TRUE
   IF(P.AND.Q)GO TO 427
   IF(PARAM)PARAM=A(I)+SINF(B(3))

If the form of an IF is incorrect, the diagnostic IF STATEMENT FORMAT ERROR is provided. (See Chapter 3 for information about evaluation of expressions.)
**DO STATEMENT**

DO s[,li = exp1,exp2[,exp3]]

This statement makes it possible to repeat groups of statements and to change the value of an integer variable during the repetition. s is the statement label ending the DO loop; i is the index variable (simple integer). exp1 are the indexing parameters; they may be unsigned integer constants or simple integer variables. The initial value assigned to i is exp1; exp2 is the largest possible value assigned to i (it must be less than 100,000); and exp3 is the increment added to i after each time through the loop. If exp3 does not appear, it is assigned the value 1.

A statement label which terminates a DO loop and has not been previously referenced except in a DO statement is ignored. A later reference to such a statement will cause a missing-statement-label indication. The following example produces a diagnostic THESE STATEMENT LABELS ARE MISSING, where 5 has not been used within the loop for a branch.

```
DO S I=1,N
5 CONTINUE
GO TO 5
```

The DO statement, the statement labeled s, and all intermediate statements are contained in the DO loop. Statement s may not be an IF or GO TO statement, FORMAT declaration, or another DO loop.

**Rules**

- The indexing parameters exp1, exp2, and exp3 are either constants or simple integer variables. Subscripted variables cause a diagnostic.
DO STATEMENT  
(continued)

Rules  
(continued)

- The indexing parameters exp₁ and exp₂ must be nonzero positive integer constants or simple integer variables. If exp₁ or exp₂ is a variable, it may have a sign.

- The incrementing parameter exp₃ must be a simple positive integer constant or variable. If exp₃ is omitted, it is assigned the value 1.

- The values of exp₁, exp₂, and exp₃ may be changed during the execution of the DO loop.

- A DO requires a terminal statement. The diagnostic A DO LOOP WHICH TERMINATES AT THIS STATEMENT INCLUDES AN UNTERMINATED DO will be generated when the terminal statement is missing.
### Incorrect Code †

**DO 50 K=1,2**

50 **DO 51 J=1,2**

51 **CONTINUE**

DO 2 I=1,10

DO 2 J=1,10

; ;

2 **CONTINUE**

DO 5 K=0,M

DO 6 J=-4,10

DO 15 L1 = A, B

DO 15 L3 = 1,15.4

DO 1 J=1,N(J),2

DO 15 L=1,3,-1

**DIMENSION J(2)**

DO 1 K=1,3

DO 2 I = 5,1

DO 8 I=1,2

8 **IP(A.EQ.B)C(I)=4.0**

**DO 101 I=1,2**

WRITE (6,101)

101 **FORMAT (*.....*)**

**DO 5 N(I)=1,5**

DO 6 I=1,3

DO 8 K=1,4

8 A = B(K)

8 **CONTINUE**

DO 15 L8=1,100000

; ;

15 **CONTINUE**

---

### Diagnostic

A PREVIOUS DO TERMINATES ON THIS DO STATEMENT

A DO VARIABLE IS USED IN AN OUTER DO LOOP

THE PARAMETERS OF A DO LOOP MUST BE AN UNSIGNED INTEGER CONSTANT OR SIMPLE INTEGER VARIABLE

A DO LOOP OUT OF RANGE

A DO LOOP TERMINATES AT THIS STATEMENT

THIS STATEMENT DOES NOT FOLLOW A DO WHICH IT TERMINATES

ILLEGAL USE OF A REPLACEMENT STATEMENT

A DO LOOP WHICH TERMINATES AT THIS STATEMENT INCLUDES AN UTERMINATED DO

A DO LOOP LIMIT EXCEEDS MACHINE CAPACITY

(Note: The limit is 99999)

† Incorrect code errors are given in italics. Note that many of these statements are acceptable according to the FORTRAN 77 standard version of the DO-loop described later in this chapter.
DO STATEMENT
(continued)

The following programming sample shows standard DO loop notation.

```fortran
DIMENSION A(6,5,3)
DO 2 K=1,N
   DO 3 J=1,M,1
      WRITE (6,101) J,K
      A(J,K)=J+K
   3 CONTINUE
 2 CONTINUE
END
```

**Card Numbers and Approximate Program Location**

<table>
<thead>
<tr>
<th>Card Number</th>
<th>Approximate Program Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>14</td>
</tr>
<tr>
<td>5</td>
<td>147</td>
</tr>
<tr>
<td>6</td>
<td>149</td>
</tr>
<tr>
<td>7</td>
<td>152</td>
</tr>
<tr>
<td>8</td>
<td>154</td>
</tr>
<tr>
<td>9</td>
<td>205</td>
</tr>
<tr>
<td>10</td>
<td>210</td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

**Variable Assignments**

- \( A = N = 217 \)
- \( M = 216 \)
- \( L = 215 \)
- \( K = 214 \)
- \( J = 213 \)

**Length of Routine**

- \( N = 230 \)

**Subroutines Called**

- \( \text{EXIT} \)

**Program Space**

- \( 1727 \)

**Origin**

- \( \text{SAMP} 136 \)

**Entry Points and Locations**

- \( \text{END} 261 \)
- \( \text{STOP} 261 \)
- \( \text{OUTPTC 1316} \)

**Overall Core Use Statistics**

- \( \text{MISC} 147 \)
- \( \text{SYSMCH 909} \)
- \( \text{SYSSCIU 6 PR 512 RANFO 64 SYSSAT 912} \)

- \( A(I,J,K) = 6 \)
- \( A(I,J,K) = 7 \)
- \( A(I,J,K) = 8 \)
- \( A(I,J,K) = 9 \)
- \( A(I,J,K) = 10 \)
- \( A(I,J,K) = 11 \)
- \( A(I,J,K) = 12 \)

**Total CPU Time in Milliseconds**

- \( 149 \)

**Total Resources Used**

- \( .17 \)
DO LOOP

EXECUTION

DO s[,i] = exp1[,exp2[,exp3]]

The initial value of the index of a DO, exp1, is increased by exp3 and compared with exp2 after executing the DO loop once, and if i does not exceed exp2, the loop is executed a second time. After this step, i is again increased by exp3 and again compared with exp2; this process continues until i exceeds exp2. Control then passes to the statement immediately following statement s, and the DO loop is satisfied. This form of a loop is executed at least once. (See end of chapter for zero-trip DO loop.) Should exp1 exceed exp2 on the initial entry to the loop, the loop is executed just once and control then passes to the statement following statement s. As shown in the examples below, after the DO loop is satisfied, i is not equal to exp2 in many cases.

Example 1

N = 1
M = 5
DO 5 K=M,N
C(K) = 42.*A(K)
  : 
5 CONTINUE

After the DO is satisfied, K = 5, and the DO has been executed once. The test is done for completion of the loop at the end. The index K was not stored since it was never used as a variable, only as an index. Therefore, K is 5, not 6, after completion.

Example 2

M3 = 1
M2 = 1
M1 = 5
DO 50 I=M1,M2,M3
I = I
  : 
50 CONTINUE
**Example 2 (continued)**

After completion, I = 4. It has been decremented by M3 = -1 and stored because of the statement I = I. When the DO loop is satisfied, the index variable i is no longer well defined. Thus, this loop with M3 negative executes only once.

**Example 3**

```plaintext
DO 2 I=1,4
   A = B(I)
2 CONTINUE
```

After the DO is satisfied I = 1. The index is left in a B-register and never stored, so after the DO, the index value is one (exp1).

**Example 4**

```plaintext
DO 2 I=1,4
   K = I
   A = B(I)
2 CONTINUE
```

After the DO is satisfied I = 5. The index is forced to store by the statement K = I. When the loop is finished I = 4 + 1 since the increment (exp3) is added at the end of the loop.

**Example 5**

```plaintext
DO 20 I=1,4
20 CONTINUE
```

After the triple nest is satisfied, I = 5, J = 5, and K = 1. No store is forced in the inner loop; outer loop indices are always stored.
Example 6

DO 20 I=1,4
DO 20 J=2,4
DO 20 K=1,4
20 KN = K

After execution of the above triple nested DO loop, a store has been forced for \( K \) since \( K \) is used as a variable as well as an index, and \( I = 5, J = 5, \) and \( K = 5 \) even though \( KN = 4 \).

Example 7

\[
N = 4 \\
DO 5 K=1,N \\
IF \ (K.EQ.N) \ GO \ TO \ 6 \\
: \\
5 \ CONTINUE \\
6 \ \ldots \ldots
\]

If a transfer out of the DO loop occurs before the DO is satisfied, the value of \( K \) is preserved and may be used in subsequent statements. At the end of the loop, \( K \) is equal to \( N \) and a branch to 6 is taken. The store is forced by the expression \( K.EQ.N \). \( K = 4 \) since the incrementing at the end of the loop is bypassed.

DO NESTS

When a DO loop contains another DO loop, the grouping is called a DO nest. Nesting may not exceed 19 levels. The last statement of a nested DO loop either must be the same as the last statement of the outer DO loop or must occur before it. If \( D_1, D_2, \ldots, D_m \) represent DO statements where the subscripts indicate that \( D_1 \) appears before \( D_2, D_2 \) appears before \( D_3, \) and \( n_1, n_2, \ldots, n_m \) represent the corresponding terminuses of the \( D_i \), then \( n_m \) must appear at or before \( n_{m-1} \).
DO NESTS (continued)

The index variable of the inner nest should be the first subscript of three-dimensional arrays, where optimization is a consideration.

\[
\begin{align*}
D_1 & \quad \text{DO 30 K = 1,10} \\
D_2 & \quad \text{DO 20 J = 1,10} \\
D_3 & \quad \text{DO 10 I = 1,10} \\
& \quad \text{A (I,J,K) = B (I,J,K)*MU} \\
& \quad \text{10 CONTINUE} \\
& \quad \text{20 CONTINUE} \\
& \quad \text{30 CONTINUE}
\end{align*}
\]

DO loops may be nested in common with other DO loops.

\[
\begin{align*}
D_1 & \quad \text{DO 1 I=1,10,2} \\
& \quad \text{DO 2 J=1,5} \\
& \quad \text{DO 3 K=2,8} \\
& \quad \text{3 CONTINUE} \\
& \quad \text{2 CONTINUE} \\
& \quad \text{DO 4 L=1,3} \\
& \quad \text{4 CONTINUE} \\
& \quad \text{1 CONTINUE}
\end{align*}
\]
Limitations

- Nineteen DO loops may be nested. The twentieth DO loop will cause the compiler diagnostic THE NESTING CAPACITY OF THE COMPILER HAS BEEN EXCEEDED.

- If a FORTRAN statement is the terminal statement of more than one DO statement, the statement label of that FORTRAN statement may not be used in an IF or a GO TO statement outside the most deeply nested DO.† Even though this construct is sometimes allowed on other compilers, no diagnostic is issued.

SUBSCRIPTING
WITHIN DO LOOPS

Where variables appear with subscripts inside a DO loop, standard subscripts should be used whenever possible. (see the section of Subscripted Variables in Chapter 2 for a definition of standard subscripts.) Standard index functions should not be separated out and given a new name (which would force an unnecessary store of the index). They should be left in the subscript even when used repeatedly. Nonstandard subscripts force additional code to be generated with a DO loop. Standard subscripts result in concise object code where index functions are calculated in B-registers. However, when there are more than seven different standard index functions in a DO loop, the 8th, 9th, ... index functions will force a store for the value of these index functions. In all cases, standard index functions are more efficient than nonstandard index functions, but the first seven standard index functions are compiled most efficiently.

Example

In the following example I + 1 and I - 1 are not separated out and are not given new variable names for use in the index function.

\[
\text{DO 2 I=1,N} \quad 2 \quad A(I+1)=B(I+1)-X(I-1)*Y(I-1)
\]

EXTENDED RANGE

A DO loop has an "extended range" if a GO TO or an IF branches out of the range of the DO where code is executed and a reentry back into the DO is performed. (*Note:* An initial entry into the end of a loop is illegal without first entering at the start of that loop. This illegal entry is not flagged, however, because of implementation of extended range. Thus no diagnostic message is issued.)
Example 1

```
DO 2 I=1,N
GO TO 10
11 ......
2 CONTINUE
10 ......
GO TO 11
```

Example 1 shows an exit to statement 10, a reentry to statement 11. A reentry occurs anytime after the GO TO 10 statement.

Example 2

```
D1
  ▼
  |   
  A1  B1

D2
  ▼
  |   
  A2  C

D3
  ▼
  |   
  B2

n1
```

Example 2 shows three nested loops with D2 and D3 at the same nesting level. If an exit is taken at A1, reentry may be effected at points B1, B2, and B3. Reentry may not be taken into loop D2 or D3, except at the DO statement, since the control variables will not have been set. If an exit is taken at A2, possible reentry may be made at C, B1, B2, or B3. No error messages are generated for improperly positioned reentries.
In example 3 an extended range DO loop is within the extended range of another loop. The control variables and the exp1, exp2, exp3 parameters may not be redefined during execution of the immediate or extended range of that DO. Care should be taken in defining the terminal statement of nested DO loops whenever extended range and reentry are used. Indices should be used as variables to force a store when using extended range. No error messages are generated for improperly positioned reentries.
The compiler will optimize the innermost DO of a nest so that the fastest possible time is achieved. The optimizing will be done provided none of the following statements appear inside the range of the innermost DO.

- A CALL to a subroutine or programmer-defined function
- Any input/output statement
- An assigned GO TO (without a list)
- Any branch out of the range of the DO
- A function call by name where an index variable is either in COMMON or in the parameter list

The optimizer code may be turned off by placing the number corresponding to a particular optimizing technique on the *FORTRAN card. The following list of codes pertains to DO loop optimization.

*FORTRAN, 1

Do not remove constant statements from a DO loop.

Example 1

```
DO 2 I=2,N
  A = B
  
2 CONTINUE
```

A *FORTRAN card which allows full optimization will remove the statement A = B from inside the DO loop.

A *FORTRAN, 1 card will turn off the optimization, and A = B will remain inside the DO loop.
Example 1 (continued)

When, under optimization, the compiler removes this constant from the loop, a nonfatal compiler error occurs, and a message is printed locating the statement displaced.

Example 2

```
DO 2 I=2,N
C = B
CALL T
2 CONTINUE
```

The statement $C = B$ will not be removed because the `CALL` statement inside the loop turns off the optimization.

Example 3

```
COMMON B
DO 11 I=1,N
A = B
R = COS(X(I))
  :
11 CONTINUE
```

A function type subprogram provided by the system, however, does allow optimization if no codes appear on the *FORTRAN card to turn off the optimization. The statement $A = B$ will be removed from the DO loop. If a subscript value appears in COMMON or in the parameter list of a function reference, the code will not be optimized.

*FORTRAN,2

Do not remove constant subexpressions from a DO.

```
DO 9 I=1,N
A = (B*Z)+(B*Z)+S(I)
9 CONTINUE
```
If the code is optimized (*FORTRAN), the phrase (B^Z) will be calculated ahead of the loop and stored in a temporary cell. Inside the loop, the save cell is recovered and used in the expression. No notice is given in the diagnostic list in this case.

If the code is not optimized (*FORTRAN,2), (B^Z) is left inside the loop.

*FORTRAN,3
Do not precalculate addresses in index registers.

```
DO 9 I=1,N
  S(I) = R(J)
9 CONTINUE
```

If the code is optimized (*FORTRAN), the address of S_i is calculated in an index register outside the loop, as well as R_j. Incremented addresses in index registers are used inside the loop for bringing in addresses and storing addresses.

Without optimization (*FORTRAN,3), all addressing for S_i and R_j is done inside the loop.

*FORTRAN,4
Do not optimize variant global functions. A global index function depends on a factor which is common throughout the range of a DO loop. A variant changes as a function along with the DO variable. A variant global function is a combination of these two factors.

```
KK = 2
DO 21 L=2,30
  J = L+1
21  R(J) = R(J+1)+V(J,KK)
```
*FORTRAN, 4

Using *FORTRAN, the address calculation of V(J,KK) is optimized. KK is a "global" index, whereas the complete address is "variant". KK is removed from the loop, and the global factor is saved in a B-register.

Without optimization using a *FORTRAN, 4 card, a memory reference to KK is done each time inside the loop.

*FORTRAN, 13

Do not set B-registers to constants.

```
DO 20K=1,18,2
   S(K) = S(K) + ...

20 CONTINUE
```

Without optimization, the 2 (m3 in the DO loop notation DO s I = m1, m2, m3) is saved in a B-register outside the loop. The index is incremented using this B-register.

Without optimization as on a *FORTRAN, 13, the constant 2 would be used throughout the loop.

*FORTRAN, 18

With a *FORTRAN, 18 card, all of the previous DO loop optimization is turned off.

*FORTRAN, 25

This card causes the compiler to interpret DO loops according to the FORTRAN 77 standard. (See the next section of this chapter.)

*FORTRAN, 60

Do not optimize any of the program whatsoever. This option removes FORTRAN and object code optimization.

Wherever possible, the following techniques are always applied.

- Use A0 as a constant 1 in the logic. (Any external function other than the ** operator will turn this off.)
Control Statements

- Place no statement label on a CONTINUE statement unless it is specifically referred to in a branch statement.

- Fetches are placed at the end of the DO after incrementing. At the last execution through the DO loop, an extra fetch is done. This extra fetch may cause a bounds error beyond the last array element. Since this technique speeds loop execution time, it is worth taking the chance of getting a bounds error.

*Note:* The use of variable dimensions in a subprogram causes more indexing logic to be generated for DO loops. If speed, rather than generality, is important, do not use variable dimensions.
CONTINUE

The CONTINUE statement is most frequently used as the last statement of a DO loop to provide loop termination when a GO TO or IF would normally be the last statement of the loop. If CONTINUE is used elsewhere in the source program, it acts as a do-nothing instruction and control passes to the next sequential program statement. The CONTINUE statement should contain a statement label in columns 1 through 5. If it is not labeled, it is ignored by the compiler. If a label contains a character in column 6, the diagnostic ILLEGAL MARK IN COLUMN SIX is printed. If the label contains a letter, ILLEGAL CHARACTER APPEARS IN A CONSTANT is the diagnostic.

PAUSE

PAUSE [string]  string ≤ 5 characters

PAUSE stops program execution with the message string displayed on the console. An operator entry from the console can continue (xxx GO) or terminate (xxx DROP) the program. (xxx is the job control point on the CRT.) Program continuation goes to the statement immediately following PAUSE. If string is omitted, it is understood to be blank. If string appears as more than five characters, only the first five characters appear with the PAUSE at the console.

END

END must be the very last statement in a program or subprogram, and there should be only one END card for each unit. It is executable in the sense that it affects termination of the program in the absence of a CALL EXIT or a return from a subprogram in the absence of a RETURN.
CALL EXIT

EXIT is a subprogram on the system library. (See the Library Routines Manual for a discussion of the EXIT subprogram.) This routine will terminate a program normally. An alternate form is the FORTRAN statement STOP. It may be omitted immediately preceding the END card.

STOP [string]  string ≤ 6 characters

STOP is an alternate entry point to the subroutine EXIT. This routine will perform a normal termination and print the value string, if it is present in the STOP statement. If string is omitted, it is understood to be blank.
IF/THEN/ELSE

GENERAL STRUCTURE

The IF/THEN/ELSE structure is used to control the execution sequence in a program. The structure begins with an IF-THEN (block IF) statement and ends with an END IF statement. Optionally, one or more ELSE IF statements and one ELSE statement can also be used in the structure. The IF-THEN, ELSE IF and ELSE statements have blocks of statements associated with them, so the general IF/THEN/ELSE structure looks like the following:

```
IF(exp1) THEN
   [statement block]
ELSE IF(exp2) THEN
   [statement block]
  ...
ELSE IF(expn) THEN
   [statement block]
ELSE
   [statement block]
END IF
```

where expi is a logical expression. Each of these statements and the associated blocks are described in detail below.
Nesting Levels

Each of the statement blocks within an IF/THEN/ELSE structure may itself contain another IF/THEN/ELSE structure. That is, these structures may be nested. However, the inner IF/THEN/ELSE structure must be entirely contained within one statement block of the outer structure. There may be as many as 21 nesting levels.

Transfer of Control

It is important to note that control may not be transferred into a statement block from outside that block. The NCAR compiler does not check for statements which result in such a transfer, but the effect of the transfer is undefined.
**IF THEN STATEMENT**

**Form**
The IF-THEN (block IF) statement has the form

\[ \text{IF}(\text{exp}) \text{ THEN} \]

where \( \text{exp} \) is a logical expression.

**Block**
This statement has associated with it an IF block which consists of all the executable statements, if any, following the IF-THEN statement up to the next ELSE IF, ELSE or END IF statement on the same nesting level.

**Execution**
The execution of an IF-THEN statement causes evaluation of the logical expression \( \text{exp} \).

- If the value of \( \text{exp} \) is true, execution continues with the first statement of the IF block. After execution of the last statement of the block, control is transferred to the corresponding END IF statement.

- If the value of \( \text{exp} \) is false, control is transferred to the next ELSE IF, ELSE or END IF statement on the same nesting level.
ELSE IF STATEMENT

Form

The ELSE IF statement has the form

ELSE IF(exp) THEN

where exp is a logical expression.

Block

This statement has associated with it an ELSE IF block which consists of all the executable statements, if any, following the ELSE IF statement up to the next ELSE IF, ELSE or END IF statement on the same nesting level.

Execution

The execution of an ELSE IF statement causes evaluation of the logical expression exp.

- If the value of exp is true, execution continues with the first statement of the ELSE IF block. After execution of the last statement of the block, control is transferred to the corresponding END IF statement.

- If the value of exp is false, control is transferred to the next ELSE IF, ELSE or END IF statement on the same nesting level.
ELSE STATEMENT

Form

The ELSE statement has the form

ELSE

Block

This statement has associated with it an ELSE block which consists of all the executable statements, if any, following the ELSE statement up to the next END IF statement on the same nesting level.

Execution

The execution of an ELSE statement itself has no effect. Once control is transferred to an ELSE statement, normal execution sequence continues with the first statement of the ELSE block.
**END IF STATEMENT**

*Form*

The END IF statement has the form

```
END IF
```

*Execution*

Execution of the END IF statement has no effect. The normal execution sequence continues with the first executable statement following the END IF statement.
Control Statements

Flow diagram for IF/THEN/ELSE structure:

IF(exp₁) THEN

[block₁]

ELSE IF(exp₂) THEN

[block₂]

ELSE IF(expₙ₋₁) THEN

[blockₙ₋₁]

ELSE

[blockₙ]

END IF
Sample Program Illustrating IF/THEN/ELSE

```
PROGRAM BKIF

C TEST BLOCK IF STATEMENT, SECTION 11.6 IN FORTRAN 77

IMPLICIT INTEGER (A,B,C,D)
PARAMETER (L=5)
DIMENSION A(L),B(L),C(L),D(L)

PRINT 10

10 FORMAT (#1x)

K=0
N=1

1 CONTINUE

IF (K .EQ. 0) THEN
   DO 2 I=1,L
   A(I)=1
   2 CONTINUE
   PRINT 101,K

ELSE IF (K .LE. L) THEN
   FORMAT (#1IF BLOCK 1 K=I2/)
   PRINT 102,K
   IF (N .LE. J) THEN
      PRINT 103,J,N
      DO 4 I=1,L
         C(I)=A(I)
         4 CONTINUE
      END IF
      ELSE IF (N .LE. L) THEN
      PRINT 104,J,N
      DO 6 I=1,L
         D(I)=D(I)+10
         6 CONTINUE
      ELSE
      PRINT 105,J,N
      END IF
      END IF
      K=K+1
      GC TO 1
      END

END IF
```
Output from Sample Program BLKIF

IF BLOCK 1 K = 0
ELSE IF 1 K = 1
IF BLOCK 2 N = 0
ELSE IF 1 K = 2
ELSE IF 2 N = 1
ELSE IF 1 K = 3
ELSE IF 2 N = 2
ELSE IF 1 K = 4
ELSE IF 2 N = 3
ELSE IF 1 K = 5
ELSE IF 2 N = 4
ELSE I 1 K = 6
B(I) = 5 5 5 5 5 5
O(I) = 40 40 40 40 40
STOP
DO-LOOP (FORTRAN 77)  
The FORTRAN 77 version of the DO loop will be accepted when the *FORTRAN, 25 control card is used.

DO-LOOP STRUCTURE  
The DO-loop structure makes it possible to repeat groups of statements while changing the value of a variable called the DO variable. As described below, the DO-loop consists of a DO statement followed by a group of statements called the DO-loop range. The last statement of the range is referred to as the terminal statement.

DO STATEMENT  
The DO statement, used to specify the beginning of a DO-loop, has the form:

\[
\text{DO } s[,] i = \text{exp}_1,\text{exp}_2[,\text{exp}_3]
\]

Where:  
- \( s \) is the label of the terminal statement of the DO-loop
- \( i \) is the name of an integer, real or double precision variable, called the DO variable.
- \( \text{exp}_1, \text{exp}_2 \) and \( \text{exp}_3 \) are each an integer, real or double precision expression.

DO-LOOP RANGE  
The range of a DO-loop consists of the executable statements between the DO statement and the associated terminal statement. The terminal statement is the last statement within the range of the DO-loop.
Control Statements

**TERMINAL STATEMENT**

The terminal statement of a DO-loop must follow the associated DO statement and must be an executable statement. The terminal statement must not be a GO TO, RETURN, STOP, END, DO, ELSE IF, ELSE, END IF, or any IF statement other than a logical IF. If the terminal statement is a logical IF, it may contain any executable statement except a DO, block IF, ELSE IF, ELSE, END IF, END or another logical IF statement.

**Rules**

- More than one DO-loop may have the same terminal statement.

- If a DO statement appears within the range of another DO-loop, the entire range of the DO-loop must be contained within the range of the outer loop.

- If a DO statement appears within an IF block, ELSE IF block or ELSE block, the entire range of the DO-loop must be contained within that block.

- If a block IF statement appears within the range of a DO-loop, the corresponding END IF statement must also appear within the range of that DO-loop.
DO-LOOP EXECUTION

The execution of a DO-loop consists of the following sequence of operations, each of which is described in more detail below:

- Execution of the DO statement
- Loop control processing
- Execution of statements in the range of the DO-loop
- Incrementation processing

The last three steps will be repeated until either loop control processing determines that no iterations remain to be performed or until the execution of a statement in the loop range causes a jump out of the loop.
The execution of a DO statement consists of the following sequence of operations:

- The initial parameter \( m_1 \), the terminal parameter \( m_2 \), and the incrementation parameter \( m_3 \) are established by evaluating \( e_1 \), \( e_2 \) and \( e_3 \) respectively. If necessary, the result is converted to the type of the DO variable \( i \) according to the rules of arithmetic conversion (see chapter 3). If \( e_3 \) does not appear, \( m_3 \) has the value one. \( m_3 \) must not be zero.

- The DO variable \( i \) becomes defined with the value of the initial parameter \( m_1 \).

- The iteration count is established according to the formula

\[
\text{MAX} \left( \text{INT} \left( \frac{m_2 - m_1 + m_3}{m_3} \right), 0 \right)
\]

Note that the iteration count is zero whenever:

- \( m_1 > m_2 \) and \( m_3 > 0 \),
- \( m_1 < m_2 \) and \( m_3 < 0 \).

At the completion of execution of the DO statement, loop control processing begins.
To determine whether further execution of the range of the DO-loop is required, the iteration count is tested.

- If the iteration count is zero, the DO-loop range is *not* executed. The execution sequence continues as though the DO-loop were not present.

- If it is not zero, the first statement in the range is executed next.

Statements in the range of a DO-loop are executed until the terminal statement is reached. Unless execution of one of the statements in the range results in the transfer of control out of the loop, execution of the terminal statement is followed by incrementation processing described below. Note that the DO variable may neither be redefined nor become undefined during the execution of the range of the DO-loop.

Incrementation processing consists of the following sequence of operations:

- The DO variable is incremented by the value of the incrementation parameter \( m_3 \).

- The iteration count is decremented by one.

- Execution continues with loop control processing described above.
Initially inactive, a DO-loop becomes active only when its DO statement is executed.

An active DO-loop becomes inactive only when:

- loop control processing determines the iteration count to be zero
- a RETURN statement is executed within its range
- control is transferred outside the range of the DO-loop
- program execution is terminated for any reason

When a DO-loop becomes inactive, the DO variable retains its last defined value unless it has become undefined.

Transfer of control into the range of a DO-loop from outside the range is not permitted. The NCAR compiler does not check for statements which result in such transfers, but the effect of the transfer is undefined.
Flow diagram for DO-loop (FORTRAN 77):

DO $s[,] i = e_1, e_2, e_3$

[DO-loop range]

iteration

count

count

zero

not zero

DO-loop
range
PROGRAM, FUNCTION, AND SUBROUTINE

ENTRY POINTS
TRANSFER STATEMENTS
ARITHMETIC STATEMENT FUNCTION

PROGRAM pgm
SUBROUTINE sub [(d1[,d2]...)]
CALL en [(a1[,a2]...)]
[typ] FUNCTION fun [d1[,d2]...]
fun (a1[,a2]...)
fun (d1[,d2]...) = exp
BLOCK DATA [sub]
RETURN
END
ENTRY en
EXTERNAL en1[,en2]...
A job is a complete set of control cards, programs, and data which may be submitted to the computer to be compiled and/or assembled, loaded, and run. FORTRAN programs may be main programs or subprograms (functions and subroutines). A job will contain one main program with or without subprograms.

A program is an ordered set of computer instructions which performs a mathematical algorithm or some data processing task or tasks. If the job has only one main program with few, if any, subprograms, it is a series of tasks done in order and is said to be written "in-line." In contrast, in a "modular" design, separate tasks may be programmed in subprograms. The main program then calls the subprograms as the overall program design requires. The main program then may be referred to as a "driver," and the collection of programs is said to be designed using a modular approach.

In the following pages, main program, subroutine, function, and arithmetic statement are defined, and the use of parameters is discussed.
7.2
Program, Function, and Subroutine

**MAIN PROGRAM**

The first statement of a main program must be the **PROGRAM** card. The form is

```
PROGRAM pgm
```

where `pgm` is a symbolic identifier consisting of one to seven alphanumeric characters and starting with a letter.

The program card is optional; however, the main program must have an **END** card. A main program may not be called as a subprogram or called by itself recursively.
SUBPROGRAMS

A subprogram is a separate computational procedure which may be referenced for execution by a main program or another subprogram. Subprograms may be called by or may call another subprogram, but a subprogram may not call itself. Communication of parameters may be done by means of an argument list or through a COMMON declaration (see Chapter 5). When an argument list is used, a distinction is made between formal and actual parameters. Parameters are transmitted by name, not by value. If a constant is a parameter, the compiler provides a temporary name for it.

FORMAL PARAMETERS

A formal parameter is the name used in the argument list of the subprogram being called. These names are local to the subprogram and are dummy names used to reference the actual parameters specified by the calling program. Examples of formal parameters are the variables in the following argument lists:

Rules

SUBROUTINE SUB(X,Y,YMIN,XMAX,ITITL)
FUNCTION FUN(V,W,I)

- Formal parameters may be array names, simple variables, or names of library functions and function and subroutine subprograms.

- Since formal parameters are local to the subprogram containing them, they may be the same as names appearing outside the procedure with no conflict.

- No element of a formal parameter list may appear in an EQUIVALENCE or DATA statement within the subroutine. If it does, the compiler diagnostics resulting are

  FORMAL PARAMETER ERROR IN EQUIVALENCE
  ILLEGAL USE OF FORMAL PARAMETER IN DATA STATEMENT
Rules (continued)

- When a formal parameter represents an array, it should be dimensioned within the subprogram. If it is not dimensioned, the array name must appear without subscripts in the subprogram, and only the first element of the array is then available to the subprogram being called. Otherwise, the subprogram will look for a missing subroutine.

Actual Parameters

An actual parameter is specified in the argument list of the calling program, and is often a variable which is stored in the calling program.

The following are permissible forms of an actual parameter:

- Arithmetic expression
- Logical expression
- Constant
- Simple or subscripted variable
- Array name
- Function or subroutine name (see EXTERNAL statement)

CALL SUBA(A,B,C(1,2),1.2E-3,4HTEST)
CALL DER(T,I,J,COS(X))
CALL UUB(A,B,COS(X),B,4.13E-2,(2.1,4.3),.TRUE.)

Note: Masking expressions are not legal.
Example 1

When an actual parameter is the name of a function or subroutine, that name must also appear in an EXTERNAL statement in the calling program. Note that functions from the FORTRAN library may not be used as externals. Functions supplied by the program are acceptable.

```fortran
PROGRAM TEST FUNCTION ST(X) EXTERNAL ST,SU ST=X*X*X RETURN END CALL S2(ST,X,ANS) CALL S2(SU,A,AN) DATA U/.33333333333333/ SU=X**U RETURN END SUBROUTINE S2(F,X,R) R = F(X) RETURN END
```

Example 2

Actual and formal parameters must agree in order, type, and number. No diagnostic is provided, but the code will be incorrect.

```fortran
PROGRAM MAIN CALL UB(A,42,4.13E-2,I,(2.1,4.3)) END SUBROUTINE UB(A,K,T,L,C) COMPLEX C END
```
Parameters are treated as names by a subprogram. Any name in the parameter list may be used to find a value and to store a value. Therefore, changes to any parameter may be made by the subprogram, whether the actual parameter is a constant or a variable. Care should be used with constants in a parameter list because the subprogram can change the parameter value, even though the actual parameter is the same number in the CALL statement. Note in the following example that the statement $X = 2.0$, where $X$ is a formal parameter, changes the actual parameter, which is a constant $1.0$.

```plaintext
PROGRAM MAIN

CALL S(1.0,A)
END

SUBROUTINE S(X,Y)

T=X+Y
X=2.0
RETURN
END
```

After the call to subroutine $S$, any further reference to the constant $1.0$ in the main program will generate incorrect results since the storage cell allocated to hold the constant $1.0$ was reset to $2.0$ during execution of the subroutine. To avoid this hazard, subroutine $S$ can be rewritten as:

```plaintext
SUBROUTINE S(XX,YY)
X=XX
Y=YY
T=X+Y
X=2.0
RETURN
END
```
A subroutine subprogram may or may not return values, depending on the parameter list or COMMON. No value is associated with the subroutine name.

The first statement of a subroutine subprogram must be in the following form:

```
SUBROUTINE sub [(d1,[,d2]...)]
```

where sub is the alphanumeric identifier and d_1 are formal parameters. The parameter list is optional.

**Rules**

- The subroutine name may not appear anywhere within the subroutine except in the SUBROUTINE statement itself.
- Any of the formal parameters may be used to return output to the calling program if those parameters are defined or redefined in the subprogram procedure.
CALL STATEMENT

To call a subprogram, use the following statement:

```
CALL en [(a_1, a_2, ...)]
```

where `en` is the name of the entry into the subroutine being called, and `a_i` are actual parameters. More than 64 arguments will produce the diagnostic MORE THAN 64 ARGUMENTS IN CALL OR FUNCTION. The subprogram name may not appear in any declarative statement in the calling program, with the exception of the EXTERNAL statement.

The CALL statement transfers control to the subroutine. A RETURN statement in the subroutine returns control to the next executable statement following the CALL statement in the calling program. If the CALL statement is the last statement in a DO loop, looping continues until the DO loop is satisfied; each trip through the loop executes a CALL to the subroutine.

**Example 1**

A, B, C are formal parameters.

```
SUBROUTINE TEST(A,B,C)
  C = A + B*EXP(A+.3)
  RETURN
END
```

Some calls have different actual parameters. For example:

```
CALL TEST(A,B,C)
CALL TEST(A(I+1)-T/UR(I)+C(J),T)
CALL TEST(SIN(Q5),VEC(I+J),OVEC(L))
```
Example 2

COMMON is used for matrix variables.

SUBROUTINE MATMUL
  COMMON/MAT/X(20,20),Y(20,20),Z(20,20)
  DO 10 I=1,20
  DO 10 J=1,20
    Z(I,J) = 0
  DO 10 K=1,20
  10 Z(I,J) = Z(I,J) + X(I,K)*Y(K,J)
  RETURN
END

Operations in MATMUL are performed on variables contained in the common block MAT. This block must be defined in all programs calling the subroutine MATMUL.

PROGRAM TEST
  COMMON/MAT/AB(20,20),CD(20,20),EF(20,20)
  ...
  CALL MATMUL
  ...
END
CALL STATEMENT
(continued)

Example 3

SUBROUTINE AGMT(SUB, ARG)
COMMON/ABL/XP(100)
ARG=0.
DO 5 I=1,100
5 ARG=ARG+XP(I)
CALL SUB(ARG)
RETURN
END

This example uses both COMMON and an argument list. The subroutine used as an actual parameter must have its name declared in an EXTERNAL statement, as in the following calling program.

PROGRAM MAIN
...
COMMON/ABL/ALST(100)
EXTERNAL SQDEV
CALL AGMT(SQDEV,V1)
...
END

The subroutine SQDEV, which is passed to subroutine AGMT as an EXTERNAL reference, also needs to be defined:

SUBROUTINE SQDEV(ZSUM)
COMMON/ABL/Z(100)
ZMEAN=ZSUM/100.
DO 5 I=1,100
5 Z(I)=(Z(I)-ZMEAN)*(Z(I)-ZMEAN)
RETURN
END
A function subprogram is an independent set of instructions which returns a result to the program for the function name. This result is placed in the equation at the point where the function is referenced. Parameters are used in the same way as in subroutines.

PROGRAM MAIN
    ...
    T = 1. - F(X)
    ...
END
FUNCTION F(A)
    ...
    F = A*A + R
    RETURN
END

The result F, calculated in the function subprogram, is placed directly in the equation for T in the main program where F(X) is referenced. (There are library functions such as COS(X), EXP(X), etc., that are used in this way.)

The general syntax for a function definition statement is:

[type] FUNCTION fun [(d_1[,d_2]...)]

The first statement of a function subprogram must be one of the following forms:

FUNCTION fun [d_1[,d_2]...]
INTEGER FUNCTION fun [(d_1[,d_2]...)]
DOUBLE FUNCTION fun [(d_1[,d_2]...)]
DOUBLE PRECISION FUNCTION fun [(d_1[,d_2]...)]
COMPLEX FUNCTION fun [(d_1[,d_2]...)]
REAL FUNCTION fun [(d_1[,d_2]...)]
LOGICAL FUNCTION fun [(d_1[,d_2]...)]
where fun is an alphanumeric identifier and d₁ are formal parameters. A FUNCTION statement may appear without parameters. The mode of the results stored in the name is determined in the function name statement or with the conventional type indication used for variables. When the type indicator is omitted, the mode is determined by the first character of the function name.

```
INTEGER FUNCTION F2(I)
```

is the same as the following two statements:

```
FUNCTION F2(I)
INTEGER F2
```

The name of the function must not appear in a DIMENSION declaration in the calling program. No diagnostic is provided, but the program will not call the function. The name must appear within the function subprogram at least once as the left-hand identifier in a replacement statement; otherwise the diagnostic will be A FUNCTION NAME WAS NOT USED AS A REPLACEMENT STATEMENT.

The function name may also appear as an element of an input list within the subprogram or as an actual parameter of a subprogram reference.
where `fun` identifies the function being called. It is an alphanumeric identifier, and its type is determined in the same way as a variable identifier. DOUBLE and COMPLEX function references must have a TYPE statement in the calling program. If the default option for typing variables has not been assumed, then specific type statements for the function name are required in the calling program. The `a_i` are actual parameters. A function reference must have at least one parameter, even if it is a dummy. A function reference may appear any place in an expression that an operand may be used. The evaluated function has a value associated with the function name. This value is returned and used in the expression as an operand.

When a function reference is encountered in an expression, control is transferred to that function. When a RETURN statement or an END in the function subprogram is encountered, control is returned to the statement containing the function reference, where expression evaluation is continued.

**Example 1**

```fortran
FUNCTION GRATER(A,B)
   IF(A.GT.B)GO TO 2
   GRATER = A-B
   RETURN
2   GRATER = A+B
   RETURN
END
```

Reference to the function `GRATER` might be:

```plaintext
W(I,J) = FA + FB - GRATER(C-D,3.*AX/BX)
```
FUNCTION PHI(ALPHA, PHI2)
  PHI = PHI2(ALPHA)
  RETURN
END

This function can be referenced:

EXTERNAL SNE
C = D-PHI(Q(K), SNE)

where SNE is later defined as:

FUNCTION SNE(S)
  SNE = S
  IF(S.LE.0) RETURN
  SNE = -SNE
  RETURN
END

The replacement statement with the function PHI will be executed as if it had been written PHI = SNE(Q(K)).

The name of the function may not be used in an EXTERNAL statement within itself. The diagnostic is VARIABLE IDENTIFIER IN EXTERNAL STATEMENT.
The statement function is a single expression defined exclusively in the program or subprogram containing it. The definition is placed before the first executable statement of the program and is of the form

\[ \text{fun}(d_1[,d_2]...) = \text{exp} \]

The name fun of the statement function is an alphanumeric identifier; a value is always associated with the name. The name of the statement function applies only to the program containing the definition, and it must not appear in an EXTERNAL statement.

The \( d_i \) are formal parameters and must be simple variables or array names. The \( d_i \) are thus dummy arguments indicating type, number, and order of arguments. They may have the same name as variables appearing in the rest of the program, since they are formal parameters only.

The expression \( \text{exp} \) may be any arithmetic expression which contains references to library functions, other statement functions, or function subprograms.

The identifiers appearing in the expression which are not parameters have their current program values.

**Examples**

**Definition:**

\[
\begin{align*}
\text{DIMENSION B(10,10)} \\
Q2(X,Y,Z) &= 2X + Y^2 - Z\cdot\cos(A) \\
C(A,I,J) &= A(I,J)
\end{align*}
\]

**Reference:**

\[
\begin{align*}
\text{AND} &= Q2(A+B,\cos(Y),T) + A\cdot\cos(Y) \\
Z &= C(B,1,2)
\end{align*}
\]
Q2 is the name of the statement function; the actual parameters may be any arithmetic expression. In the example, \( A+B \) is the actual parameter for the formal parameter \( X \), \( \cos(Y) \) for \( Y \), and \( T \) for \( Z \). Actual parameters may be subscripted variables.

During compilation, the arithmetic statement function definition is inserted in the code where the actual reference is made. This reference appears as an operand in an expression.

**Rules**

- The statement function name must not appear in a DATA, DIMENSION, EQUIVALENCE, COMMON, or EXTERNAL statement; the name may appear in a TYPE declaration, but cannot be dimensioned.

- Statement function names must not appear as actual or formal parameters in subprogram references, since they are defined only locally.

- Actual and formal parameters must agree in number and order. However, the type of the statement function name or the type of formal parameters is ignored; the mode of the evaluated statement function is determined by the mode of the expression given the actual parameters.† When the number of entries in the call is not the same as the number of entries in the statement function, the diagnostic ARITHMETIC STATEMENT FUNCTION PARAMETERS DO NOT AGREE IN NUMBER is provided. If the order does not agree, no diagnostic is provided and care must be taken in defining actual parameters.

† This is not standard in the American National Standard Institute interpretation of the FORTRAN language.
• All statement functions must precede the first executable statement of the program or subprogram, but they must follow all declarative statements (DIMENSION, TYPE, etc.), or a diagnostic is provided that DECLARATIVE STATEMENTS MUST PRECEDE ALL ARITHMETIC STATEMENTS.

• A statement function may not call itself.

• A statement function must have at least one argument

• Statement functions may generate more instructions in the code than the same program written without the use of statement functions. When the argument is an index function, the compiled code generates a nonstandard index function. This may slow down the program execution.

Examples

Definition:

COMPLEX Z
Z(X,Y) = (1.,0.)*EXP(X)*COS(Y) + (0.,1.)*EXP(X)*SIN(Y)

Reference:

COMPLEX ZZ
ZZ = Z(A,B)
Mathematical algorithms, such as sine and tangent, that are used frequently have been written and stored in a reference library and are available to the programmer through the compiler. These may be functions or subroutines or in-line algorithms.

A number of these routines are available as in-line algorithms. Code to accomplish these functions is inserted directly in the program requesting them by the compiler. Table 1 gives the in-line functions available on the NCAR compiler.

Certain mathematical routines and functions are supplied by the system library and need not be submitted with the program. These routines are automatically loaded if the user program references them. The names of the FORTRAN library routines, listed alphabetically, are given in Table 2.
Table 7-1. In-Line Functions Available on NCAR Compiler

<table>
<thead>
<tr>
<th>Description</th>
<th>Forms</th>
<th>Parameter Type</th>
<th>Result Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Absolute Value</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>To obtain absolute value</td>
<td>ABS(X)</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td></td>
<td>[ABSF(X)]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>To obtain absolute value</td>
<td>IABS(I)</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td>[XABSF(I)]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>To obtain absolute value</td>
<td>DABS(D)</td>
<td>Double</td>
<td>Double</td>
</tr>
<tr>
<td><strong>Conversion</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>To truncate a real argument</td>
<td>AINT(X)</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td></td>
<td>[INTF(X)]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>To truncate a real number</td>
<td>INT(X)</td>
<td>Real</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td>[XINTF(X)]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>IFIX(X)</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td>[XFIXF(X)]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>To float an integer number to floating point</td>
<td>FLOAT(I)</td>
<td>Integer</td>
<td>Real</td>
</tr>
<tr>
<td></td>
<td>[FLOATF(I)]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>To obtain the imaginary part of a complex argument</td>
<td>AIMAG(C)</td>
<td>Complex</td>
<td>Real</td>
</tr>
<tr>
<td>To obtain a double-precision number from a real number</td>
<td>DBLE(X)</td>
<td>Real</td>
<td>Double</td>
</tr>
<tr>
<td>To obtain the real part of a complex argument</td>
<td>REAL(C)</td>
<td>Complex</td>
<td>Real</td>
</tr>
<tr>
<td>To convert real to complex</td>
<td>CMPLX(X₁,X₂)</td>
<td>Real</td>
<td>Complex</td>
</tr>
<tr>
<td>To obtain the complex conjugate of an argument</td>
<td>CONJG(C)</td>
<td>Complex</td>
<td>Complex</td>
</tr>
<tr>
<td>To convert double-precision argument to single-precision</td>
<td>SNGL(D)</td>
<td>Double</td>
<td>Real</td>
</tr>
</tbody>
</table>
Table 7-1. In-Line Functions Available on NCAR Compiler (continued)

<table>
<thead>
<tr>
<th>Description</th>
<th>Forms</th>
<th>Parameter Type</th>
<th>Result Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Location</strong>†</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>To determine the address of the variable name stated in the argument</td>
<td>ALOC(VAR_i)</td>
<td>Not applicable</td>
<td>Real</td>
</tr>
<tr>
<td></td>
<td>[LOCF(VAR_i)]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LOC(VAR_i)</td>
<td>Not applicable</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td>[XLOCF(VAR_i)]</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Maximum (or minimum) of n Arguments</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AMAX0(I_1,I_2,...,I_n)</td>
<td>Integer</td>
<td>Real</td>
</tr>
<tr>
<td></td>
<td>[MAXOF(I_1,I_2,...,I_n)]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AMAX1(X_1,X_2,...,X_n)</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td></td>
<td>[MAX1F(X_1,X_2,...,X_n)]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MAX0(I_1,I_2,...,I_n)</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td>[XMAXOF(I_1,I_2,...,I_n)]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MAX1(X_1,X_2,...,X_n)</td>
<td>Real</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td>[XMAX1F(X_1,X_2,...,X_n)]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AMINO(I_1,I_2,...,I_n)</td>
<td>Integer</td>
<td>Real</td>
</tr>
<tr>
<td></td>
<td>[MINOF(X_1,X_2,...,X_n)]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AMIN1(X_1,X_2,...,X_n)</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td></td>
<td>[MIN1F(X_1,X_2,...,X_n)]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MINO(I_1,I_2,...,I_n)</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td>[XMINOF(I_1,I_2,...,I_n)]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MIN1(X_1,X_2,...,X_n)</td>
<td>Real</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td>[XMIN1F(X_1,X_2,...,X_n)]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

† The "real" mode of result may not be practical when dealing with addresses. Integer is the mode of result used with most applications.
### Table 7-1. In-Line Functions Available on NCAR Compiler (continued)

<table>
<thead>
<tr>
<th>Description</th>
<th>Forms</th>
<th>Parameter Type</th>
<th>Result Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Modulus</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$X_1 - (X_1/X_2)X_2$</td>
<td>AMOD($X_1,X_2$)</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td></td>
<td>[MODF($X_1,X_2$)]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>where</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$(X_1/X_2) = integer$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>part</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MOD($I_1,I_2$)</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td>[XMODF($I_1,I_2$)]</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Positive Difference</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$a_1 - \text{MIN}(a_1,a_2)$</td>
<td>DIM($X_1,X_2$)</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td></td>
<td>[DIMF($X_1,X_2$)]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>IDIM($I_1,I_2$)</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td></td>
<td>[XDIMF($I_1,I_2$)]</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Sign</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sign of $X_2$ times absolute</td>
<td>SIGN($X_1,X_2$)</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>value of $X_1$</td>
<td>[SIGNF($X_1,X_2$)]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sign of $I_2$ times absolute</td>
<td>ISIGN($I_1,I_2$)</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td>value of $I_1$</td>
<td>[XSIGNF($I_1,I_2$)]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 7-1. In-Line Functions Available on NCAR Compiler (continued)

<table>
<thead>
<tr>
<th>Description</th>
<th>Forms</th>
<th>Parameter Type</th>
<th>Result Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Logical Functions</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logical product:</td>
<td>A=AND(X,...)</td>
<td>any type</td>
<td>no mode</td>
</tr>
<tr>
<td>Bit-by-bit logical</td>
<td>B=AND(A₁,A₂,...,Aₙ)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AND. Variable number</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>of arguments. n=2 or 3 or...or 63.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logical sum:</td>
<td>A=OR(A₁,A₂,...,Aₙ)</td>
<td>any type</td>
<td>no mode</td>
</tr>
<tr>
<td>Bit-by-bit logical</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OR. Variable number</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>of arguments. n=2 or 3 or...or 63.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exclusive OR:</td>
<td>A=XOR(A₁,A₂,...,Aₙ)</td>
<td>any type</td>
<td>no mode</td>
</tr>
<tr>
<td>Bit-by-bit exclusive</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OR. Variable number</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>of arguments. n=2 or 3 or...or 63.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Complement:</td>
<td>B=COMPL(A)</td>
<td>any type</td>
<td>no mode</td>
</tr>
<tr>
<td>Bit-by-bit Boolean</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>complement of A (i.e., not A)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Shifting</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shift:</td>
<td>B=SHIFT(A,I)</td>
<td>A - any type</td>
<td>no mode</td>
</tr>
<tr>
<td>Shift the number in</td>
<td></td>
<td>I - integer</td>
<td></td>
</tr>
<tr>
<td>A,I bit positions:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>left circular</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>if I &gt; 0; right with</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sign extension</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>if I &lt; 0 , 0&lt;</td>
<td>I</td>
<td>≤60</td>
<td></td>
</tr>
</tbody>
</table>
### Table 7-2. FORTRAN Library Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
<th>Parameter Mode</th>
<th>Result Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACOS(X)</td>
<td>Arcosine</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>ALOG(X)</td>
<td>Natural log of X</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>ALOG10(X)</td>
<td>Log to the base 10 of X</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>ASIN(X)</td>
<td>Arcsine</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>ATAN(X)</td>
<td>Arctangent X radians</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>ATAN2(X₁, X₂)</td>
<td>Arctangent X₁/X₂</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>CABS(C)</td>
<td>Absolute value</td>
<td>Complex</td>
<td>Real</td>
</tr>
<tr>
<td>CBAIEX(C, I)†</td>
<td>C**I</td>
<td>Complex (to integer)</td>
<td>Complex</td>
</tr>
<tr>
<td>CCOS(C)</td>
<td>Complex cosine</td>
<td>Complex</td>
<td>Complex</td>
</tr>
<tr>
<td>CEXP(C)</td>
<td>Complex exponential</td>
<td>Complex</td>
<td>Complex</td>
</tr>
<tr>
<td>CLOG(C)</td>
<td>Complex log function</td>
<td>Complex</td>
<td>Complex</td>
</tr>
<tr>
<td>COS(X)</td>
<td>Cosine X radians</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>CSIN(C)</td>
<td>Complex sine</td>
<td>Complex</td>
<td>Complex</td>
</tr>
<tr>
<td>CSQRT(C)</td>
<td>Complex square root</td>
<td>Complex</td>
<td>Complex</td>
</tr>
<tr>
<td>CUBRT(Y)</td>
<td>Cube root</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>DATAN(D)</td>
<td>Double arctangent</td>
<td>Double</td>
<td>Double</td>
</tr>
<tr>
<td>DATAN2(D₁, D₂)</td>
<td>Double arctangent D₁/D₂</td>
<td>Double</td>
<td>Double</td>
</tr>
<tr>
<td>DBADEX(D₁, D₂)†</td>
<td>D₁**D₂</td>
<td>Double (to double)</td>
<td>Double</td>
</tr>
<tr>
<td>DBAIEX(D, I)†</td>
<td>D**I</td>
<td>Double (to integer)</td>
<td>Double</td>
</tr>
<tr>
<td>DBAREX(D, X)†</td>
<td>D**X</td>
<td>Double (to real)</td>
<td>Double</td>
</tr>
<tr>
<td>DCOS(D)</td>
<td>Double cosine</td>
<td>Double</td>
<td>Double</td>
</tr>
<tr>
<td>DEXP(D)</td>
<td>Double exponential function (e^D)</td>
<td>Double</td>
<td>Double</td>
</tr>
<tr>
<td>DLOG(D)</td>
<td>Natural log of D (base e)</td>
<td>Double</td>
<td>Double</td>
</tr>
</tbody>
</table>

† These routines must be called with a ** operator in FORTRAN. They may not be used as functions. Use of the function name is only allowed in assembly language. These routines must save and restore A0 as well as B-registers used. Other library functions need only save the B-registers.
<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
<th>Parameter Mode</th>
<th>Result Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>DLOG10(D)</td>
<td>Log to base 10 of D</td>
<td>Double</td>
<td>Double</td>
</tr>
<tr>
<td>DMAX1(D₁,D₂,...,Dₙ)</td>
<td>Maximum of n arguments</td>
<td>Double</td>
<td>Double</td>
</tr>
<tr>
<td>DMIN1(D₁,D₂,...,Dₙ)</td>
<td>Minimum of n arguments</td>
<td>Double</td>
<td>Double</td>
</tr>
<tr>
<td>DMOD(D₁,D₂)</td>
<td>D₁ modulo D₂</td>
<td>Double</td>
<td>Double</td>
</tr>
<tr>
<td>DSIGN(D,D₂)</td>
<td>Sign of D₂ times</td>
<td>D₁</td>
<td></td>
</tr>
<tr>
<td>DSIN(D)</td>
<td>Sine of double-precision argument</td>
<td>Double</td>
<td>Double</td>
</tr>
<tr>
<td>DSQRT(D)</td>
<td>Square root of double</td>
<td>Double</td>
<td>Double</td>
</tr>
<tr>
<td>EXP(X)</td>
<td>e to xth power (e^x)</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>IBAIEX(I₁,I₂)†</td>
<td>I₁**I₂</td>
<td>Integer</td>
<td>Integer</td>
</tr>
<tr>
<td>IDINT(D)</td>
<td>Convert double-precision argument to integral floating-point number (rounded)</td>
<td>Double</td>
<td>Real</td>
</tr>
<tr>
<td>RANF(X)</td>
<td>Random number generator; uniform distribution 0.&lt; RANF(X) &lt; 1.</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>RBAIEX(X₁,I)†</td>
<td>X₁**I</td>
<td>Real (to integer)</td>
<td>Real</td>
</tr>
<tr>
<td>RBAREX(X₁,X₂)†</td>
<td>X₁**X₂</td>
<td>Real (to real)</td>
<td>Real</td>
</tr>
<tr>
<td>SIN(X)</td>
<td>Sine X radians</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>SQRT(X)</td>
<td>Square root of X</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>TAN(X)</td>
<td>Tangent X radians</td>
<td>Real</td>
<td>Real</td>
</tr>
<tr>
<td>TANH(X)</td>
<td>Hyperbolic tangent X radians</td>
<td>Real</td>
<td>Real</td>
</tr>
</tbody>
</table>

† These routines must be called with a ** operator in FORTRAN. They may not be used as functions. Use of the function name is only allowed in assembly language. These routines must save and restore AO as well as B-registers used. Other library functions need only save the B-registers.
A number of miscellaneous subroutines are also included in the library and need not be submitted with the user deck. These are not the standard routines associated with STANDARD FORTRAN programs, and may not be available at other installations. These are essentially NCAR routines. Included in Table 7-3 is a list of the routines.

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Entry Points</th>
<th>Subroutine</th>
<th>Entry Points</th>
<th>Subroutine</th>
<th>Entry Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACGOER</td>
<td>ACGOER</td>
<td>INQUIRE</td>
<td>INQUIRE</td>
<td>PWRY</td>
<td>PWRY</td>
</tr>
<tr>
<td>BACKSPACE</td>
<td>BACKSP, IFENDF, REWINM</td>
<td>INPUTC</td>
<td>INPUTC</td>
<td>Q8QRSRD, NUMP</td>
<td></td>
</tr>
<tr>
<td>BRANRD</td>
<td>BRANRD, BRANCK, BRANST, BRANRL</td>
<td>INPUTS</td>
<td>INPUTS</td>
<td>RDBK</td>
<td>RDBK</td>
</tr>
<tr>
<td>BUFFEI</td>
<td>BUFFEI, BUFFEO, IOCHECK, LENGTHF</td>
<td>IOB</td>
<td>INPUTB, OUTPTB</td>
<td>RPTIN/RPTOUT</td>
<td></td>
</tr>
<tr>
<td>BUFRD</td>
<td>BUFINT, BUFWT, BUFRD, BUFCL</td>
<td>IOPROC</td>
<td>BSTAPE, RDTAPE, WRTAPE, IOWAIT</td>
<td>SAVEF</td>
<td></td>
</tr>
<tr>
<td>BUFTIO</td>
<td>Q8QRB, Q8QWTB</td>
<td>JOBID</td>
<td>JOBID</td>
<td>SETPASS</td>
<td></td>
</tr>
<tr>
<td>CLOSE</td>
<td>CLOSE</td>
<td>KODER</td>
<td>KODER</td>
<td>SKIPFILE</td>
<td></td>
</tr>
<tr>
<td>CLD</td>
<td>CLOCKF, DATEF</td>
<td>LCMQREQ</td>
<td>LCMQREQ, LCMRD, LCMRT</td>
<td>SORT</td>
<td></td>
</tr>
<tr>
<td>CURVED</td>
<td>CURVED</td>
<td>MACHINE</td>
<td>MACHINE</td>
<td>SSW</td>
<td></td>
</tr>
<tr>
<td>DASHD</td>
<td>DASHD</td>
<td>NEWVOL</td>
<td>NEWVOL</td>
<td>SYSRCL</td>
<td></td>
</tr>
<tr>
<td>DEBUG</td>
<td>DEBUG</td>
<td>NITG</td>
<td>NITG</td>
<td>TIMEF</td>
<td></td>
</tr>
<tr>
<td>DUMP</td>
<td>DUMP</td>
<td>OUTPTC</td>
<td>OUTPTC</td>
<td>ULIBER</td>
<td></td>
</tr>
<tr>
<td>ENCD</td>
<td>ENCD</td>
<td>OUTPTS</td>
<td>OUTPTS</td>
<td>UNLCRD</td>
<td></td>
</tr>
<tr>
<td>ENDFIL</td>
<td>ENDFIL</td>
<td>OPEN</td>
<td>OPEN</td>
<td>UNLCRD, UNLCWT, UNLOCK</td>
<td></td>
</tr>
<tr>
<td>EXIT</td>
<td>END, EXIT, STOP</td>
<td>OVERLAY</td>
<td>OVERLAY</td>
<td>UNLOAD</td>
<td></td>
</tr>
<tr>
<td>FLRGCK</td>
<td>FLRGCK</td>
<td>PAUSE</td>
<td>PAUSE</td>
<td>VARGCK</td>
<td></td>
</tr>
<tr>
<td>GBYTE</td>
<td>GBYTE, GBYTES, SBYTE, SBYTES</td>
<td>PDUMP</td>
<td>PDUMP</td>
<td>VOLCPY</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PSYMD</td>
<td>PSYMD</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
BLOCK DATA

A BLOCK DATA subprogram is a separate, nonexecutable subprogram used for initialization of data. The first statement of a BLOCK DATA subprogram must have the following form:

```
BLOCK DATA [sub]
```

sub is a symbolic identifier consisting of from one to seven alphanumeric characters and starting with a letter.

Rules

- Only data statements and specification statements may be included. No executable statements are allowed.
- There may be more than one BLOCK DATA subprogram in a deck, each having a unique alphanumeric name.
- There may be only one unnamed or blank BLOCK DATA subprogram.
- If the deck contains OVERLAYS and SEGMENTS, use a named BLOCK DATA subprogram, e.g., BLOCK DATA BLKDATA.
- DATA may not be entered in blank COMMON with a DATA statement.
- All specification statements for a COMMON block must be included, such as TYPE, DIMENSION, and EQUIVALENCE statements, even though DATA statements do not appear for all variables in that block.
- DATA may be initialized in more than one COMMON block in any one BLOCK DATA subprogram.

Example

```
BLOCK DATA IRT
COMMON/XYZ/A(4)
DATA A/1.,2.,3.,4./
```
RETURN AND END STATEMENTS

RETURN

A subprogram may contain one or more RETURN statements to indicate the end of logic flow within the subprogram and return control to the calling program. The form is:

RETURN

In function subprograms, control returns to the statement containing the function reference. In subroutine subprograms, control returns to the next executable statement following the CALL. A RETURN statement in the main program causes the diagnostic RETURN NOT ALLOWED IN MAIN PROGRAM.

END

The main program and every subprogram must contain one END statement, and it must be the last statement of the program or subprogram. The END card in a main program generates a CALL EXIT. The END statement in a subprogram generates a RETURN.
ENTRY STATEMENT

The ENTRY statement provides alternate entry points to a function or subroutine subprogram. Its form is

ENTRY en

where en is a symbolic identifier consisting of one to seven alphanumeric characters and starting with a letter.

The ENTRY statement should appear directly ahead of the first statement to be executed when the subprogram is called by that entry name. Only one name may be given a particular entry point.

Example 1

SUBROUTINE T1(A,B)
  ...
ENTRY T2
  ...
ENTRY T3
  ...
END

In the example, arguments appear on the SUBROUTINE card, not on the ENTRY statement card. The ENTRY card just establishes the place to enter the subprogram.

In the calling program, the reference to the entry name is made just as if reference were being made to the FUNCTION or SUBROUTINE in which the ENTRY is embedded:

CALL T1(X,Y)
CALL T2(XX,YY)
CALL T3(W,Z)
Example 2

FUNCTION ARG(A,B)
ARG = A+B
RETURN
ENTRY SOLX
ARG = B-A
RETURN
ENTRY SOLY
ARG = A*B - A*B
RETURN
END

A reference to the function ARG in a calling program must include parameters. For the following example, the call would be

\[ R = \text{ARG}(\cos(X) \cdot 3., T) \]
\[ Z = \text{SOLX}(X, Y) + \text{SOLY}(X_1, Y_1) \]

In this example notice that the subprogram may be entered at SOLX and SOLY as alternate entry points to ARG, which is the function name entry. The code following the entry points must define the same variable ARG to be returned to the calling program. The entry points SOLX and SOLY do not specify arguments in the subprogram definition. The calling programs use the entry point name and the arguments to branch to the function subroutine or subprogram. The arguments appear in the call (note SOLX(X,Y) above).
Rules

- In the subprogram, the name of the entry point appears only in the ENTRY statement.

- Formal parameters do not appear with the ENTRY statement. The diagnostic is INCORRECT FORM FOR THE ENTRY STATEMENT.

- In a function subprogram, the value is returned in the same variable name used in the function definition, regardless of entry point.

- ENTRY may not appear within a DO loop. The diagnostic is AN ENTRY STATEMENT MAY NOT OCCUR INSIDE A DO LOOP.

- All declarative statements must appear ahead of all executable statements including the ENTRY statement.

- An ENTRY statement may not be labeled. The diagnostic is DO NOT LABEL AN ENTRY STATEMENT.

- Entry points must agree in type with the function name. The mode of the results will be incorrect, but no diagnostic is provided.

- Calling programs use the entry point name and the actual argument list, if any, to branch to the function subroutine or subprogram.

- The implementation described does not conform to the FORTRAN 77 specification for entry.
When the actual parameter list which calls a function or subroutine subprogram contains a function or subroutine name as an argument, that name must be declared in an EXTERNAL statement in the calling program. Its form is

```
EXTERNAL en1[,en2]...
```

where en₁ is the name of a function or subroutine. The EXTERNAL statement must precede the first executable statement of any program which calls a function or subroutine subprogram using the EXTERNAL name. When it is used, EXTERNAL always appears in the calling program. It may not be used with statement functions; if it is, a compiler diagnostic is given.

**Example**

```
PROGRAM TEST
EXTERNAL TNG,ST

R = F1(TNG,X)

U = F1(ST,X)

END

FUNCTION F1(F,X)

F1 = F(X) - 49.3641*X*X

END

FUNCTION TNG(Y)

TNG = SIN(Y)/COS(Y)

END

FUNCTION ST(S)

ST = S**2.4+COS(S)

END
```
EXTERNAL STATEMENT (continued)

All programmer generated subprograms may be used as externals in an actual parameter list. FORTRAN library functions may not be used as externals. If the names of FORTRAN library functions appear in an EXTERNAL statement, the diagnostic INTRINSIC OR LIBRARY FUNCTION CANNOT BE DECLARED EXTERNAL will be printed.
In many subprograms, especially those performing matrix manipulation, the programmer may wish to vary array dimensions each time a subprogram is called.

This is accomplished by specifying the array name and its dimensions as formal parameters in the FUNCTION or SUBROUTINE statement. The corresponding actual parameters specified in the calling program are values used by the calling program as dimensions. The maximum dimensions that any given array may assume are determined by dimensions in a DIMENSION, COMMON, or TYPE statement in the calling program at compile time.

The following rules must be adhered to or the program will fail in ways that are difficult to debug. Compiler diagnostics are not provided in all cases, so care must be taken to write correct code.

- The formal parameters representing the array dimensions must be simple integer variables. The array name must also be a formal parameter.

- The actual parameters representing the array dimensions may be unsigned integer constants or integer variables. A floating point or a zero argument may cause an infinite loop.

- If the total number of elements of a given array in the calling program is N, then the total number of elements of the corresponding array in the subprogram may not exceed N. The code may be overwritten in the calling program.
In the example, the array X and the variable dimensions K,L must appear as formal parameters in the SUBROUTINE statement. The actual parameters in the main program vary from CALL to CALL. The second call to MAT sends the array B and its dimensions 5 and 6 as actual parameters. All three must appear in the CALL.

The compiler does not check to see if the limits of the array established by the DIMENSION statement in the main program are exceeded.
All FORTRAN programs and subprograms are compiled as independent units. A SUBROUTINE or FUNCTION card must appear as the first card of a subprogram and the FORTRAN END card must be the last card of a program or subprogram. A PROGRAM card is optional in a main program. All statements between these two cards are assumed by the compiler to belong to one program or subprogram with a unique name.

Example

Sample Deck Arrangement:

Main program TEST

\{ 
\begin{align*}
\text{PROGRAM TEST} & \quad \text{(optional)} \\
\vdots \\
\text{CALL SUBA}(C) \\
\vdots \\
\text{END}
\end{align*}
\}

Subroutine SUBA

\{ 
\begin{align*}
\text{SUBROUTINE SUBA}(C) \\
\vdots \\
\text{END}
\end{align*}
\}

Function F1

\{ 
\begin{align*}
\text{FUNCTION F1}(T) \\
\vdots \\
\text{END}
\end{align*}
\}
VIII

INPUT/OUTPUT FORMATS

STANDARD SUBSCRIPTS
NONSTANDARD SUBSCRIPTS
ARRAY TRANSMISSION
CONVERSION SPECIFICATIONS
EDITING SPECIFICATIONS

s FORMAT (spec₁,...,j(specₘ,...),specₙ,...)
INTRODUCTION

The FORMAT statement is used in conjunction with the input/output (I/O) statement in the transmission of data. The specification for the conversion to or from BCD is controlled by the FORMAT statement. The I/O statement specifies the I/O device and process—READ, WRITE, etc.—and the list of data to be moved. The FORMAT statement specifies the form in which the data is to be moved. In binary input/output statements no FORMAT statement is used, as the data is transmitted in binary without conversion.
8.2
Input/Output Formats

INPUT/OUTPUT
LIST

The list portion of an I/O statement indicates the data items and the order, from left to right, of their transmission. List items may be array names, simple or subscripted variables, or an I/O statement implied DO loop. List items are separated by commas, and their order must correspond to the FORMAT specification referred to by the I/O statement.

Execution of the I/O statement is begun and terminated by the list. External records are always read or written until the list is satisfied, repeating the format scan as required.

STANDARD
SUBSCRIPTS

Subscripts in an I/O list may be standard or nonstandard index functions. Standard index functions are of the form:

\[
\begin{align*}
  (c^\times I\pm d) & \quad (I\pm d) \\
  (I^\times c\pm d) & \quad (c^\times I) \\
  (d+I^\times c) & \quad (I) \\
  (d+c^\times I) & \quad (c) \\
  (c+I) & \quad
\end{align*}
\]

where \( c \) and \( d \) are unsigned integer constants, and \( I \) is a simple integer variable, previously defined, or defined within the I/O statement DO loop. The following are examples of FORTRAN READ statements:

```fortran
READ 100, A, B, C, D
READ 100, B(3,4), A(I,J,7), H
READ 101, J, A(J), L, B(I,J)
READ 102, A(5*I+2, 5*L-3, 5*K), C, D, (I+7)
```

The following examples show standard subscripts within an I/O statement DO loop:

```fortran
READ 100, A, B, C, D(I), I=1,10), E(5,7), F(J), G(I), H(I), I=2,6,2)
READ 100, I, J, K, ((A(II+1,JJ+2, KK+3), II=1,I), JJ=1,J), KK=1,K)
READ 100, ((A(I,J), I=2,10,2), B(J,L), J=1,5), E, F, G(L+5,M-7)
READ 100, C(J), C(J+2), C(J+4), J=1,4)
READ 100, C(J), C(J+2), C(J+4), J=1,2)
```
8.3
Input/Output Formats

NONSTANDARD
SUBSCRIPTS

Nonstandard subscripts are any arithmetic expression not defined above as standard. The expression is evaluated, and the result is truncated and used as the index.

```
WRITE (6,100) C(X+Y),C(T)
WRITE (6,100) C(COS(A)),C(MAX0(1,3)+1)
```

Limitations

- Nonstandard subscripts may not be used within an I/O statement implied DO loop. The diagnostic will be IMPROPER SUBSCRIPT IN AN INPUT OR OUTPUT STATEMENT.
- Subscripted subscripts are never allowed.
- More subscripts on a variable than specified in the DIMENSION statement for that variable will cause the diagnostic TOO MANY SUBSCRIPT INDICES.
- The DO variable in an I/O list DO may not use the same DO variable as a DO loop around the I/O statement. The diagnostic A DO VARIABLE IS USED IN AN OUTER DO LOOP appears.

ARRAY
TRANSMISSION

Part or all of an array can be represented for transmission as a single I/O list item by using DO loop notation in the WRITE statement. The general form of the implied DO loop is:

```
(((A(I,J,K),I=m1,m2[,m3]), J=n1,n2[,n3]), K=p1,p2[,P3])
```

where \( m_1, n_1, \) and \( p_1 \) are unsigned integer constants or simple integer variables. If \( m_3, n_3, \) or \( p_3 \) is omitted, the number used is equal to 1. \( I, J, K \) are subscripts of \( A \).

All notation must be explicitly defined in the WRITE statement.
During execution, each subscript (index variable) is set to the initial index value: \( I = m_1, J = n_1, K = p_1 \). The first index variable defined in the list is incremented first, following the same rules as DO loop execution. When the first index variable reaches the maximum value, it is reset, the next index variable to the right is incremented, and the process is repeated until the last index variable has been incremented. If \( m_1 > m_2 \) initially, one card is read.

**Rules**

- An array name which appears without subscripts in an I/O list causes transmission of the entire array. The first index or subscript of the array varies most rapidly.

- A DO loop can be used to transmit a simple variable more than one time. For example, the list item \((A(K),B,K=1,5)\) causes the transmission of variable B five times. However, in the case \((B,(A(K),K=1,5))\), B is transmitted only once.

- A list of the form \( K,(A(I),I=1,K) \) is permitted, and on input the input value of K is used as the value of K in the DO loop.

**Examples**

READ 100,(A(I),I=1,10)  
READ 100,((A(JV,JX),JV=2,20,2),JX=1,30)  
READ 100,(BETA(3*JON+7),JON=JONA,JONB,JONC)  
READ 100,(((IMLST(I,J+1,K-2),I=1,125),J=2,N),K=IVAR,IVMAX,4)  
READ 100,(A(I),B(I),I=1,10)

Nested DO loop list items are specified in the order of their indexing.
8.5
Input/Output Formats

READ 100,((((((A(I,J,K),B(I,L),C(J,N),I=1,10),J=1,5),K=1,8),
L=1,15),N=2,7)

Data are transmitted in the following sequence:

A(1,1,1),B(1,1),C(1,2),A(2,1,1),B(2,1),C(1,2)... 
...A(10,1,1),B(10,1),C(1,2), A(1,2,1),B(1,1),C(2,2)... 
...A(10,2,1),B(10,1),C(2,2)...A(10,5,1),B(10,1),C(5,2)... 
...A(10,5,8),B(10,1),C(5,2)...A(10,5,8),B(10,15),C(5,2)... 
...A(10,5,8),B(10,15),C(5,7)

Although the above list is highly redundant, the items illustrate DO loop execution. The following list will transmit the array E(3,3) by columns:

READ 100,((E(I,J),I=1,3),J=1,3)

The following list will transmit the array E(3,3) by rows:

READ 100,((E(I,J),J=1,3),I=1,3)

The statements

DIMENSION ARRAY (3,4,7)
READ 100, ARRAY

are equivalent to the statements

DIMENSION ARRAY (3,4,7)
READ 100,(((ARRAY(I,J,K),I=1,3),J=1,4),K=1,7)

An I/O statement loop and a DO statement loop are similar, but not equivalent.

The I/O statement loop

READ 100,(A(I),I=1,3)

is the same as

READ 100,A(1),A(2),A(3)
The DO loop

```
DO 2 I=1,3
  2 READ 100,A,(I)
```

is the same as

```
READ 100,A(1)
READ 100,A(2)
READ 100,A(3)
```

The same elements of A are output, but in an I/O loop only one read is generated, whereas in the DO loop three reads are performed.
The formatted I/O statements require a FORMAT declaration which contains the conversion and editing information relating to the internal/external structure of the corresponding I/O list items. A FORMAT declaration has the following form:

\[ s \text{ FORMAT (spec}_1,\ldots,\text{j(spec}_m,\ldots),\text{spec}_n,\ldots) \]

\( \text{spec}_i \) is a FORMAT specification, and \( \text{j}_k \) is an optional repetition factor which must be an unsigned integer constant. If \( \text{j}_k \) is omitted, the repeat will be continued to the end of the list. The FORMAT declaration is nonexecutable and may appear anywhere in the program. It is used by the I/O statement to convert internal/external structures. All FORMAT declarations must have a statement label in columns 1-5, as shown by \( s \). The maximum number of FORMAT statements in any program unit is 64. (e.g. program, subroutine, or function subprogram.)
The data items in an I/O list are converted from external to internal or from internal to external representation according to FORMAT conversion specifications. Each item in the list must have a FORMAT specification associated with it. FORMAT specifications may also contain editing codes.

**FORTRAN Conversion Specifications**

- **Ew.d** Single-precision floating point with exponent
- **Fw.d** Single-precision floating point without exponent
- **Gw.d** General floating point
- **Dw.d** Double-precision floating point with exponent
- **Iw** Decimal integer
- **Ow** Octal integer
- **Aw** Alphanumeric
- **Rw** Alphanumeric
- **Lw** Logical
- **nP** Scaling factor

**FORTRAN Editing Specifications**

- **wX** Intraline spacing
- **wH** Heading and labeling
- **/** Begin new record (n/, advances n records)
- ***...* Alphanumeric string included
- **'*...' Alphanumeric string included

Both w and d are unsigned integer constants: w specifies the field width in number of character positions in the external record, and d specifies the number of digits to the right of the decimal within the field.

Star (*) and apostrophe (') are delimiters for an alphanumeric string to be included in the print line.

*Note:* Complex data items are converted on I/O according to a pair of consecutive Ew.d or Fw.d specifications.
Each item in the I/O list must have a FORMAT specification associated with it. One FORMAT specification will be used repeatedly for more than one item if only one FORMAT specification appears.

The E specification converts the number in the input field to a real number and stores it in the appropriate location in storage.

The total number of characters in the input field is specified by w. The field is scanned from left to right; blanks within the field are interpreted as zeros.

The integer subfield begins with a sign (+ or -) or a digit and may contain a string of digits. The integer field is terminated by a decimal point, a D, an E, a + or -, or the end of the input field.

The fraction subfield which begins with a decimal point may contain a string of digits. The field is terminated by a D, an E, a + or -, or the end of the input field.

The exponent subfield may begin with a D, an E, or a + or -. When it begins with D or E, the + is optional between E, or D and the string of digits of the subfield. The value of the string of digits in the exponent subfield must be in the range $-293$ to $+322$ in order to be printed. A number as small as $10^{-295}$ may be held in memory even though on output it prints as zero (see Constants in Chapter 2).
### Ew.d INPUT (continued)

#### Permissible Subfield Combinations

<table>
<thead>
<tr>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1.6327E-04</td>
<td>integer fraction exponent</td>
</tr>
<tr>
<td>-32.7216</td>
<td>integer fraction</td>
</tr>
<tr>
<td>+328+5</td>
<td>integer exponent</td>
</tr>
<tr>
<td>.629+5</td>
<td>fraction exponent</td>
</tr>
<tr>
<td>+136</td>
<td>integer only</td>
</tr>
<tr>
<td>136</td>
<td>integer only</td>
</tr>
<tr>
<td>.07628431</td>
<td>fraction only</td>
</tr>
<tr>
<td>E-06 (interpreted as zero)</td>
<td>exponent only</td>
</tr>
</tbody>
</table>

#### Rules

- In the Ew.d specification, d acts as a negative power of ten scaling factor when an external decimal point is not present. The internal representation of the input quantity is

  \[(\text{integer subfield}) \times 10^{-d} \times 10^\text{(exponent subfield)}\]

  For example, if the specification is E7.8, the input quantity 3267+05 is converted and stored as

  \[3267 \times 10^{-8} \times 10^5 = 3.267.\]

  The input quantity \[|3210000|\] would be stored as .0321.

- If a decimal point occurs in the input field, the decimal point overrides d. The input quantity 3.67294+5 read by an E9.d specification is always stored as \[3.67294 \times 10^5\], regardless of the value of d.

- When d does not appear, it is assumed to be zero (E9,E15).

- The field length specified by w in Ew.d should always be the same as the length of the input field containing the input number. When it is not, incorrect numbers may be read, converted, and stored. No diagnostic is provided in these cases.
8.11
Input/Output Formats

- An exponent greater than 322 or less than -295 will not be accepted on input. The diagnostic is EXPONENT TOO LARGE ON DATA INPUT.

- ILLEGAL DATA ENCOUNTERED is printed for input fields, such as letters, that cannot be interpreted.

<table>
<thead>
<tr>
<th>Ew.d Input Field</th>
<th>Ew.d Format Specification</th>
<th>Converted Value</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>+143.26E-03</td>
<td>E11.2</td>
<td>.14326</td>
<td>All subfields present</td>
</tr>
<tr>
<td>-12.437629E+1</td>
<td>E13.6</td>
<td>-124.37629</td>
<td>All subfields present</td>
</tr>
<tr>
<td>8936E+004</td>
<td>E9.10</td>
<td>.008936</td>
<td>No fraction subfield; input number converted as $8936 \times 10^{-1+4}$</td>
</tr>
<tr>
<td>327.625</td>
<td>E7.3</td>
<td>327.625</td>
<td>No exponent subfield</td>
</tr>
<tr>
<td>4.376</td>
<td>E5</td>
<td>4.376</td>
<td>No d in specification</td>
</tr>
<tr>
<td>-.0003627+5</td>
<td>E11.7</td>
<td>-36.27</td>
<td>Integer subfield contains minus only</td>
</tr>
<tr>
<td>-.0003627E5</td>
<td>E11.7</td>
<td>-36.27</td>
<td>Integer subfield contains minus only</td>
</tr>
<tr>
<td>blanks</td>
<td>Bw.d</td>
<td>-0.</td>
<td>All subfields empty</td>
</tr>
<tr>
<td>1E1</td>
<td>E3.0</td>
<td>10.</td>
<td>No fraction subfield; input number converted as $1 \times 10^1$</td>
</tr>
<tr>
<td>E+06</td>
<td>E10.6</td>
<td>0.</td>
<td>No integer or fraction subfield; zero stored regardless of exponent field constants</td>
</tr>
<tr>
<td>l.bEbl</td>
<td>E6.3</td>
<td>10.</td>
<td>Blanks are interpreted as zeros</td>
</tr>
</tbody>
</table>
Input/Output Formats

**Ew.d OUTPUT**

E conversion is used to convert real numbers in storage to the BCD character form for output. The field occupies w positions in the output record; the corresponding real number appears right-justified in the field in one of the following forms:

\[
\text{Sa.a...aE} \pm \text{ee} \quad 0 < \text{ee} < 99 \\
\text{Sa.a...aE} \pm \text{ee} \quad 100 < \text{ee} < 308
\]

Field width = w

<table>
<thead>
<tr>
<th>Format Ew.d</th>
<th>S</th>
<th>X</th>
<th>.</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>E</th>
<th>+</th>
<th>X</th>
<th>X</th>
</tr>
</thead>
</table>
| Sign        | Leading digit always one space without decimal-d | Places after exponential-d | Exponent always four spaces on output. It has form +XXX for exponents > 99.

a.a...a are the most significant digits of the integer and fractional part, and eee are the digits in the exponent. S is the sign, blank for plus on output. If d is zero or blank, the decimal point and digits to the right of the decimal do not appear as shown above. Field w must be wide enough to contain the significant digits, signs, decimal point, E, and the exponent. Generally, \( w > d + 7 \). Positive numbers need not reserve a space for the sign of the number. If the field width is less than the number of places specified for the decimal field, a diagnostic is printed at execution time.

If the field is not wide enough to contain the output value, an asterisk is inserted in the high-order position of the field. If the field is longer than the output value, the quantity is right-justified with blanks in the excess positions to the left.
Examples

PRINT 10,A
10 FORMAT (1Hb,E10.3) or +67.32
A contains -67.32
Result: -6.732E+01 or b6.732E+01

PRINT 10,A
10 FORMAT (1Hb,E13.3) or +67.32
A contains -67.32
Result: bbb-6.732E+01 or bbbb6.732E+01

PRINT 10,A
10 FORMAT (1Hb,E9.3) Since A is positive, no provision is necessary for the sign
A contains +67.32.
Result: 6.732E+01

PRINT 10,A
10 FORMAT (1Hb,E10.4) A contains -67.32. No provision is made for the sign, thus an * indicates a bad format
A contains -67.32.
Result: *7320E+02

PRINT 100,A
100 FORMAT (1Hb,E10.2) Where A=+oo
Where A=+oo
(37777777777777777777B)
and is output with an E format
(17777777777777777777B)
and A is printed with an E format
Result: RRRRR
Result: IIIII
**Input/Output Formats**

**Fw.d INPUT**

This specification is a modification of Ew.d. The input field consists of an integer and a fraction subfield. An omitted subfield is assumed to be zero. All rules listed under Ew.d apply.

<table>
<thead>
<tr>
<th>Fw.d Input Field</th>
<th>Fw.d Format Specification</th>
<th>Converted Value</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>367.2593</td>
<td>F8.4</td>
<td>367.2593</td>
<td>Integer and fraction field</td>
</tr>
<tr>
<td>37925</td>
<td>F5.7</td>
<td>.0037925</td>
<td>No fraction subfield; input number converted as $37925 \times 10^{-7}$</td>
</tr>
<tr>
<td>-4.7366</td>
<td>F7</td>
<td>-4.7366</td>
<td>No d in specification, but decimal in input field is used</td>
</tr>
<tr>
<td>.62543</td>
<td>F6.5</td>
<td>.62543</td>
<td>No integer subfield</td>
</tr>
<tr>
<td>.62543</td>
<td>F6.2</td>
<td>.62543</td>
<td>Decimal point in input field overrides d of specification</td>
</tr>
<tr>
<td>+144.15E-03</td>
<td>F11.2</td>
<td>0.14415</td>
<td>Exponents are legitimate in F input and may have P-scaling</td>
</tr>
</tbody>
</table>

**Fw.d OUTPUT**

The field occupies $w$ positions in the output record; the corresponding list item must be a floating point quantity, and it appears on output as a decimal number, right-justified in the field $w$, as

```
S X X X X X X X X X . X X X X X X ...
```

$w$
S is the sign. X represents the significant digits of the number to be printed or output. The number of decimal places to the right of the decimal is specified by d. If d is zero or omitted, the decimal point and digits to the right do not appear. If the number is positive, the plus sign is suppressed.

If the field is too short to accommodate the number, one asterisk appears in the high-order position of the output field.

If the field w is longer than required to accommodate the number, the number is right-justified with blanks occupying the excess field positions to the left.

Examples

PRINT 10,A
10 FORMAT (1X,F7.3)
Result: b32.694

PRINT 11,A
11 FORMAT (1X,F10.3)
Result: bbbb32.694

PRINT 12,A
12 FORMAT (1X,F6.3)
A contains -32.694;
no provision for minus sign
Result: *2.694

PRINT 13,A,A
13 FORMAT (1X,F4.3,F6.3)
A contains .32694
Result: .327b0.327

PRINT 14,A,B,C,D
14 FORMAT (1X,F10.4)
A contains +0
B contains -0
C contains +.000000001
D contains -.000000001
Result: bbbbo.0000
bbbo-0.0000
bbbbbo.0000
bbbb-0.0000
Input/Output Formats

**Gw.d INPUT**

Gw.d input specification is the same as Ew.d input specification.

**Gw.d OUTPUT**

The G conversion is a general conversion used for real data output. The floating point quantity is output as an F number or an E number depending on the magnitude of the number itself. W designates the field length as before. d, however, is the number of significant digits of the value to be represented unless the number reverts to a standard E format where d is the number of digits following the decimal point.

In the following table N is the magnitude of the number being converted. Four blanks are inserted within the field, right-justified if F format is used within a G specification.

<table>
<thead>
<tr>
<th>Range of Number</th>
<th>Type of Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>.1 ≤ N &lt; 1</td>
<td>F(w-4).d,4X</td>
</tr>
<tr>
<td>1 ≤ N &lt; 10</td>
<td>F(w-4.).(d-1),4X</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>10^{d-2} ≤ N &lt; 10^{d-1}</td>
<td>F(w-4).1,4X</td>
</tr>
<tr>
<td>10^{d-1} ≤ N &lt; 10^d</td>
<td>F(w-4).0,4X</td>
</tr>
<tr>
<td>Otherwise</td>
<td>Ew.d</td>
</tr>
</tbody>
</table>
### 8.17  
**Input/Output Formats**

<table>
<thead>
<tr>
<th>Internal Number</th>
<th>Gw.d Format Specification</th>
<th>Printer Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>23.146</td>
<td>G10.3</td>
<td>23.1</td>
</tr>
<tr>
<td>.023</td>
<td>G10.3†</td>
<td>2.300E-02</td>
</tr>
<tr>
<td>123456789.2</td>
<td>G14.4†</td>
<td>1.2346E+08</td>
</tr>
<tr>
<td>123.456</td>
<td>G14.4</td>
<td>123.5</td>
</tr>
<tr>
<td>.001</td>
<td>G10.2†</td>
<td>1.00E-03</td>
</tr>
<tr>
<td>1.234E-1</td>
<td>G15.3</td>
<td>.123</td>
</tr>
<tr>
<td>1.23E+5</td>
<td>G15.6</td>
<td>123000</td>
</tr>
<tr>
<td>234.567</td>
<td>G10.5</td>
<td>234.57</td>
</tr>
<tr>
<td>&quot;</td>
<td>G10.4</td>
<td>234.6</td>
</tr>
<tr>
<td>&quot;</td>
<td>G10.3</td>
<td>235</td>
</tr>
<tr>
<td>&quot;</td>
<td>G10.2†</td>
<td>2.35E+02</td>
</tr>
<tr>
<td>&quot;</td>
<td>G10.1†</td>
<td>2.3E+02</td>
</tr>
</tbody>
</table>

† In these G specifications, the internal number does not conform to N as defined above so that d may *not* be used as the number of significant digits. Therefore, the format reverted to Ew.d.
D conversion corresponds to Ew.d input. In double precision, 27 significant digits are permitted in the combined integer-fraction field. D or E is acceptable as the beginning of an exponent subfield on input. (Note: E is not acceptable in a replacement statement to define a double-precision quantity.)

Example

DOUBLE Z,Y,X
READ 1,Z,Y,X
1 FORMAT(D18.11,D15,D17.4)

Input Card:

```
-6.3167529843E-03  +2.718926453147 6293477538869D-09
```

Dw.d OUTPUT

The field occupies w positions of the output record. The corresponding list item, which must be a double-precision quantity, appears as a decimal number, right-justified in the field w, as one of the following:

\[
SXX...X.X....X+eee \quad 100 \leq eee \leq 293
\]

\[
SX...X.X....XD+ee \quad 0 \leq ee \leq 99
\]

S indicates the sign (blank for plus). D conversion corresponds to Ew.d output, but uses a double-precision (two-word) quantity. Single-precision information must not be output using the D specification.
The field is $w$ characters in length, and the corresponding list item may be a decimal integer constant or a logical constant.

The input field $w$ which consists of an integer subfield may contain only the characters $+, -$, the digits 0 through 9, or blank. When a sign appears, it must precede the first digit in the field. Blanks are interpreted as zeros. Where the whole field is blank, the value is $-0$. The value is stored right-justified in the specified variable.

Example

```
READ 10,I,J,K,L,M,N,N1
10 FORMAT (I3,I7,I2,I3,I2,2I4)
```

Input Card:

```
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>139</td>
<td>bb-15bb</td>
<td>18</td>
<td>bb7</td>
<td>b3</td>
<td>bb4</td>
</tr>
</tbody>
</table>

```

In Storage:

- I contains 139
- J contains -1500
- K contains 18
- L contains 7
- M contains 3
- N contains 104
- NL contains -0
Iw OUTPUT

I specification is used to output decimal integer values. The output quantity occupies w output record positions; it appears right-justified in the field w as

\[ S | \overline{XXXX} \]

S is the sign. XXXX are the decimal digits (maximum 18) of the integer. If the integer is positive, the plus sign is suppressed.

If the field w is larger than required, the output quantity is right-justified with blanks occupying excess positions to the left. If the field is too short, characters are printed starting from the right, with an asterisk in the leftmost position.

Example

```
PRINT 10,I,J,K
10 FORMAT (1X,I8,I10,I5)
```

I contains -3762
J contains +4762937
K contains +13

Result: bbb-3762bbb4762937bbbl3
Octal integer values are converted under 0 specification. The field is w characters in length, and the corresponding list item must be an integer variable.

The input field w consists of an integer subfield only (maximum of 20 octal digits). The only characters that may appear in the field are +, -, blank, and 0 through 7. Only one sign is permitted and, if present, must precede the first digit in the field. Blanks are interpreted as zeros.

Example

```plaintext
TYPE INTEGER,P,Q,R
READ 10,P,Q,R
10 FORMAT (010,012,02)
```

Input Card:
```
          -1
3737373737 666b6644b444 -0
```

In Storage:
```
P: 00000000003737373737
Q: 00000000666666440444
R: 77777777777777777777
```

A negative number is represented in one's complement form.

A negative octal number is represented internally in 20-digit seven's complement form obtained by subtracting each digit of the octal number from seven. For example, if -703 is an input quantity, its internal representation is 7777777777777777074. That is:

```
77777777777777777777
- 0000000000000000703
7777777777777777074
```
Ow OUTPUT

0 specification is used to output internal values as an octal representation of the binary word in memory. The output quantity occupies w output record positions, and it appears right-justified in the field as

XXXXXX

The X are octal digits. If w is less than 20, the right-most w digits appear, with an asterisk in the first digit to indicate that there are nonzero numbers left out. If w is greater than 20, the number is right-justified in the field with blanks to the left of the output quantity. A negative number is output in its ones-complement internal form. If w is 20, 20 'gits appear.

Example

A=25252525676767676767B
PRINT 100,A
100 FORMAT(1X,020)
PRINT 101,A,A
101 FORMAT(1X,018/1X,025)

Result: 25252525676767676767
*52525256767676767
25252525676767676767
This specification accepts characters in the FORTRAN character set, including blank. The internal representation is display code; the field width is \( w \) characters.

If \( w \) exceeds 10, the input quantity is the rightmost 10 characters in the field. If \( w \) is 10 or less, the input quantity goes to the designated storage location as a left-justified BCD word; remaining spaces are blank filled.

**Example**

```plaintext
READ 10,Q,P,O
10 FORMAT (A8,A8,A4)
```

Input Card:

```
S.S---8---8--IS 14243055150516245555
LUX MENT IS LUX 0 RBIS
```

In Storage:

```
Octal Word:
Q: LUXMENThbb Q: 14243055150516245555
P: ISbLUXb0bb P: 11235514253055175555
O: RBISb00000 O: 22021123555555555555
```

A conversion is used to output alphanumerical characters. If \( w \) is 10 or more, the output quantity appears right-justified in the output field, blank filled to the left. If \( w \) is less than 10, the output quantity represents the leftmost \( w \) characters.
8.24
Input/Output Formats

**Rw INPUT**

This specification is the same as the Aw input unless w is less than 10, in which case the input quantity goes to the designated storage location as a right-justified binary zero filled word.

**Example**

```
READ 10,Q,P,O
10 FORMAT (R8,R8,R4)
```

Input Card:

```
0
```

In Storage:

```
Octal Word:

<table>
<thead>
<tr>
<th>0:</th>
<th>100: LUXMENT</th>
<th>Q:</th>
<th>0001425305515051624</th>
</tr>
</thead>
<tbody>
<tr>
<td>P:</td>
<td>100: ISbLUXb0</td>
<td>P:</td>
<td>0001123551425305517</td>
</tr>
<tr>
<td>0:</td>
<td>100: RBIS</td>
<td>0:</td>
<td>000000000002220271123</td>
</tr>
</tbody>
</table>
```

**Rw OUTPUT**

This specification is the same as the Aw output unless w is less than 10, in which case the output quantity represents the rightmost characters in the word.

**Example**

```
PRINT 100,B,C
100 FORMAT (1H0,R5,R3)
```

Printed Output:

```
PRSTUXYZ
```

In Storage:

```
B:  KLMNOPRSTU
C:  LTTEMP3XYZ
```
This specification accepts logical variables for input. \( w \) is the width of the external field. This field must contain \( T \), \( F \), TRUE or FALSE, preceded and/or followed by optional blanks.

Output specifications are given for logical variables. \( w \) is the width of the field where \( T \) or \( F \) may be preceded by \( w-1 \) blanks.

**Example**

```
TYPE LOGICAL L,K
L=.TRUE.
K=.FALSE.
PRINT 100, L,K
100 FORMAT (1H0,2L5)
```

Result: bbbTbbbF

A logical constant in a replacement statement may be \( .T. \), \( .F. \), \( .TRUE. \), \( .FALSE. \). (See Logical Constants in Chapter 2.)
**SCALE FACTOR**

A scale factor may precede the D, E, and F conversion. The scale factor is:

\[
\text{External number} = \text{internal number} \times 10^n \text{ scale factor}
\]

The scale factor applies to \(Fw.d\) on both input and output and to \(Ew.d\) and \(Dw.d\) on output only. A scaled specification is written in FORTRAN as:

- \(nPDw.d\)
- \(nPEw.d\)
- \(nPFw.d\)
- \(nP\)

Where \(n\) is a signed integer constant.

**Rules**

- P preceded by blank is interpreted as 0P.
- The scale factor is assumed to be zero if no other value has been given; however, once a value has been given, it holds for all D, E, and F specifications following the scale factor within the same FORMAT declaration. To nullify this effect in subsequent D, E, and F specifications, a zero scale factor, 0P, must precede a D, E, or F specification. When a FORMAT is repeated from a single I/O statement because the list is not exhausted, the last scale factor continues to apply to specifications at the beginning unless scale factors are included for each specification. Use (0PF10.5,1PE15.6) rather than (F10.5,1PE15.6).
- Scale factors on D or E input specifications are ignored.
The scaling specification \( n_P \) may appear independent of a D, E, or F specification, but it holds for all D, E, and F specifications that follow within the same FORMAT statement unless changed by another \( n_P \). For example, the specification \((3P, 3I9, F10.2)\) is the same as the specification \((3I9, 3PF10.2)\) when using the same FORMAT for repeated scans.

### Fw.d SCALING

#### Input

The number in the input field is divided by \( 10^n \) and stored. For example, if the input quantity 314.1592 is read under the specification 2PF8.4, the internal number is 
\[
314.1592 \times 10^{-2} = 3.141592.
\]

#### Output

The number in the output field is the internal number multiplied by \( 10^n \). In the output representation, the decimal point is fixed; the number moves to the left or right, depending on whether the scale factor is plus or minus. For example, the internal number 3.1415926538 may be represented on output under scaled F specifications as follows:

<table>
<thead>
<tr>
<th>Fw.d Format Specification</th>
<th>Output Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>F13.6</td>
<td>3.141593</td>
</tr>
<tr>
<td>1PF13.6</td>
<td>31.415927</td>
</tr>
<tr>
<td>3PF13.6</td>
<td>3141.592654</td>
</tr>
<tr>
<td>-1PF13.6</td>
<td>.314159</td>
</tr>
</tbody>
</table>
8.28
Input/Output Formats

Ew.d OR Fw.d
SCALING

Output

The scale factor has the effect of shifting the output number left \( n \) places while reducing the exponent by \( n \). Using 3.1415926538, some output representations corresponding to scaled E specifications are:

<table>
<thead>
<tr>
<th>Format Specifications</th>
<th>Output Representations</th>
</tr>
</thead>
<tbody>
<tr>
<td>E20.2</td>
<td>3.14 E+00</td>
</tr>
<tr>
<td>1PE20.2</td>
<td>31.42 E-01</td>
</tr>
<tr>
<td>2PE20.2</td>
<td>314.16 E-02</td>
</tr>
<tr>
<td>3PE20.2</td>
<td>3141.59 E-03</td>
</tr>
<tr>
<td>4PE20.2</td>
<td>31415.93 E-04</td>
</tr>
<tr>
<td>5PE20.2</td>
<td>314159.27 E-05</td>
</tr>
<tr>
<td>-1PE20.2</td>
<td>0.31 E+01</td>
</tr>
</tbody>
</table>
EDITING SPECIFICATIONS

wx SPECIFICATION

This specification may be used to include w blanks in an output record or to slip w characters on an input record to permit spacing of I/O quantities. Blank is represented as IX. If w is not specified as in (F4.1,X,I5), it is interpreted as IX. 0X is not permitted.

Examples

INTEGER A
PRINT 10,A,B,C
10 FORMAT (12,6X,F6.2,6X,E12.5)

A contains 7
B contains 13.6
C contains 1462.37

Results: b7bbbbbbbl3.60bbbbbbbl.46237E+03

Note: The initial characters of a line on the printer is interpreted as a carriage control command.

READ 502, I, S, NAM
502 FORMAT (4X,I6,3X,F8,2X,A10)

Input Card:

< 2Cbbb9999EEbl. 216 58bXXbbbbbbMARY

In Storage:

I: 9999
S: 1.21658
NAM: MARY
The H specification may be used to read Hollerith characters into an existing H field within the FORMAT specification itself.

**Example 1**

```plaintext
READ 10
10 FORMAT (27Hbbbbbbbbbbbbbbbbbbbbbbbb)
```

Input Card

```
THIS IS A VARIABLE HEADING
```

After the READ, the FORMAT statement labeled 10 contains the alphanumeric information read from the input card; a subsequent reference to statement 10 in an output statement acts as follows:

```plaintext
PRINT 10
```

Result: bTHIS IS A VARIABLE HEADING

Note: The initial character, a blank, is interpreted by printer as a carriage control command.

**Example 2**

```plaintext
READ 11
11 FORMAT (* *)
```

Where the card is punched (bbVARIABLE), PRINT 11 produces the line bbVARIABLE, where the initial blank is for carriage initial on the printer.
This specification provides for the output of 6-bit characters, including blanks, in the form of comments, titles, and headings. \( w \) is an unsigned integer specifying the number of characters to the right of the \( H \) that are transmitted to the output record; \( w \) may specify a maximum of 136 characters. In the case of records sent to the printer, the first character for carriage control is excluded from the maximum count of 136. \( H \) denotes a Hollerith field. The comma following the \( H \) specification is optional.

**Examples**

Note: The initial characters in each print record are interpreted as carriage control by the printer.

```
PRINT 20
20 FORMAT (28HbBLANKSbCOUNTbINbANbHbFIELD.)
```

Result: BLANKS COUNT IN AN H FIELD.

```
PRINT 30,A
30 FORMAT (6HbLMAX=,F5.2)
```

Result: LMAX=bl.50

```
PRINT 504
504 FORMAT (*0 ASTERISKS MAY BE USED TO ENCLOSE A HOLLERITH FIELD ONLY IN A FORMAT STATEMENT*)
```

Result: ASTERISKS MAY BE USED TO ENCLOSE A HOLLERITH FIELD ONLY IN A FORMAT STATEMENT

```
TEMP = 32.
PRINT 504,TEMP
504 FORMAT (1H0*TEMPERATURE=*F3.0,* DEGREES*)
```

Result: TEMPERATURE= 32 DEGREES

```
PRINT 200,PE
200 FORMAT ('OFOR PE='F2,0)
```

Result: FOR PE=b2
The slash (/) signals the end of a record anywhere in the specifications list. It need not be separated from the other list elements by commas; consecutive slashes may appear in a list. During output, the slash is used to skip lines, cards, or tape records. The maximum number of printed characters per line is 137, 136 characters printed plus 1 initial character for carriage control. One line on the line printer is a record. \texttt{K(/)} results in \(k-1\) lines being taken on output.

During input, a slash (/) specifies that control passes to the next record or card. \texttt{K(/)} results in \(K-1\) cards being skipped on input. One card has a maximum of 80 characters.

**Example 1**

\begin{verbatim}
PRINT 10
10 FORMAT (16X,7HHEADING///6X,12HINPUT OUTPUT)
\end{verbatim}

Printout:

\begin{verbatim}
HEADING
\end{verbatim}  (line 1)
\begin{verbatim}
   INPUT OUTPUT
\end{verbatim}  (line 2)
\begin{verbatim}
   (line 3)
\end{verbatim}
\begin{verbatim}
   (line 4)
\end{verbatim}

Each line corresponds to a BCD record. The second and third records are null and produce the line spacing illustrated.

**Example 2**

\begin{verbatim}
PRINT 11,A,B,C,D
11 FORMAT (2E10.2/2F7.3)
\end{verbatim}

A contains -11.6
B contains .325
C contains 46.327
D contains -14.261

Printout:

\begin{verbatim}
-1.16E+01bb3.25E-01
46.327-14.261
\end{verbatim}  (line 1)
\begin{verbatim}
\end{verbatim}  (line 2)
Example 3

PRINT 11,A,B,C,D
11 FORMAT (2E10.2/ /2F7.3)

Printout:

```
-1.15E+01 3.25E-01 (line 1)
46.327 14.261 (line 2)
```

Example 4

PRINT 15,(A(I),I=1,9)
15 FORMAT (8HbRESULTS2(/)(3F8.2))

Printout:

```
RESULTS
3.62  -4.03  -9.78 (line 2)
-6.33   7.12   3.49 (line 4)
 6.21  -6.74  -1.18 (line 5)
```

Note: see section below on unlimited groups
REPEATED FORMAT SPECIFICATIONS

Any FORMAT specification may be repeated by using an unsigned integer constant repetition factor, k, as follows: K(spec), where spec is any conversion specification except nP. For example, to print two quantities, K,L:

```
PRINT 10,K,L
10 FORMAT (I2,I2)
```

Specifications for K,L are identical; the FORMAT statement could use a repeat factor with I2.

```
10 FORMAT (2I2)
```

When a group of FORMAT specifications repeats itself, as in

```
FORMAT(E15.3,F6.1,I4,I4,E15.3,F6.1,I4,I4)
```

the use of k produces

```
FORMAT (2(E15.3,F6.1,2I4))
```

The parenthetical grouping of FORMAT specifications is called a repeated group. Repeated groups may be nested to two levels:

```
FORMAT (spec1,n(spec2) ...)
```

A grouping (k(kspec)) such as

```
FORMAT(2(2E15.3,2F6.1,2I4))
```

is allowed. More than two levels in a repeated parenthesis group produces the diagnostic PAREN GROUP NOT CLOSED IN FORMAT STATEMENT.
FORMAT specifications may be repeated without using a repetition factor. The innermost parenthetical group that has no repetition factor is unlimited and will be used repeatedly until the I/O list is exhausted. Parentheses are the controlling factors in repetition. The right parenthesis of an unlimited group is equivalent to a slash. Specifications to the right of an unlimited group can never be reached.

Example

Where A=1.2, B=2.3, I=1, J=2, K=3

```
PRINT 100,A,B,I,J,K,A,B
100 FORMAT(E16.3,F20.7,2(I4),(I3,F7.1),F8.2)
```

Result: 1.200E+00 2.3000000 1 2 3 1.2

R

The first two fields print according to E16.3 and F20.7. Since 2(I4) is a repeated parenthetical group, the next two fields are printed according to I4. The print fields follow (I3,F7.1), an unlimited group which does not have a repetition factor until the list elements are exhausted. F8.2 can never be reached, and B will be printed using I3, producing a range error. No diagnostic is given for format specifications to the right of an unlimited group.
VARIABLE FORMAT

FORMAT specifications may be defined at the time of program execution. The specification, including left and right parentheses but not the statement label or the word FORMAT, are read under A conversion or defined in a DATA statement and stored in an integer array. The name of the array containing the specifications may be used in place of the FORMAT statement labels in the associated I/O operation. The array name that appears specifies the location of the first word of the format information. This name may appear in an I/O statement as a format with a constant subscript or without subscripts. Variable subscripts are not allowed.

Assume the following FORMAT specifications:

(E12.2,F8.2,I7)

The information could be punched in an input card and read by a program such as:

```
DIMENSION IFMT(2)
READ 100,(IFMT(I),I=1,2)
100 FORMAT (2A10)
```

IFMT contains the FORMAT specification as follows:

- IFMT(1) contains (E12.2,F8.
- IFMT(2) contains 2,I7)

A subsequent output statement uses the FORMAT specification as follows:

```
PRINT IFMT,A,B,I
```

This produces exactly the same result as fixed format in the following program:

```
PRINT 101,A,B,I
101 FORMAT (E12.2,F8.2,I7)
```
Another way to define variable format is with the DATA statement:

\[
\begin{align*}
\text{DIMENSION } & \text{IF1(3),IF2(3),A(6),IP(3),TEMP(3)} \\
\text{DATA IF1}/26H(1H,2F6.3,17,2E12.2,3I1),IF2/24H(1H,I6,6X,} \\
& +3F4.1,2E12.2)/
\end{align*}
\]

The following output statement

\[
\text{PRINT IF1,(A(I),I=1,2),K,B,C,(IP(J),J=1,3)}
\]

is the same as

\[
\text{PRINT 102,(A(I),I=1,2),K,B,C,(IP(J),J=1,3)}
\]

102 FORMAT (1H,2F6.3,I7,2E12.2,3I1)

The output statement

\[
\text{PRINT IF2,LA,(A(M),M=3,4),A(6),(TEMP(I),I=2,3)}
\]

is the same as the following one with fixed format:

\[
\text{PRINT 103,LA,(A(M),M=3,4),A(6),(TEMP(L),L=2,3)}
\]

103 FORMAT (1H,I6,6X,3F4.1,2E12.2)
IX

INPUT/OUTPUT STATEMENTS

DATA TRANSFER
INPUT/OUTPUT TERMINOLOGY
READING AND WRITING
DATASET SUPPORT STATEMENTS
FILE POSITIONING
SUMMARY TABLE OF FORTRAN SYNTAX

Synchronous Input/Output Statements
Asynchronous Input/Output Statements
ENCODE/DECODE Statements
Dataset Support Statements
File Positioning Statements
Data may be transferred from an external device to computer memory, from computer memory to an external device or from one place in computer memory to some other location in computer memory. Data transfers are accomplished in FORTRAN by using the input/output statements described in this chapter.
9.2
Input/Output Statements

UNITS

The unit number in an input/output statement refers to an external device such as a 1/2" tape, a volume on a mass storage device, or the card punch or printer. In core-to-core transfers, the device referenced may also be a variable located in central memory. The unit is designated as u, or, in keyword form, UNIT=u.

SOURCE DATA

Data are transferred as a stream of bits that represent values or characters. The data are organized into records containing one or more values. Data records are separated by record marks. The records and record marks may then be organized into a file. An end-of-file mark may be used to mark the end of a collection of records, i.e., a file.

A more complete description of data sources, data paths and data access is available in two manuals, The NCAR Terabit Memory System (NCAR/TN-124+IA), and The NCAR Data Storage System (NCAR/TN-125+IA). This chapter includes all of the input/output statements with a description of the parameters included in each. More detailed descriptions of related conventions such as the Job Control Language (JCL) appear in the manuals referenced above.
THE I/O LIST

The form iolist indicates an input/output list in statements discussed in this section. The names of variables and array elements to be input or output are specified in the list.

Examples

\begin{verbatim}
WRITE (6,100) A,B,I
PRINT 100, (I,J),(T,U)
DIMENSION A(10),B(10)
WRITE (2)(A(I),I=1,10),B(J),J=1,10)
BUFFER IN (8,1)(A(I),B(10))

DIMENSION A(5),B(5)
READ (5,100)(I,J),A,(B(L),L=1,5)
\end{verbatim}

An array may be input or output using implied DO notation. In the following example, \( N \) (in this case \( N \) is 5) elements of \( A \) are printed.

\begin{verbatim}
DIMENSION A(10)
N = 5
PRINT 100,(A(I),I=1,N)
\end{verbatim}

This same list array may be input or output without DO notation by stating only the array name. In this case, all elements of \( A \) specified in the DIMENSION statement are output (10 in the following example):

\begin{verbatim}
DIMENSION A(10)
PRINT 100,A
\end{verbatim}

RULES

- Constants may not appear in an I/O list.
- Nonstandard subscripts may not appear in I/O lists.
- An I/O list may be empty.
INPUT/OUTPUT TERMINOLOGY

SYNCHRONOUS AND ASYNCHRONOUS

Transfers may be classified as synchronous or asynchronous, depending on whether control is returned to the program before or after completion of the data transfer.

Synchronous data transfer occurs when control is returned to the executing program only after the input/output operation is completed. Asynchronous (often called buffered) data transfer occurs when control is returned to the FORTRAN program before input/output has been completed.

CORE-TO-CORE TRANSFER

Data transfers such as ENCODE/DECODE that move data from one part of core to another core location are synchronous transfers, because the operation must be completed before control is returned to the next FORTRAN statement.
Programmers may use direct or sequential access to files in FORTRAN statements. It is possible to access a file directly by name and its records directly by record number or to read datasets and records sequentially. In addition to the READ and WRITE statements, there are three FORTRAN statements used in support of these access methods: The OPEN, CLOSE and INQUIRE statements. The OPEN statement is used to connect old or new files to a unit or to change some of the attributes of connection to a file. The CLOSE statement disconnects a dataset from a unit. The use of the INQUIRE statement allows the programmer to determine the properties of a file or of connection to a file. The direct access capability and OPEN, CLOSE and INQUIRE statements are FORTRAN 77 features.

Parameters used in these statements have the keyword letters designated in capitals followed by an equals sign and lower case letters, which symbolize a constant or variable which defines either an attribute or an inquiry value.
A data transfer is unformatted if there is no editing of the data to or from character code using a FORMAT statement. A data transfer with editing specified in a FORMAT statement is said to be formatted.

Synchronous and asynchronous input/output statements may use either direct or sequential access methods to transfer records and files that may be either formatted or not formatted.

Core-to-core transfers are always synchronous using sequential access and FORMAT statements.

<table>
<thead>
<tr>
<th>Control</th>
<th>Access Method</th>
<th>Editing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asynchronous</td>
<td>direct</td>
<td>formatted or unformatted</td>
</tr>
<tr>
<td>Asynchronous</td>
<td>sequential</td>
<td>formatted or unformatted</td>
</tr>
<tr>
<td>Synchronous</td>
<td>direct</td>
<td>formatted or unformatted</td>
</tr>
<tr>
<td>Synchronous</td>
<td>sequential</td>
<td>formatted or unformatted</td>
</tr>
</tbody>
</table>
The input/output statements are divided into various groupings in the following pages that describe the syntax. Data transfers from an external device to memory are referred to as reading. Writing describes output operations that transfer information from computer memory to an external device.

The syntax of these statements contains a list of keyword parameters within parentheses that describe the kind of input/output specified. Square brackets indicate optional items in the syntax.

The next two sections describe the generalized form of synchronous and asynchronous input/output on the NCAR system. Following that, some of the specific forms are described in more detail, e.g., the conventional, discrete forms that may be derived from the generalized forms. Other statements that are not generalizable are also included, like BUFFER IN and ENCODE/DECODE.
An input/output statement is called synchronous if control is not returned to the FORTRAN program until the read or write operation has been completed. While either direct or sequential access may be used, any unit specified for direct access must be opened for that type of access in a FORTRAN OPEN statement.

**Form**

```fortran
READ ([UNIT=]u[,[,FMT=]f][,REC=rm][,ERR=s1][,END=s2][,IOSTAT=ios]) iolist
WRITE ([UNIT=]u[,[,FMT=]f][,REC=rm][,MODE=md][,ERR=s1][,IOSTAT=ios]) iolist
```

**Examples**

```fortran
READ (9,100)A,B
READ (UNIT=8,100)X,Y
READ (16,FMT=100,END=25)X
WRITE (UNIT=20,REC=13,ERR=99,END=98)X
WRITE (20,100,ERR=90,END=98,IOSTAT=N:IO)X
```
9.9
Input/Output Statements

INPUT PARAMETERS

u
UNIT=u

u is an expression specifying the unit number associated with a volume on a *VOLUME card and in a FORTRAN OPEN statement, if it appears. If the unit appears without the keyword, it must be the first parameter.

f
FMT=f

f is a FORMAT statement number. If the keyword FMT= is omitted, f must be the second parameter. If f appears in the parameter list, the records in the dataset are formatted; if f does not appear, the records are binary records (unformatted).

REC=r

rn is an expression whose value is the record number specified for reading or writing using direct access. If the parameter REC=rn is specified, the END=s parameter may not be used. The record number specifier is not valid for sequential access.
INPUT PARAMETERS
(continued)

MODE=md

md is an expression defining the mode of data in the record, and will be used to assist in conversion of data between computers. Once given in a write operation, subsequent operations will use the same value unless re-specified. This keyword is ignored in read operations.

0 The data in the record is considered to be normal character data internal to the machine.
1 The data in the record is considered to be binary, bit-serial.
2 The data in the record is considered to be Binary Coded Decimal (BCD).
3 The data in the record is considered to be American Standard Code for Information Interchange (ASCII).
4 The data in the record is considered to be Expanded Binary Coded Decimal Interchange Code (EBCDIC).
5 The data in the record is considered to be in the binary integer format.
6 The data in the record is considered to be normalized floating point numbers.
7 The data in the record is DPC card images truncated to include only the initial non-blank characters in the image.

DEFAULT: MODE=0 for formatted; MODE=1 for binary.

ERR=s₁

s₁ is a statement label to which the program branches if an error condition occurs. If this parameter is omitted, the program terminates on an error condition. Without the use of this keyword parameter in conjunction with the INQUIRE statement, it is very difficult to determine the cause of an error. In using the error branch, the INQUIRE statement may be used to determine the type of error.
If an end-of-file is the last record read, a branch to
statement $s_2$ is taken. If this parameter is omitted,
execution proceeds to the next executable statement following
the end file record. Use of $\text{END}=s_2$ is permitted only with
sequential access.

**OUTPUT PARAMETERS**

$I\text{OSTAT}=\text{ios}$

ios is an integer variable or array element name indicating
the completion status of the input/output operation.

ios=0 successful operation
ios>0 an error code appears in bits 0-11
ios<0 an end-of-file on a read or an end-of-tape on a
write

Error codes are defined under STATUS in chapter 8 of *The NCAR
Terabit Memory System*. Execution terminates on an error
condition unless an $\text{ERR}=s$ parameter appears in the list of
keywords.

$i\text{olist}$

iolist is an input/output list containing the names of
variables and array elements to be processed.

**SYNTAX ERROR
DIAGNOSTICS**

END MUST REFER TO STATEMENT LABEL
ERR MUST REFER TO STATEMENT LABEL
ERR STATEMENT LABEL OUT OF RANGE
FORMAT MUST BE INTEGER CONSTANT
ONLY 1ST OR 2ND ARG MAY BE DEFINED WITHOUT A KEYWORD
UNIT NOT DEFINED
UNRECOGNIZABLE I/O STATEMENT
nTH EXPRESSION NOT RECOGNIZABLE
nTH KEYWORD DOUBLY DEFINED
SPECIFIC FORMS OF SYNCHRONOUS I/O

A number of forms of synchronous I/O are available at NCAR. Some may be derived from the generalized forms in the previous section and are special cases of the generalized form. Some have their own special form, and remain from previous FORTRAN implementations to provide compatibility with past programming conventions. The section is subdivided into input/output, formatted and unformatted.

SPECIFIC OUTPUT STATEMENTS UNDER FORMAT CONTROL

Parameters in the forms used in this section are defined as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>format statement label or array with Hollerith format data</td>
</tr>
<tr>
<td>iolist</td>
<td>input/output list which may contain an implied DO</td>
</tr>
<tr>
<td>u</td>
<td>unit number, unsigned integer constant or variable</td>
</tr>
</tbody>
</table>

READ f[,iolist]

READ f[,iolist] reads one or more card images from the standard input unit, covering the information from left to right in accordance with FORMAT specification f, and stores the converted data in the storage locations named by iolist.

Example

```
READ 10,A,B,C
10 FORMAT (3F10.4)
```

The formatted READ statements transfer at least one record of information from a specified unit (u) to storage locations named by iolist, according to FORMAT specification (f). To read from the card reader, u may either be omitted or set to 5. The following two READ statements are equivalent:

Example

```
READ(5,10)A,B,C
READ 10,A,B,C
```
PRINT f[,iolist] transfers information from the storage locations given by iolist to the standard output unit. The information is transferred as line printer images, 136 characters or fewer per line, in accordance with the FORMAT declaration, f. The maximum record length is 136 characters, with the first character of every record used for carriage control. Characters in excess of the print line are lost; each new record starts a new print line.

<table>
<thead>
<tr>
<th>Carriage Control Character</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank or any character other than the following</td>
<td>Single space before printing</td>
</tr>
<tr>
<td>0</td>
<td>Double space before printing</td>
</tr>
<tr>
<td>1</td>
<td>Eject page before printing causing the record to be printed at the top of a new page</td>
</tr>
<tr>
<td>+</td>
<td>Suppress spacing before printing, causing two successive records to be printed on the same line</td>
</tr>
<tr>
<td>-</td>
<td>Triple space</td>
</tr>
</tbody>
</table>

The characters 0, 1, + and - are not printed as first characters of a line since they are carriage control; any other character which is first in a line causes single spacing and is not printed.

Example

```
PRINT 50,A,B,C(I,J)
50 FORMAT (3X,8HMINIMUM=F17.7,2X,8HMAXIMUM=F8.8, 2X,10HVALUE IS $F8.2)
PRINT 51,(A(I),I=1,20)
51 FORMAT (*1VECTORA*/1H ,10F10.2))
```
PUNCH f[,iolist] transfers information from the storage locations given by iolist identifiers to the standard punch unit. The information is transferred as card images, 80 characters or fewer per card, in accordance with the FORMAT declaration, f.

Example

```
PUNCH 52,ACCT,ISTAT,LOC,TE,PE,1
52 FORMAT (F8,3X4A10,2F10.2,I5)
```

The above format assumes the following dimension statement:

```
DIMENSION ISTAT (2),LOC(2)
```

WRITE (u,f)[,iolist] The formatted output statements are forms which transfer information from storage locations given by iolist to a specified output unit (u) according to the FORMAT declaration, f. The unit number is 6 to designate the printer. If it is not 6, some external device such as tape or the mass storage device must be specified on a *ASSIGN or *VOLUME card. See *User's Guide, Part II (NCAR/TN-106+IA) for job control language information.

The record containing up to 136 characters is recorded on magnetic tape in even parity if u is assigned to a tape unit. The number of words in iolist output according to the FORMAT declaration (f) determines the number of records that are written on a unit.

On the NCAR system, unit 6 refers to the printer, and unit 5 to the card reader. If the user switches these unit numbers by mistake, no message is printed, but the I/O statement will not work.
If the programmer fails to allow for a printer control character, the first character of the output data is lost on the printed listing.

Example

```
WRITE (6,53)A,B,C,D
53 FORMAT (IHO,4E21.9)
WRITE (6,54)
54 FORMAT (32H THIS STATEMENT HAS NO DATA LIST)
```
UNFORMATTED
INPUT STATEMENTS

READ (u)[iolist]  The unformatted input statements are forms which transfer one record of information from a specified unit (u) to storage locations named by iolist. Since a FORMAT statement is not given, no character conversions are performed on the data; the data is read in binary mode, odd parity.

A record read by READ (u) in FORTRAN should have been written in binary (unformatted) mode by a corresponding WRITE statement in FORTRAN. However, the number of words in the list of READ *(u)iolist may be fewer than the number of words in the corresponding WRITE statement. Only those variables read will be stored in the program.

If iolist is omitted, READ (u) spaces over one record. If iolist is shorter than the record written, READ *(u)iolist spaces over the unread elements at the end of the record. (Unread list elements may appear only at the end of a list.)

Examples

DIMENSION C(264)
READ (10)C
DIMENSION BMAX(10),M2(10,5)
DO 9 I=1,10
9 READ (7)BMAX(I),(M2(I,J),J=1,5)
READ(7)((A(I,J),I=1,100),J=1,50)
READ (8)
The unformatted output statements are forms which transfer information from storage locations given by iolist to a specified output unit (u) in binary mode, odd parity. If iolist is omitted, the WRITE (u) statement acts as a do-nothing statement. One FORTRAN WRITE statement constitutes one record on tape.

**Example**

```fortran
DIMENSION A(250), B(4000), AMAX(10), M(10,5)
WRITE (10), A, B
DO 5 I=1,10
  5 WRITE (9), AMAX(I), (M(I,J), J=1,5)
```

The number of words in iolist determines the number of physical records that are written on that unit for a given record in cases where the reference is to 1/2" tape.

The conversion parameter (CONV=LG) on the *VOLUME card (see chapter 3 of *The NCAR Terabit Memory System* and pages 3.8-9 of *An Introduction to the NCAR Data Storage System*) inhibits writing the two control words when the disk system is used for staging. These control words will only appear on 1/2" magnetic tape. Units assigned to the drum or disk do not have this record format, nor does the use of CONV=LG on the *VOLUME card generate this format. Without CONV=LG on the *VOLUME card, the two extra control words are written, unless VSN=DISK, or VSN=DRUM.

**Example**

```fortran
*ASSIGN, DISK=1

*VOLUME, VSN=3, NUM=3, STAGE, IN=NS, NM, STAGEOUT=ZT, CONV=LG
*VOLUME, VSN=4, NUM=3, STAGE, IN=NS, NM, STAGEOUT=ZT, CONV=BN
*ASSIGN, B3141=4

PROGRAM ID
COMMON X(J)
DO 1 I=1,5, J
  1 X(I) = RANF(4)
DO 3 NUNIT=1,4
WRITE (NUNIT), X
3 CONTINUE
END
```

### TAPE ACTIVITY SUMMARY

<table>
<thead>
<tr>
<th>TAPE</th>
<th>PHYSICAL</th>
<th>CHANNEL</th>
<th>PARITY ERRORS</th>
<th>READER ERRORS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DUM2</td>
<td>B</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DUM3</td>
<td>B</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B0141</td>
<td>B</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**STORAGE**

<table>
<thead>
<tr>
<th>TAPE</th>
<th>PHYSICAL</th>
<th>CHANNEL</th>
<th>PARITY ERRORS</th>
<th>READER ERRORS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DUM2</td>
<td>B</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DUM3</td>
<td>B</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B0141</td>
<td>B</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FILE MARKS</th>
<th>RECORDS</th>
<th>WRITE RECORDS</th>
<th>WRITTEN RECORDS</th>
<th>WRITTEN WORDS</th>
<th>FILE MARKS</th>
<th>RECORDS</th>
<th>WRITE RECORDS</th>
<th>WRITTEN RECORDS</th>
<th>WRITTEN WORDS</th>
</tr>
</thead>
</table>
**STRUCTURE OF LOGICAL RECORDS**

This statement will write a logical record having 63 words of data. The logical tape record is in the format shown in the following example:

Physical Record 1:

- **Count of data words in this physical record**
- **First physical record in this logical record**
- **Checksum of the 510 data words (60 bit ACL... add with end-around carry)**

<table>
<thead>
<tr>
<th>Bits</th>
<th>1</th>
<th>23</th>
<th>18</th>
<th>18</th>
<th>60</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Physical Record 2:

- **Last physical record in this logical record**
- **Data words**
- **Second physical record**
- **Checksum of 122 data words**

<table>
<thead>
<tr>
<th>Bits</th>
<th>1</th>
<th>23</th>
<th>18</th>
<th>18</th>
<th>60</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Since the maximum size of a physical record for an unformatted binary write on 1/2" tape is 510 data words, the logical record of 630 words is broken into two physical records. Due to the presence of control words, a total of 634 words is written onto tape, in this example (2+510+2+120).
An input/output statement is called asynchronous if control is returned to the FORTRAN program before the read or write operation has been completed. Buffered or asynchronous input/output allows the user to initiate an I/O request and continue executing while it is being processed. Asynchronous requests are recognized by the presence of the keyword NWORDS in the keylist. This I/O form may be used with either direct or sequential access method. A single address, fwa, which is a variable or array element, locates the first word of the block of words to be processed. Asynchronous input/output is a local extension to the FORTRAN standard and is not portable. The alternate form is BUFFER IN/BUFFER OUT (see the next section). The system status word, ns, will be zero from the time the operation is initiated until it is completed.

**FORM**

```
READ ([UNIT=]u[,REC=rn],NWORDS=n[,NSTATE=ns]) fwa

WRITE ([UNIT=]u[,REC=rn][,MODE=md],NWORDS=n[,NSTATE=ns]) fwa
```

**Examples**

```
READ (10,NWORDS=100,NSTATE=NST) A(1)
WRITE (16,NWORDS=1000,REC=10,NSTATE=NSTA) X(1)
```
INPUT PARAMETERS

u UNIT=u

u is an expression specifying the unit number associated with a volume on a VOLUME card and in a FORTRAN OPEN statement, if it appears. If the unit appears without the keyword, it must be the first parameter.

REC=rm

rn is an expression whose value is the record number of the record desired within the dataset connected for direct access. rm must be specified if the file is open for direct access. It is not valid for sequential access.

MODE=md

md is an expression defining the mode of data in the record, and will be used to assist in conversion of data between computers. Once given in a write operation, subsequent operations will use the same value unless re-specified. This keyword is ignored in read operations.

md 0 The data in the record is considered to be normal character data internal to the machine.
1 The data in the record is considered to be binary, bit-serial.
2 The data in the record is considered to be Binary Coded Decimal (BCD).
3 The data in the record is considered to be American Standard Code for Information Interchange (ASCII).
4 The data in the record is considered to be Expanded Binary Coded Decimal Interchange Code (EBCDIC).
5 The data in the record is considered to be in the binary integer format.
6 The data in the record is considered to be normalized floating point numbers.
7 The data in the record is DPC card images truncated to include only the initial non-blank characters in the image.

DEFAULT: Mode=1.
NWORDS=n

n is an expression that, when evaluated, provides the number of words to be transmitted in the data transfer. In writing, n must be less than or equal to the record length specified in the OPEN statement. n is a required parameter in all asynchronous input/output statements.

OUTPUT PARAMETERS

NSTATE=ns

ns is an integer variable returned by the system to indicate the status of the current operation. This may be tested at any time after the read or write, and will contain zero if the operation is not yet complete. See the discussion of the status word in chapter 8 of The NCAR Terabit Memory System.

ns>0 the count of words transferred is in bits 0-17 and bit 49 is set if an end-of-file was encountered.
ns<0 the specific error code is in bits 48-58.
ns=0 the operation is not complete.

The operation will be forced to completion by either an INQUIRE on this unit or a subsequent READ or WRITE on this unit.

fwa

fwa is a variable name which establishes the beginning of the buffer to be transferred. The length of the buffer is specified as n by the keyword NWORDS. Note that a list may not be given here, and will be flagged by the compiler as an error.
9.22
Input/Output Statements

SYNTAX ERROR
DIAGNOSTICS

- MODE MUST BE DEFINED FOR I/O STATEMENT
- UNIT MUST BE DEFINED FOR I/O STATEMENT
- UNIT NOT DEFINED
- UNRECOGNIZABLE I/O STATEMENT
- nth EXPRESSION OR KEYWORD NOT RECOGNIZABLE
- nth KEYWORD DOUBLY DEFINED
Buffering data in and out of the computer while calculations are performed is an effective way to solve problems requiring large amounts of core. One odd- and one even-numbered logical unit should be assigned to the drum for buffered I/O using the drum.

Three characteristics of the buffer I/O statements are given below:

- The mode of transmission (0 or 1) in a buffer control statement must be specified by a parity indicator.

- The buffer control statements are not associated with a list; data transmission is from a first word address (fwa) to a last word address (lwa).

- A buffer control statement initiates data transmission and then returns control to the program, permitting the program to perform other tasks while data transmission is in progress. Before using any of the buffered data, the status of the buffer operation should be checked.

A magnetic tape written in odd parity must be buffered in odd parity; a tape written in even parity must be buffered in even parity. (While the buffer statement works for tapes made with other I/O statements, care should be taken to allow for differences in record format.)
Parameters in the forms used in this section are defined as follows:

<table>
<thead>
<tr>
<th>u</th>
<th>unit number, unsigned integer constant or variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>parity; the recording mode interpretations for tapes and drums are:</td>
</tr>
<tr>
<td></td>
<td>0 selects even parity †</td>
</tr>
<tr>
<td></td>
<td>1 selects odd parity</td>
</tr>
<tr>
<td>s</td>
<td>a statement label</td>
</tr>
<tr>
<td>fwa</td>
<td>a variable identifier; first word of data block to be transmitted</td>
</tr>
<tr>
<td>lwa</td>
<td>a variable identifier; last word of data block to be transmitted. The fwa must be less than or equal to the lwa in the buffer statement. If not, the job will be terminated.</td>
</tr>
</tbody>
</table>

The buffer transmits information from unit u in mode p to storage locations fwa sequentially through lwa.

This statement transmits information from storage locations fwa through lwa and writes one record on logical unit u in mode p. The record structure for buffered I/O is different from a binary unformatted WRITE (u). The record size is the length from fwa to lwa with no breakdown for smaller block sizes. It is essentially a long record WRITE depending entirely on fwa and lwa.

† The p parameter set to 0 will specify ASCII if a 9-track tape has been assigned. A p value of 0 is typically used for character data and 1 for numeric data.
**RECORD STRUCTURE**

**BUFFER OUT (KTAPE,1)(DATA(1),DATA(632))**

This statement will write a record containing 632 data words.

```
|word1|word2|word3|...................|word632|
```

It is poor practice to generate excessively long tape records with BUFFER OUT statements.

**K=LENGTHF(u)**

The length function (type integer) is used to find the number of 60-bit words buffered in during the last I/O operation on unit u. It may be used with buffered I/O statements and is preceded by an IF(UNIT,u) test to ensure that the I/O is completed and there were no errors. LENGTHF(u) will force completion of an operation. However, if this unit is referenced and there were unchecked errors in the last operation, the job will be terminated.

**STATUS STATEMENT**

**IF(UNIT,U)s1,s2,s3,s4**

This statement checks the status of units used for BUFFER IN/BUFFER OUT operations. It will not work in conjunction with other I/O statements.

The transfer points $s_i$ are interpreted as follows:

- $s_1$: not ready
- $s_2$: ready and no previous error
- $s_3$: tape mark sensed on last operation
- $s_4$: parity error sensed on last operation
As soon as the IF(UNIT,u)sl,s2,s3,s4 is encountered in a FORTRAN program, the execution waits for the unit to be ready and data transfer to be completed. No branch is ever taken to statement s1, as control will not return to the FORTRAN program until the unit is ready. The statement label "s1" should appear as the label on the IF(UNIT,u) statement itself to prevent a MISSING STATEMENT LABEL diagnostic.

When true buffering is to be achieved, the code for calculations to be continued during buffering must appear between the buffer statement and the IF(UNIT,u) statement. These calculations should be long enough to use much of the wait time, or the program will be stopped at the IF(UNIT,u) test until I/O is completed.

Example

```
... BUFFER IN(3,1)(A(1),A(100)) (calculations not involving A) ...
... 10 IF(UNIT,3)10,11,12,13 ...
```

Note that blocking and staging by the system invoked when using the *VOLUME or *TLIB card may override the asynchronous code supplied by the user in a FORTRAN program. Consult the Library and Consultation Office if in doubt about the appropriate code to use.
INTERNAL FILES

ENCODE/DECODE STATEMENTS

The ENCODE/DECODE statements transfer information from one area of memory to another area of memory. They are similar to formatted READ/WRITE instructions since information is transferred under FORMAT specifications. No peripheral (i.e., I/O) device is used in ENCODE/DECODE. The array that references the information encoded or decoded is called an internal file.

Parameters

Parameters in the forms used in this section are defined as follows: (Let ICC equal Internal Character Code. The internal character code currently refers to the display code character set of the 7600, using 6-bit characters and 10 characters per word.)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cc</td>
<td>an unsigned integer constant or a simple integer variable (not subscripted) specifying the number of characters in the ICC record. When cc is not a multiple of 10, blanks will be used to fill the last word.</td>
</tr>
<tr>
<td>f</td>
<td>a statement number, a variable identifier, or a formal parameter representing the FORMAT statement.</td>
</tr>
<tr>
<td>v</td>
<td>a variable identifier or an array identifier which supplies the starting location of the ICC record. v is always in ICC whether ENCODE or DECODE is used. The identifier may have standard or nonstandard subscripts.</td>
</tr>
<tr>
<td>iolist</td>
<td>input/output list; may contain an implied DO.</td>
</tr>
</tbody>
</table>
ENCODE/DECODE

Example

C WRITE IOLIST INTO INTERNAL FILE V
   ENCODE (cc,f,v)iolist

ENCODE is similar to a WRITE of the internal file v (in ICC). v is generated by the ENCODE from iolist (the list).

Example

C READ FROM INTERNAL FILE V INTO IOLIST
   DECODE (cc,f,v)iolist

DECODE is similar to READ. iolist (the list) is generated by the DECODE from the internal file v (in ICC).

Keep in mind that v is always the ICC array. The direction of the arrows shows whether v (ICC) is being encoded (defined) or decoded (broken down) by the FORMAT specification.

ENCODE(cc,f,v)[iolist] ENCODE is a data transfer from the list variables to some other array called the internal file. The transfer is done using a FORMAT specification so that the destination array contains the encoded list variables in internal character code.

   ENCODE (10,100,IA)A
100 FORMAT (*bA=*F7.2)

The number A is sent to IA using the FORMAT statement 100. IA is the internal file. The character count, 10, is the number of characters specified in the FORMAT statement including the alphanumeric information.

If A is equal to 1.0, then after the ENCODE
   A is 17204000000000000000B)
      (1.0 in floating point)
   IA is 55015455555534573333B
      ( A = 1.00 )

Input/Output Statements

ENCODE/DECODE

Statements (continued)
ENCODE (cc,f,v)iolist is similar to PRINT f,iolist except that the destination is an internal file. The information from iolist is converted according to FORMAT f and stored, character by character, starting at the left end of the array specified by v, the internal file.

cc is the character count per record; "record" in this case is like line length in a normal PRINT statement. In the associated FORMAT statement, f, an end of "record" is indicated by a slash or by reaching the last right parenthesis with a repeat factor.

The FORMAT statement indicates how many characters are to be used in the conversion of each list variable, the type of conversion for that variable, and the record length of the statement.

RULES

- cc must be less than or equal to 150.

- The maximum number of characters specified in the longest record in the FORMAT statement f must be less than or equal to cc.

- If cc is not a multiple of 10, the encoded record will be blank filled to the next larger multiple of 10; however, only cc characters will actually undergo conversion.

- v must be dimensioned if the total number of characters encoded is greater than 10.
Example 1

```plaintext
DIMENSION L(2)
I = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
J = '1234567890'
ENCODE (5, 100, L) I, J
100 FORMAT (A5/A7)
```

<table>
<thead>
<tr>
<th>L(1)</th>
<th>L(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C D E b b b b b</td>
<td></td>
</tr>
<tr>
<td>1 2 3 4 5 b b b b b</td>
<td></td>
</tr>
</tbody>
</table>

The slash in the format indicates that two records are specified. Since the character count per record has a maximum of five, only five characters of J were converted, even though the FORMAT called for seven. Since \( cc \) is 5, each record was blank filled to 10, the next larger number which is a multiple of 10.

Example 2

```plaintext
DIMENSION L(4)
I = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
J = '1234567890'
ENCODE (18, 100, L) I, J
100 FORMAT (A5/A7)
```

<table>
<thead>
<tr>
<th>L(1)</th>
<th>L(2)</th>
<th>L(3)</th>
<th>L(4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C D E b b b b b</td>
<td>b b b b b</td>
<td>1 2 3 4 5 b b b b b</td>
<td>b b b b b</td>
</tr>
<tr>
<td>b b b b b</td>
<td>b b b b b</td>
<td>b b b b b</td>
<td>b b b b b</td>
</tr>
</tbody>
</table>

Only the number of characters called for in the FORMAT (five and seven for each record) are actually converted. The record length (18) is not a multiple of 10, so blank fill is provided to the next larger multiple of 10 (20) for each record. Thus, L(2) and L(4) are blank words to complete the character count of 18 for each record.
Example 3

```
DIMENSION L(20),A(10),B(10)
DO 1 I=1,10
A(I) = I
1 B(I) = 1+20
ENCODE (20,100,L)(A(I),B(I),I=1,10)
100 FORMAT(2HA=,F8.2,2HB=,F7.1)
```

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>i</td>
<td>l</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>2</td>
<td>l</td>
<td>0</td>
<td>b</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>b</td>
</tr>
</tbody>
</table>

There are 20 characters per record and 10 records to fill L from L(1) to L(20), two words per record. The Hollerith in the FORMAT is included in the character count.
ENCODE/DECODE
STATEMENTS
(continued)

ENCODE may be used to calculate a field definition in a FORMAT specification at object time. Assume that in the statement FORMAT(2A8,Im) the programmer wishes to specify m at some point in the program, subject to the restriction $2 \leq m \leq 9$.

Example 4

```plaintext
IF(M.LT.10.AND.M.GT.1)1,2
1 ENCODE (8,100,IFMT)M
100 FORMAT (6H(2A8,I,I,IIH))
;
PRINT IFMT, IA,NA,J
```

M is tested to ensure it is within limits. If not, control goes to statement 2 which could be an error routine. If M is within limits, ENCODE first packs the six Hollerith characters (2A8,I, then, if M = 5, M is encoded to include 5 so that it is now (2A8,I5. Then the right parenthesis is added by using 1H). IFMT contains (2A8,I5) after the encode. IFMT must be type integer.

The PRINT statement will print IA and NA under specification A8, and the quantity J under specification I2, or I3, or ..., or I9 according to the value of M.

Example 5

```plaintext
DIMENSION LTIT(4)
ENCODE (38,100,LTIT)IDATE,TE
100 FORMAT (*APRIL,*,I2,*,1969,TEMPERATURE AT*,F3.0,*DEGREES*)
```

A title for plotting using a graphics subroutine might be encoded inserting the appropriate day and temperature for each graph. Encoding a new IDATE and TE for each graph provides a variable title.
DECODE\(cc,f,v\)\[iolist\] DECODE is a data transfer from an ICC array to the specified list variables. The transfer is done according to the FORMAT statement; the ICC array is interpreted, and list variables are stored from the ICC array as binary numbers.

\[
\text{DECODE}(10,100,\text{IA})A
\]
\[100 \text{ FORMAT}(3\text{XF7.2})\]

The information in IA, the internal file, is internal character code.

\[
55015455555534573333 \\
b \quad A = b b b 1.00
\]

3X in FORMAT 100 skips the first three characters, bA-, then F7.2 is used to store a floating point 1 in A. cc, the character count for the format, is 10.

DECODE \((cc,f,v)\)iolist is similar to READ \(f,\)iolist. The ICC information contained in cc consecutive characters starting at v is converted according to FORMAT \(f\) and stored in the list variables iolist.

The number of characters decoded per record is dependent on the list iolist and the format f.
Rules

- cc refers to the number of characters to be decoded per record. The record is the smallest number of consecutive computer words that can contain cc characters. cc must be less than or equal to 150.

- If the FORMAT specification is exhausted (i.e., coming to the last right parenthesis) before cc characters have been decoded or the list iolist is satisfied, a new record is started at the beginning of the FORMAT statement.

- A slash in the FORMAT specification initiates a new record.

- If the list iolist is satisfied before reaching the end of the FORMAT specification or before cc characters have been decoded, decoding will stop.

- An execution diagnostic will occur if DECODE attempts to process an illegal ICC code.

- If the list iolist contains a dimensioned variable with no specific indexing treatment, it is assumed to mean the whole array, as specified in the DIMENSION statement.
Examples

The following data applies to each of the four examples below:

Given: DIMENSION L(5),A(6)

L(1) = 1 2 3 4 5 6 7 8 9 0
L(2) = 1 1 . 4 A B C 7 2 .
L(3) = 0 9 8 7 6 5 4 3 2 1
L(4) = 2 3 . 7 C X Y b 1 1
L(5) = 5 2 6 3 7 1 8 0 9 4

Example 1

C READ FROM INTERNAL FILE L INTO A
100 FORMAT(2F5.0,F4.1)
DECODE (14,100,L)(A(I),I=1,6)

produces
A(1) = 1234.5
A(2) = 67890.
A(3) = 11.4
A(4) = 9876.
A(5) = 54321.
A(6) = 23.7

Fourteen characters and three words are decoded per record.
Since there are six words in the array A, the format is
scanned twice.
Example 2

C READ FROM INTERNAL FILE L INTO IOLIST
101 FORMAT(I2/F4.1)
   DECODE (10,101,L)(I,A,J,B,K)

produces  I = 12
          A = 11.4
          J = 9
          B = 23.7
          K = 52

DECODE (4,101,L)(I,A,J,B,K) would produce identical results. Since five words are specified in the list, the scan repeats until the list is satisfied. Decoding stops after using I2 for K in the third repeat of the scan.

Example 3

C READ FROM INTERNAL FILE L INTO K
102 FORMAT(3I2,F5.0,F3.1,A3)
   DECODE (17,102,L)K

produces  K = 12

DECODE stops when the list is exhausted. Only one word is specified, K, so the scan stops after the first I2 in the list.

Example 4

C READ FROM INTERNAL FILE L INTO IOLIST
102 FORMAT(3I2,F5.0,F3.1,A3)
   DECODE (17,102,L)I,J,K,X,Y,NAM

produces  I = 12
          J = 34
          K = 56
          X = 78901.
          Y = 1.4
          NAM = ABCBbbbbbbb

The format is scanned completely once, since there are six variables specified in the list.


**Generalized Form**

Transfers from an iolist in memory to or from an internal file is also specified in FORTRAN 77. This is not yet implemented on the NCAR 7600 compiler, since the internal file requires a CHARACTER data type name in the generalized form.
The dataset support statements are OPEN, CLOSE, SETPASS and INQUIRE. These are used to perform various operations on files and to check the status of file characteristics. These statements, which are FORTRAN 77 constructs, are used in addition to the READ and WRITE statements.

The OPEN statement is used to connect files to a unit. The unit is associated with a volume in the job control language. When an input/output statement appears referencing the unit associated with a volume, data are transferred to or from the dataset according to the attributes in the OPEN statement and the keyword specifiers in the I/O statement. A subsequent OPEN statement on the same unit for the same file may change some of the attributes and access methods described in the keyword parameters in the list for a previously connected dataset in a volume. Temporarily, there is a restriction on parallelism in the use of the OPEN statement. Two datasets on the same volume may be connected consecutively, but may not be connected at the same time. An implicit CLOSE is assumed between successive OPENs on the same unit.

Square brackets denote optional items; constant parameters of type character must be enclosed in apostrophes or preceded by nH.

```
OPEN (iden, keylist)

OPEN ([UNIT=]u[,FILE=dsi[,FILESEQ=n][,ERR=s][,FORM=fm]
[,STATUS=sta][,RECL=r1en][,MAXREC=maxr][,ACCESS=acc]
[,GEN=gen][,EXPDT=exp][,PASS=pass][,IOSTAT=ios]
[,BLANK=blnk])
```
IDENT PARAMETERS

ident specifies the unit identifier, the dataset name, or the file sequence number. In all cases the unit must be specified. It may appear in any one of the following keyword identifier combinations:

- [UNIT=]u
- [UNIT=]u,FILE=dsi
- [UNIT=]u,FILESEQ=n
- [UNIT=]u,FILE=dsi,FILESEQ=n

u

UNIT=u

u is an expression specifying the unit. There are two forms for specifying the unit: UNIT=u or simply u. u is the unit number specified on the *VOLUME card. The parameter u must appear in the list and it must appear first in an OPEN statement if the UNIT= is omitted.

Unit numbers may range from 1 to 3552. Unit numbers greater than 3552 may be handled if the user understands the systems implications of the number opened. See a systems programmer for units greater than 3552.

FILE=dsi

dsi is a character constant or variable of 17 or fewer characters, beginning with a letter or number and dimensioned for two words. It is left-justified with no embedded blanks or commas, and specifies the dataset name for a new file. Unused character positions in the two words used for dsi should be blank-filled. If STATUS='OLD', dsi serves to identify an already existing file. Note that this parameter is optional. If the file status is NEW and this parameter is omitted, the system default label is used. If the file status is OLD and this parameter is omitted, positioning is by file sequence number.

DEFAULT: FILE=17HNCARSYSTEMND1nnnn, where nnnn is the FILESEQ number.
IDENT PARAMETERS
(continued)

FILESEQ=n
n is an integer constant or variable that specifies the sequence number of the dataset to be opened. It represents the current position of a file on a volume. The value of n may range from 1 to 600. Note that this parameter is optional. If both FILE=dsi and FILESEQ=n are omitted, the file is positioned as though FILESEQ=1.

ERROR CONDITION
STATEMENT LABEL

ERR=s
s is a statement label to which the program branches when an error condition occurs during the execution of the OPEN statement.

DEFAULT: The program terminates on an error condition.

KEYLIST PARAMETERS

FORM=fm
fm is an alphanumeric constant or variable (dimensioned for 2) specifying whether the data is FORMATTED or UNFORMATTED.

DEFAULT: FORM='UNFORMATTED'

STATUS=sta
sta is an alphanumeric constant or variable specifying whether the dataset is NEW, OLD, SCRATCH, or UNKNOWN. Datasets that are NEW have not been created prior to execution of the OPEN statement. At the end of each run, SCRATCH files are deleted.

DEFAULT: STATUS='UNKNOWN'
9.41
Input/Output Statements

RECL=rlen
rlen is an expression that establishes the maximum record length of all records in the dataset. rlen may not have a negative value. If given for an OLD file with a defined record length, the length must match. The value is undefined if not given for a NEW sequential file. For a NEW file to be used for direct access, length is a required parameter.

MAXREC=maxr
maxr is an expression specifying the maximum number of records that may appear in a dataset. An error occurs if an attempt is made to reference a record number larger than maxr. This parameter may only appear with NEW files.

DEFAULT: MAXREC=1000 for direct access on a NEW file, and MAXREC=undefined for sequential access.

ACCESS=acc
acc is an alphanumeric constant or variable whose value is DIRECT or SEQUENTIAL. It defines the method for accessing records within the dataset.

DEFAULT: ACCESS='SEQUENTIAL'

EXPDT=exp
exp is a character constant or variable. It contains the julian expiration date, consisting of two numeric characters for the year, followed by three numeric characters for the day within the year. Files will be deleted on or after the expiration date and data will not be recoverable.

DEFAULT: EXPDT=one year later than the creation date
KEYLIST PARAMETERS
(continued)

†PASS=pass

pass is a security parameter supplied by the user which allows four kinds of access to a dataset. To establish passwords that are valid here, assign them using the FORTRAN SETPASS instruction. The four levels are:

<table>
<thead>
<tr>
<th>owner</th>
<th>unlimited access</th>
</tr>
</thead>
<tbody>
<tr>
<td>read/write</td>
<td>user may read and write the dataset, but is unable to change the password using the SETPASS instruction</td>
</tr>
<tr>
<td>read</td>
<td>access to the dataset is for reading only</td>
</tr>
<tr>
<td>write</td>
<td>access to the dataset is for writing only</td>
</tr>
</tbody>
</table>

Refer to the section on SETPASS for information on creating and maintaining passwords.

†BLANK=blnk

blnk is an alphanumeric constant or variable that is NULL or ZERO. NULL causes all blanks in an input field to be removed. ZERO causes all blanks to be zeros.

DEFAULT: In the current implementation, this parameter is always set to ZERO. Omit it.

RETURN PARAMETER

IOSTAT=ios

ios specifies whether an error occurred during the execution of the OPEN statement.

ios=0  successful open
ios≠0  error condition

† This feature was not yet implemented at publication time.
SYNTAX ERROR  
DIAGNOSTICS

ERR MUST REFER TO STATEMENT LABEL
ERR STATEMENT LABEL OUT OF RANGE
OPEN STATEMENT-ERROR IN ACCESS. VALUE MUST BE DIRECT OR SEQUENTIAL
OPEN STATEMENT-ERROR IN ERR. MUST BE STATEMENT LABEL
OPEN STATEMENT-ERROR IN EXPDT. MUST BE ALPHANUMERIC DATE
OPEN STATEMENT-ERROR IN FILE. MUST BE 17 CHARACTERS OR LESS ALPHANUMERIC CONSTANT OR VARIABLE
OPEN STATEMENT-ERROR IN FILESEQ
OPEN STATEMENT-ERROR IN STATUS
UNIT NOT DEFINED
UNRECOGNIZABLE I/O STATEMENT
nTH EXPRESSION OR KEYWORD NOT RECOGNIZABLE
nTH KEYWORD DOUBLY DEFINED
THE CLOSE STATEMENT

The CLOSE statement is used to disconnect a dataset from a given unit. Executing a CLOSE for a dataset that does not exist causes no action. An implicit CLOSE is assumed between two successive OPENs on the same unit.

FORM

CLOSE ([UNIT=]u[,STATUS=sta][,ERR=s][,IOSTAT=ios])

INPUT PARAMETERS

u
UNIT=u

u is an expression specifying the unit. There are two forms for specifying the unit: UNIT=u or simply u. u is the unit number specified on the *VOLUME card. The parameter u must appear in the list and it must appear first if the UNIT= is omitted.

Unit numbers may range from 1 to 3552. Unit numbers greater than 3552 may be handled if the user understands the systems implications of the number opened. See a systems programmer for units greater than 3552.

STATUS=sta

sta is an alphanumeric constant or variable assigned a value of KEEP or DELETE. If KEEP is specified, the dataset continues to exist following the CLOSE statement. Using DELETE as the parameter value, the dataset no longer exists for the executable program following the CLOSE statement.

At the end of execution of a job, all units connected to named files are closed implicitly with a status of KEEP, except SCRATCH files, which are deleted.

DEFAULT: STATUS='KEEP', unless the status in the OPEN is SCRATCH, where the dataset is deleted when closed.
**ERROR CONDITION**

**STATEMENT LABEL**

ERR=s

s is a statement label to which the program branches when an error condition occurs during execution of the CLOSE statement.

DEFAULT: If ERR= is omitted, the program terminates on an error condition.

**RETURN PARAMETER**

IOSTAT=ios

ios specifies whether an error occurred during the execution of the CLOSE statement.

ios=0 successful close
ios≠0 error condition

**SYNTAX ERROR**

**DIAGNOSTICS**

CLOSE STATEMENT--ERROR IN ERR. MUST BE STATEMENT LABEL
CLOSE STATEMENT--ERROR IN STATUS. MUST BE EITHER KEEP OR DELETE
ERR MUST REFER TO STATEMENT LABEL
ERR STATEMENT LABEL OUT OF RANGE
UNIT NOT DEFINED
UNRECOGNIZABLE I/O STATEMENT
nTH EXPRESSION OR KEYWORD NOT RECOGNIZABLE
nTH KEYWORD DOUBLY DEFINED
THE INQUIRE STATEMENT

The attributes or properties of a dataset or a connection are requested by the programmer using the INQUIRE statement. Values of parameters defining these attributes used in a keyword list are returned to the program after the INQUIRE statement is executed. The form of the INQUIRE statement is:

\[
\text{INQUIRE (ident, keylist)}
\]

\[
\text{INQUIRE ([UNIT=]u[,FILE=dsi][,FILESEQ=n][,ERR=e][,IOSTAT=i][,EXIST=e][,OPENED=o][,NAMED=n][,NUMBER=u][,NAME=f][,RECL=r][,MAXREC=m][,ACCESS=a][,SEQUENTIAL=s][,DIRECT=d][,FORM=f][,FORMATTED=f][,UNFORMATTED=u][,NSTATE=n][,MODE=m][,NEXTREC=n][,SEQFIL=n][,GEN=g][,EXPDT=e][,VSN=n][,NUMREC=n][,CRECL=c][,GENTIME=g][,BLANK=b][,CRDT=c][,}]
\]

IDENT PARAMETERS

Ident specifies the unit identifier, the dataset name, or the file sequence number. In all cases the unit must be specified. It may appear in any one of the following keyword identifier combinations:

- [UNIT=]u
- [UNIT=]u,FILE=dsi
- [UNIT=]u,FILESEQ=n
- [UNIT=]u,FILE=dsi,FILESEQ=n
**u**

**UNIT=u**

`u` is an expression specifying the unit. There are two forms for specifying the unit: `UNIT=u` or simply `u`. `u` is the unit number specified on the *VOLUME* card. The parameter `u` must appear in the list and it must appear first if the `UNIT=` is omitted.

**FILE=dsi**

`dsi` is a character constant or variable beginning with a letter or number and dimensioned for two words. It is left-justified with no embedded blanks or commas, and specifies the dataset name with 17 or fewer characters. Unused character positions in the two words reserved for `dsi` should be blank-filled.

**FILESEQ=n**

`n` is an expression designating the sequence number of the file or dataset. This identifier takes precedence over the `FILE=dsi` parameter in the following sense: if both `FILE=` and `FILESEQ=` are specified and do not match, the dataset with the specified `FILESEQ` number is used.

**ERROR CONDITION**

**STATEMENT LABEL**

**ERR=s**

`s` is a statement label to which the program branches when an error condition occurs during the processing of the *INQUIRE* statement.

DEFAULT: If `ERR=` is omitted, the program terminates on an error condition.
**KEYLIST PARAMETERS**

Keylist is a list of one or more of the following keyword forms. Values of dataset properties are returned in the variable names used in the INQUIRE statement. Keylist items which provide properties of a file or a connection apply to the file located by a combination of ident parameters. If neither FILE= nor FILESEQ= are given and the file is open, the keylist items apply to the first file. FILESEQ takes precedence over the FILE= as a keylist identifier. If both FILE= and FILESEQ= are given and they do not match on the file dsi, EXIST= will be returned FALSE and other properties will be returned for the file in that file sequence position.

**IOSTAT=ios**

Ios is an integer variable returned with the error condition that occurred during execution of the INQUIRE statement.

- Ios=0 successful inquiry
- Ios#0 error condition indicated in the right 12 bits of the word

Error codes are defined under STATUS in chapter 8 of *The NCAR Terabit Memory System*.

SUGGESTED FORMAT: 04

**EXIST=ex**

Ex is a logical variable. It is returned with the value .TRUE. if there is a dataset with the specified name dsi on the connected volume. Otherwise, it is .FALSE.. If FILESEQ= is given as an identifier, only the file in that sequential position is checked.

SUGGESTED FORMAT: L2

**OPENED=od**

Od is a logical variable which is returned with .TRUE. if the file dsi has been assigned or connected to a unit. If the dataset is not connected, od is returned with the value .FALSE..

SUGGESTED FORMAT: L2
**NAMED=nmd**
nmd is a logical variable returned with the value .TRUE. if
the dataset has a user-supplied name, otherwise the value
.FALSE. is returned.
SUGGESTED FORMAT: L2

**NUMBER=un**
un is an integer variable returned with the unit opened to
the dataset name, dsi, if any. If no unit is connected to
the file, un is undefined.
SUGGESTED FORMAT: I5

**NAME=fn**
fn is an alphanumeric array dimensioned for two words, and
returned with the name for that file. fn is 17 characters
in length and is left-justified with three characters of
blank fill. If EXIST=.FALSE., fn will be different from dsi.
SUGGESTED FORMAT: 2A10

**RECL=rlen**
rlen is an integer variable returned with the maximum record
length of a file that has been specified in an OPEN state-
ment. If rlen has not been defined by the user in an OPEN
statement, a system default of 0 will be returned.
SUGGESTED FORMAT: I7

**MAXREC=maxr**
maxr is an integer variable returned with the value of the
maximum record number allowed in a file, if defined.
SUGGESTED FORMAT: I9

**ACCESS=acc**
acc is a variable returned with the alphanumeric value
DIRECT or SEQUENTIAL, which applies to the current connect-
ion of that file. If there is no connection, acc is
undefined.
SUGGESTED FORMAT: A10
KEYLIST PARAMETERS
(continued)

SEQUENTIAL=seq

seq is an alphanumeric variable returned with:

YES  sequential access is allowed on this dataset when opened for sequential access
NO   (not used at publication time)
UNKNOWN (not used at publication time)

SUGGESTED FORMAT: A10

DIRECT=dir

dir is an alphanumeric variable returned with:

YES  direct access is allowed on this dataset when opened for direct access
NO   (not used at publication time)
UNKNOWN (not used at publication time)

SUGGESTED FORMAT: A10

FORM=fm

fm is an alphanumeric variable returned with a value of FORMATTED OR UNFORM for the current connection. If there is no connection, fm is undefined.

SUGGESTED FORMAT: A10

FORMATTED=fmt

fmt is an alphanumeric variable that specifies if the file may be connected for formatted input/output:

YES  allowed on all files
NO   (not used)
UNKNOWN (not used)

SUGGESTED FORMAT: A10
UNFORMATTED=unf

unf is an alphanumeric variable that specifies if the file may be connected for unformatted input/output:

YES allowed on all files
NO (not used)
UNKNOWN (not used)

SUGGESTED FORMAT: A10

NSTATE=ns

ns is an integer variable interpreted as a 60-bit octal number, returned with the contents of the system status cell for the previous input/output statement executed for the unit assigned to that dataset. See chapter 8 of The NCAR Terabit Memory System for the contents of the status cell.

SUGGESTED FORMAT: 020

MODE=md

md is a variable returned with the mode of the data referenced in the previous input statement executed on that unit:

md 0 The data in the record is considered to be normal character data internal to the machine.
1 The data in the record is considered to be binary, bit-serial.
2 The data in the record is considered to be Binary Coded Decimal (BCD).
3 The data in the record is considered to be American Standard Code for Information Interchange (ASCII).
4 The data in the record is considered to be Expanded Binary Coded Decimal Interchange Code (EBCDIC).
5 The data in the record is considered to be in the binary integer format.
6 The data in the record is considered to be normalized floating point numbers.
7 The data in the record is DPC card images truncated to include only the initial non-blank characters in the image.

SUGGESTED FORMAT: I2
**KEYLIST PARAMETERS**

*continued*

**NEXTREC=nr**  
nr is an integer variable. The value returned in nr is the number of the record previously read or written on that unit plus one. If no reads or writes have been executed since the last REWIND or OPEN, NEXTREC=1.  
SUGGESTED FORMAT: I9

**SEQFIL=n**  
n is an integer variable returned with the current file sequence number.  
SUGGESTED FORMAT: I4

**GEN=gen**  
gen is an integer variable returned with the generation number. If the unit or file is not connected, the variable is undefined.  
SUGGESTED FORMAT: I4

**EXPDT=exp**  
exp is an alphanumeric variable defining the julian expiration date, consisting of two numeric characters for the year, followed by three numeric characters for the day within the year. If the unit or file is not connected, the variable is undefined. The default value is one year later than the creation date.  
SUGGESTED FORMAT: A5

**VSN=name**  
name is a variable returned with the Volume Serial Number of the volume currently connected to the unit. It consists of 6 alphanumeric characters.  
SUGGESTED FORMAT: A6

**NUMREC=nr**  
nr is an integer variable returned with the number of records in the file being referenced.  
SUGGESTED FORMAT: I9
**CRECL=cl**

cl is a variable returned with the total length of the record read in the previous input/output statement on that unit or file without regard to the number of words specified in the READ statement.

SUGGESTED FORMAT: I8

**GENTIME=gnt**

GENTIME returns the time the file was last read or written in the following format:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>59 - 45</td>
<td>Last time the file was read</td>
</tr>
<tr>
<td>44 - 36</td>
<td>Last day of year the file was read</td>
</tr>
<tr>
<td>35 - 30</td>
<td>Last year the file was read, 0 = 1976</td>
</tr>
<tr>
<td>29 - 15</td>
<td>Last time the file was written</td>
</tr>
<tr>
<td>14 - 6</td>
<td>Last day of year the file was written</td>
</tr>
<tr>
<td>5 - 0</td>
<td>Last year the file was written</td>
</tr>
</tbody>
</table>

SUGGESTED FORMAT: 020

**BLANK=blnk**

blnk is a variable returned with 'ZERO'. The 'NULL' value is not implemented at this time. BLANK='ZERO' interprets blanks as zeros in reading from the input file.

SUGGESTED FORMAT: A4

**CRDT=crdt**

crdt is an alphanumeric variable defining the julian creation date, consisting of two numeric characters for the year, followed by three numeric characters for the day within the year. If the unit or file is not connected, the variable is undefined.

SUGGESTED FORMAT: A5
Input/Output Statements

**SYNTAX ERROR**

ERR MUST REFER TO STATEMENT LABEL
ERR STATEMENT LABEL OUT OF RANGE
INQUIRE STATEMENT-ERROR IN ACCESS. VALUE MUST BE DIRECT, OR SEQUENTIAL
INQUIRE STATEMENT-ERROR IN BLANK
INQUIRE STATEMENT-ERROR IN CRDT
INQUIRE STATEMENT-ERROR IN ERR. MUST BE A STATEMENT LABEL, OR PROGRAM TERMINATES
INQUIRE STATEMENT-ERROR IN EXPDT
INQUIRE STATEMENT-ERROR IN EXIST. MUST BE EITHER .TRUE. OR .FALSE.
INQUIRE STATEMENT-ERROR IN FILE
INQUIRE STATEMENT-ERROR IN FILESEQ
INQUIRE STATEMENT-ERROR IN GEN
INQUIRE STATEMENT-ERROR IN GENTIME
INQUIRE STATEMENT-ERROR IN MAXREC
INQUIRE STATEMENT-ERROR IN MODE
INQUIRE STATEMENT-ERROR IN NAME
INQUIRE STATEMENT-ERROR IN NEXTREC
INQUIRE STATEMENT-ERROR IN NSTATE
INQUIRE STATEMENT-ERROR IN NUMBER
INQUIRE STATEMENT-ERROR IN OPENED
INQUIRE STATEMENT-ERROR IN RECL. MUST BE AN INTEGER VARIABLE
INQUIRE STATEMENT-ERROR IN SEQFIL
UNIT NOT DEFINED
UNRECOGNIZABLE I/O STATEMENT
nTH EXPRESSION OR KEYWORD NOT RECOGNIZABLE
nTH KEYWORD DOUBLY DEFINED
Passwords may be assigned to datasets within a volume using the SETPASS instruction. A dataset may have up to four levels of security using passwords. In order to add or change a password, the dataset must be currently opened; that is, connected to a unit. If SETPASS has never been used to assign a password, the dataset has unlimited access to all users. The four levels of security that may be established are:

<table>
<thead>
<tr>
<th>Level</th>
<th>Access Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>owner</td>
<td>unlimited access</td>
</tr>
<tr>
<td>read/write</td>
<td>access for reading or writing the dataset, but no access for changing a password</td>
</tr>
<tr>
<td>read</td>
<td>access for reading only</td>
</tr>
<tr>
<td>write</td>
<td>access for writing only</td>
</tr>
</tbody>
</table>

**Form**

```plaintext
SETPASS ([UNIT=]u,PASS=pass[,RWPASS=rwpass][,RPASS=rpass][,WPASS=wpass][,CPASS=cpass])
```

**Parameters**

- **`u`**
  - `UNIT=u`
  - `PASS=pass`
  - `PASS` is an 8-character alphanumeric constant or variable specifying the "owner's" password associated with the dataset identified by the parameter, `u`. Blank fill is used if fewer than 8 characters are given. Embedded blanks and commas are not allowed. If the password has not yet been defined, the initial specification establishes `pass` as the owner's password. The owner password must be specified in assigning values to all other levels of passwords using the SETPASS instruction.
Input/Output Statements

PARAMETERS (continued)

RWPASS=rwpass rwpass is an 8-character alphanumeric constant or variable specifying access only for reading and writing the dataset connected to the unit, u. Blank fill is used if fewer than 8 characters are given. Embedded blanks and commas are not allowed. If rwpass is not defined, no access is allowed with the keyword RWPASS.

RPASS=rpass rpass is an 8-character alphanumeric constant or variable specifying access for reading the dataset connected to the unit, u. Blank fill is used if fewer than 8 characters are given. Embedded blanks and commas are not allowed. If rpass is not defined, the dataset may be read without the use of a password.

WPASS=wpass wpass is an 8-character alphanumeric constant or variable specifying access to the dataset connected to the unit, u, only for writing. Blank fill is used if fewer than 8 characters are given. Embedded blanks and commas are not allowed. If wpass is not defined, no access is allowed with the keyword WPASS.

CPASS=cpass cpass is an 8-character alphanumeric constant or variable, specifying that a new owner password, cpass, be substituted for the current owner password, pass. Blank fill is used if fewer than 8 characters are given. Embedded blanks and commas are not allowed. Both keyword parameters, pass and cpass, must be entered, if the owner password is to be changed. CPASS=0 is used to remove all passwords.
**Examples**

SETPASS (9,PASS=8H12345678,RPASS=8H56781234)
If the owner password for the dataset assigned to unit 9 is 12345678, set or change the read only password to 56781234. wpass and rwpass remain unchanged.

SETPASS (9,PASS=8H12345678,CPASS=8H87654321)
pass is changed to 87654321. rpass, rwpass and wpass remain the same.

SETPASS (9,PASS=8H12345678,WPASS=8H4444444A,CPASS=8H9A998B88)
pass is changed to 9A998B88, wpass to 4444444A; rpass and rwpass remain unchanged.

SETPASS (9,PASS=8H12345678,CPASS=0)
All passwords are removed.

**SYNTAX ERROR**

- SETPASS-CPASS KEYWORD IN ERROR
- SETPASS-PASS KEYWORD IN ERROR
- SETPASS-RPASS KEYWORD IN ERROR
- SETPASS-RWPASS KEYWORD IN ERROR
- SETPASS-WPASS KEYWORD IN ERROR
- UNIT NOT DEFINED
- UNRECOGNIZABLE I/O STATEMENT
- nTH EXPRESSION OR KEYWORD NOT RECOGNIZABLE
- nTH KEYWORD DOUBLY DEFINED
FILE POSITIONING
WITH SEQUENTIAL
ACCESS AND
SYNCHRONOUS
INPUT/OUTPUT

File positioning statements are used only with sequential access to volumes. This access method is characterized as positional and order-dependent. The unit specified is assumed to contain a sequence of ordered files and records constituting the data. These files may be user-named or have names assigned by the system when used with the NCAR tape staging feature. Manipulation is basically positional. The following statements may be used to position sequential files.

BACKSPACE u
BACKSPACE ([UNIT=]u[,IOSTAT=ios][,ERR=s])
ENDFILE u
ENDFILE ([UNIT=]u[,IOSTAT=ios][,ERR=s])
REWIND u
REWIND ([UNIT=]u[,IOSTAT=ios][,ERR=s])
SKIPFILE u
SKIPFILE (u[,COUNT=m][,ERR=s])
BACKFILE u
BACKFILE (u[,COUNT=m][,ERR=s])
IF (EOF,u) s₁,s₂

The end-of-file record concept is a positional one and appropriately applies only to sequential files, where end-of-file records are encountered during a read. Changes to these statements are not anticipated in the near future. Some, that are non-standard conforming, may be removed from the compiler as users turn to the new syntax. File positioning statements may not be used on a file currently open.
In order to maintain the possibility of future growth in the area of data structures, and to conform to the standard, the file positioning statements have been disallowed in the random access FORTRAN statements implemented. An example is the REWIND u, which positions a sequential file or files to the volume load point. An OPEN statement will provide the user access to the beginning (first record) of a named dataset. A BACKSPACE will also cross dataset boundaries just as it has in the past. This is an extension to the standard. Notice that the END=s parameter may only be used with sequential files. For direct access, an ERR=s specifier would note that a user was attempting to cross a named dataset boundary.
Parameters in the forms used in this section are defined as follows (square brackets indicate optional items):

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><em>u</em></td>
<td>unit number, unsigned integer constant or variable</td>
</tr>
<tr>
<td><em>s</em></td>
<td>a statement label</td>
</tr>
<tr>
<td><em>m</em></td>
<td>an integer constant greater than zero</td>
</tr>
<tr>
<td><em>ios</em></td>
<td>an integer variable</td>
</tr>
</tbody>
</table>

The magnetic tape on unit *u* is rewound to the load point. If the tape is already rewound, the statement acts as a do-nothing statement. Tapes must be rewound before the first read.

Keywords in the form are:

**UNIT=*u***  
*u* is an expression specifying the unit. There are two forms for specifying the unit, UNIT=*u* or simply *u*. *u* is the unit number specified on the *VOLUME card.

**IOSTAT=*ios***  
*ios* is an integer variable returned with the error condition that occurred during execution of the REWIND statement. If *ios=0*, the rewind was successful. If *ios≠0*, an error condition is indicated in the right 12 bits of the word.

**ERR=*s***  
Program control transfers to the FORTRAN statement numbered *s* if an error occurs.
BACKSPACE u or
BACKSPACE ([UNIT=]u [,ISTAT=ios] [,ERR=s])

The magnetic tape on unit u is backspaced one logical record.
If a tape is at load point (rewound), this statement acts as
a do-nothing statement.

Keywords in the form are:
UNIT=u u is an expression specifying the unit.
There are two forms for specifying the
unit, UNIT=u or simply u. u is the unit
number specified on the *VOLUME card.

ISTAT=ios ios is an integer variable returned with
the error condition that occurred during
execution of the BACKSPACE statement. If
ios=0, the backspace was successful. If
ios≠0, the error condition is indicated in
the right 12 bits of the word.

ERR=s Program control transfers to the FORTRAN
statement numbered s if an error occurs.

ENDFILE u or
ENDFILE ([UNIT=]u [,ISTAT=ios] [,ERR=s])

This statement writes a tape mark on magnetic tape unit u.

Keywords in the form are:
UNIT=u u is an expression specifying the unit.
There are two forms for specifying the
unit, UNIT=u or simply u. u is the unit
number specified on the *VOLUME card.

ISTAT=ios ios is an integer variable returned with
the error condition that occurred during
execution of the ENDFILE statement. If
ios=0, the endfile was successful. If
ios≠0, the error condition is indicated in
the right 12 bits of the word.

ERR=s Program control transfers to the FORTRAN
statement numbered s if an error occurs.
9.62
Input/Output Statements

BACKFILE u or
BACKFILE (u [,COUNT=m] [,ERR=s])

The magnetic tape on unit u is repositioned backward one or more files.

Keywords in the form are:

- **u**: u is an expression specifying the unit, and must match the unit number on the VOLUME card.
- **COUNT=m**: m is the number of files to be backspaced. If there is no preceding file, the unit is positioned at its initial point. If n=0, no repositioning occurs.
- **ERR=s**: Program control transfers to the FORTRAN statement numbered s if an error occurs.

**Example 1**

BACKFILE 9

**Example 2**

BACKFILE (9,COUNT=3,ERR=201)

BACKFILE (9,COUNT=3)
**Input/Output Statements**

**SKIPFILE u** or **SKIPFILE (u [,COUNT=m] [,ERR=s])**

The magnetic tape on unit u is repositioned forward one or more files.

Keywords in the form are:

- **u**: u is an expression specifying the unit, and must match the unit number on the VOLUME card.
- **COUNT=m**: m is the number of files to be forward spaced. Do not use SKIPFILE if there are no more file marks on the tape. Proceeding to the end of tape may use large amounts of peripheral processor time.
- **ERR=s**: Program control transfers to the FORTRAN statement numbered s if an error occurs.

**Example 1**

```
SKIPFILE 9
```

<table>
<thead>
<tr>
<th>Initial Position ((\checkmark))</th>
<th>Final Position ((\checkmark))</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File</strong></td>
<td><strong>File</strong></td>
</tr>
<tr>
<td><strong>Mark</strong></td>
<td><strong>Mark</strong></td>
</tr>
<tr>
<td>record</td>
<td>record</td>
</tr>
<tr>
<td>record</td>
<td>record</td>
</tr>
</tbody>
</table>

**Example 2**

```
SKIPFILE (9,COUNT=3,ERR=201)
SKIPFILE (9,COUNT=3)
```

<table>
<thead>
<tr>
<th>Initial Position ((\checkmark))</th>
<th>Final Position ((\checkmark))</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File</strong></td>
<td><strong>File</strong></td>
</tr>
<tr>
<td><strong>Mark</strong></td>
<td><strong>Mark</strong></td>
</tr>
<tr>
<td>record(s)</td>
<td>record(s)</td>
</tr>
<tr>
<td>record(s)</td>
<td>record(s)</td>
</tr>
</tbody>
</table>

---
IF(EOF,u)s_1,s_2 A check of the previous read operation is made to determine if an end-of-file has been encountered on unit u. If it has, control is transferred to statement s_1; if not, control is transferred to statement s_2. It may not be used to check an EOF on units 5 or 6, the standard reader and printer units. (Note that BUFFER IN can be used with even parity to read from the card reader. In this case the IF(UNIT,u) test may be used to check for an EOF.)
EXAMPLE USING
SEQUENTIAL ACCESS

*VOLUME,7,VSN=D0001,STAGEIN=NS,STAGEOUT=MD
*FORTRAN,FL

PROGRAM DEM1
DIMENSION B(6)

C

USE SEQUENTIAL ACCESS AND SYNCHRONOUS WRITE TO CREATE A FILE OF ONE DATASET HAVING TEN RECORDS

OPEN(UNIT=7,STATUS='NEW')
DO 10 IREC=1,10
READ(5,1000) B
WRITE(UNIT=7) B
10 CONTINUE
CLOSE(UNIT=7)

C

OPEN FILE WITH SEQUENTIAL ACCESS AND READ ALL RECORDS

OPEN(UNIT=7,STATUS='OLD')
DO 20 IREC=1,10
READ(UNIT=7) B
20 CONTINUE
CLOSE(UNIT=7)

C

1000 FORMAT(6F10.1)
END
Program DEM1 creates two copies of the same file, one on disk (during execution time) and one on MSD (during job termination phase).

Many of the permissible arguments in the OPEN, CLOSE, READ and WRITE statements have not been specified in the above calls, thereby assuming default values.

In creating the above files, defaults are used to establish the file name, type of access, and mode of writing on unit 7. The following call is equivalent to the first OPEN call on program DEM1:

```
OPEN(UNIT=7,FILE=17HNCARSYSTEMHD10001,FILESEQ=1,
     FORM='UNFORMATTED',STATUS='NEW',RECL=6,
     ACCESS='SEQUENTIAL',GEN=1,EXPDT=78093,BLANK='ZERO'
```

Because sequential access is used in program DEM1, specification of argument RECL is not required in the first OPEN call.

After execution, the status of the created file is 'KEEP', due to the status default in the last CLOSE call.
EXAMPLE USING DIRECT ACCESS

*VOLUME,7,VSN=DD0002,STAGEIN=NS,CONV=LG
*FORTRAN,FL

PROGRAM DEM2
DIMENSION DATA(6)
C
C USE DIRECT ACCESS AND SYNCHRONOUS WRITE TO CREATE
C A FILE OF ONE DATASET HAVING FIVE RECORDS
C
OPEN(UNIT=7,RECL=11,STATUS='NEW',MAXREC=25,ACCESS='DIRECT')
DO 10 IREC=1,5
READ(5,1000) IMONTH,IDAY,IYEAR,DATA
WRITE(UNIT=7,REC=6-IREC) IMONTH,IDAY,IYEAR,DATA
10 CONTINUE
CLOSE(UNIT=7)
C
C ACCESS FOURTH RECORD DIRECTLY
C
OPEN(UNIT=7,STATUS='OLD',ACCESS='DIRECT')
READ(UNIT=7,REC=4) IMONTH,IDAY,IYEAR,DATA
CLOSE(UNIT=7)
C
1000 FORMAT(3 2,4X,6F10.2)
END

COMMENTS

- RECL must be specified in the first OPEN call in program DEM2, since direct access and new status are used.

- In the first OPEN call, MAXREC is set to 25 even though only five records are written in this run. This is done in anticipation of adding 20 more records to the dataset at a later date. Similarly, RECL may be set to accommodate future expansion of record length.

- The RECL argument of the WRITE statement allows the records to be numbered independently of the order in which they are generated. In program DEM2, the numbering is reversed.
EXEMPLARY USING DIRECT AND SEQUENTIAL ACCESS ON THE SAME UNIT

```
*VOLUME,7,VSN=D0001,STAGEIN=MA,STAGEOUT=MD,CONV=LG  
*FORTRAN,FL
  PROGRAM DEM3
  DIMENSION B(6)
  OPEN FILE D0001 FROM PROGRAM DEM1 WITH DIRECT ACCESS
  OPEN(UNIT=7,STATUS='OLD',ACCESS='DIRECT')
  READ(UNIT=7,REC=4) B
  ADD 1. TO EACH DATA ELEMENT IN RECORD 4 AND REWRITE RECORD
  DO 20 I=1,6
     B(I)=B(I)+1.
  20 CONTINUE
  WRITE(UNIT=7,REC=4) B
  CLOSE(UNIT=7)
  OPEN FILE WITH SEQUENTIAL ACCESS AND SUM DATA ELEMENTS OVER ALL RECORDS
  OPEN(UNIT=7,STATUS='OLD',ACCESS='SEQUENTIAL')
  SUM=0.
  DO 30 IREC=1,10
     READ(UNIT=7) B
     DO 40 I=1,6
        SUM=SUM+B(I)
     40 CONTINUE
  30 CONTINUE
  CLOSE(UNIT=7)
  WRITE(6,1000) SUM
  1000 FORMAT(IH*SUM =*,F10.1)
END
```

COMMENTS

- The file with VSN=D0001 was created in program DEM1 using sequential access.

- Program DEM3 modifies the file created by program DEM1. To avoid the problem of the disk copy not matching the TMS-4 copy, STAGEOUT=MD is used on the *VOLUME card.
EXAMPLE OF USER CATALOG PROGRAM

```
*SEQUENCE, 990934
7600 RUN
DATE 02/08/78
TIME 02/25/04
2 JOB, 8000, 44340002, ADAMS
3 VOLUME, 9, VSN=PD453, STAGEIN=MA
```

**INPUT -- FORTRAN, FL**

<table>
<thead>
<tr>
<th>CARD NUMBER</th>
<th>APPROXIMATE PROGRAM LOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PROGRAM OEM4</td>
</tr>
<tr>
<td>2</td>
<td>DIMENSION NAME(2)</td>
</tr>
<tr>
<td>3</td>
<td>DIMENSION IDATA(10)</td>
</tr>
<tr>
<td>4</td>
<td>LOGICAL EXIST, OPENED, NAME</td>
</tr>
<tr>
<td>5</td>
<td>IGEN, EXPDT, VSN, CRECL, BLANK</td>
</tr>
<tr>
<td>6</td>
<td>PRINT 100</td>
</tr>
<tr>
<td>7</td>
<td>100 FORMAT(1H120X<em>INQUIRE CATALOG</em>)</td>
</tr>
<tr>
<td>8</td>
<td>DO 2 I=1,600</td>
</tr>
<tr>
<td>9</td>
<td>OPEN (UNIT=9, FILESEQ=I, ERR=90, IOSTAT=IOSTAT, ACCESS=#SEQUENTIAL#)</td>
</tr>
<tr>
<td>10</td>
<td>DO 20 II=1, I</td>
</tr>
<tr>
<td>11</td>
<td>READ(9, NWORDS=3) IDATA</td>
</tr>
<tr>
<td>12</td>
<td>20 CONTINUE</td>
</tr>
<tr>
<td>13</td>
<td>INQUIRE (UNIT=9, FILESEQ=I, ERR=91, IOSTAT=IOSTAT, EXIST=EXIST,</td>
</tr>
<tr>
<td>14</td>
<td>OPENED=OPENED, NAMED=NAMED, NUMBER=NUMBER, NAME=NAME, CRECL=CRECL,</td>
</tr>
<tr>
<td>15</td>
<td>MAXREC=MAXREC, ACCESS=ACCESS, SEQUENTIAL=SEQUENTIAL, DIRECT=DIRECT,</td>
</tr>
<tr>
<td>16</td>
<td>FORM=FORM, FORMATTED=FORMATTED, UNFORMATTED=UNFORMATTED, NSTATE=NSTATE,</td>
</tr>
<tr>
<td>17</td>
<td>MODE=MODE, NEXTREC=NEXTREC, SEQFIL=SEQFIL, GEN=GEN, EXPDT=EXPDT,</td>
</tr>
<tr>
<td>18</td>
<td>VSN=VSN, NUMREC=NUMREC, CRECL=CRECL, BLANK=BLANK</td>
</tr>
<tr>
<td>19</td>
<td>IF (.NOT.EXIST) GO TO 92</td>
</tr>
<tr>
<td>20</td>
<td>PRINT 102, NAME=NSTATE, NUMBER</td>
</tr>
<tr>
<td>21</td>
<td>102 FORMAT(///1X, ,2A10)</td>
</tr>
<tr>
<td>22</td>
<td>PRINT 101, IOSTAT, EXIST, OPENED, NAMED, CRECL, MAXREC,</td>
</tr>
<tr>
<td>23</td>
<td>IACCESS, DIRECT, FORM, FORMAT, UNFORM, MODE,</td>
</tr>
<tr>
<td>24</td>
<td>2NEXTREC, SEQFIL, GEN, EXPDT, VSN, NUMREC, CRECL, BLANK</td>
</tr>
<tr>
<td>25</td>
<td>21 FORMAT (1X,#IOSTAT=I04, #EXIST=I2, 12X#OPENED=I2, 12X#NAMED=I2</td>
</tr>
<tr>
<td>26</td>
<td>21 IF (EXIST) THEN 17, 6X#MAXREC=I9, 2X#ACCESS=A10, 4X#SEQUENTIAL=A13/</td>
</tr>
<tr>
<td>27</td>
<td>21 DIRECT=I10, 1X#FORMATTED=A10, 3X#UNFORMATTED=A10</td>
</tr>
<tr>
<td>28</td>
<td>21 IX#MODE=I12, 11X#NEXTREC=I12, 11X#SEQFIL=I14, 9X#GEN=I14</td>
</tr>
<tr>
<td>29</td>
<td>4/I#EXPDT=I15, 7X#VSN=I6, 8X#NUMREC=I19, 5X#CRECL=I18, 4X</td>
</tr>
<tr>
<td>30</td>
<td>5X#BLANK=AA4)</td>
</tr>
<tr>
<td>31</td>
<td>2 CONTINUE</td>
</tr>
<tr>
<td>32</td>
<td>90 STOP 90</td>
</tr>
<tr>
<td>33</td>
<td>91 STOP 91</td>
</tr>
<tr>
<td>34</td>
<td>92 STOP 92</td>
</tr>
<tr>
<td>35</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>END</td>
</tr>
</tbody>
</table>

LENGTH OF ROUTINE: 394

**VARIABLE ASSIGNMENTS**

<table>
<thead>
<tr>
<th>NAME</th>
<th>IDATA</th>
<th>IOSTAT</th>
<th>EXPDT</th>
<th>NSTATE</th>
<th>NUMREC</th>
</tr>
</thead>
<tbody>
<tr>
<td>VSN</td>
<td>224</td>
<td>225</td>
<td>226</td>
<td>245</td>
<td>233</td>
</tr>
<tr>
<td>ERR</td>
<td>254</td>
<td>253</td>
<td>252</td>
<td>244</td>
<td>243</td>
</tr>
<tr>
<td>NUMBER</td>
<td>246</td>
<td>245</td>
<td>244</td>
<td>243</td>
<td>242</td>
</tr>
</tbody>
</table>

**SUBROUTINES CALLED**

<table>
<thead>
<tr>
<th>OUTPTC</th>
<th>QQQAEM3</th>
<th>QQQAMB</th>
<th>QQQAEM2</th>
<th>STOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPILE TIME = 63 MILLISECS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**INPUT -- RUN, I**

```
*FORTRAN, FL
```

**9.69 Input/Output Statements**
**EXAMPLE OF USER CATALOG PROGRAM**

(continued)

PROGRAM SPACE IS 3253

<table>
<thead>
<tr>
<th>ORIGIN ENTRY POINTS AND LOCATIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 DEM4 20</td>
</tr>
<tr>
<td>360 Q08ERR 360</td>
</tr>
<tr>
<td>373 Q06I0U 373 IU903 616</td>
</tr>
<tr>
<td>627 Q08AMB3 627</td>
</tr>
<tr>
<td>740 EXIT 768 END 763 STOP 743</td>
</tr>
<tr>
<td>760 Q08AMB2 768</td>
</tr>
<tr>
<td>1054 K0DER 1054</td>
</tr>
<tr>
<td>2157 Q08AM91 2157</td>
</tr>
<tr>
<td>2453 OUTPTC 2463 QTPTER 3146</td>
</tr>
<tr>
<td>3162 Q0GRD0 3162 Q0GRWTB 3169</td>
</tr>
</tbody>
</table>

OVERALL CORE USE STATISTICS (DECIMAL)

THE MAXIMUM SCM IS 54272
THE PROGRAM CURRENTLY USES 1723
THE LOADING PROCESS USED 492822
THE MAXIMUM LCM IS 452622
THE PROGRAM CURRENTLY USES 54924

LCM AREA MAP BY BUFFER TYPE

<table>
<thead>
<tr>
<th>MISC 185</th>
<th>SYSI0U 24</th>
<th>RANFE 64</th>
<th>RD 2240</th>
<th>PR 2240</th>
<th>DISK 8384</th>
<th>SSCR7 3162</th>
<th>SYSL80 256</th>
<th>SYSL81 1024</th>
<th>SYSL82 1024</th>
<th>SYSL87 1324</th>
<th>SYSL3IC 8192</th>
<th>SYSSAT 512</th>
<th>SYSDM1 1024</th>
</tr>
</thead>
</table>

INQUIRE CATALOG

PRCCTDATA
NSTATE=00000007400100000003 ON UNIT 9

<table>
<thead>
<tr>
<th>IOSTAT=0000</th>
<th>EXIST= T</th>
<th>OPENED= T</th>
<th>NAMED= T</th>
</tr>
</thead>
<tbody>
<tr>
<td>RECL= 0</td>
<td>MAXREC= 0</td>
<td>ACCESS=SEQUENTIAL</td>
<td>SEQUENTIAL= YES</td>
</tr>
<tr>
<td>DIRECT=YES</td>
<td>FORM=UNIFORM</td>
<td>FORMATTED= YES</td>
<td>UNFORMATTED= YES</td>
</tr>
<tr>
<td>MODE= 0</td>
<td>NEXTREC= 2</td>
<td>SEQFILE= 1</td>
<td>GEN= 0</td>
</tr>
<tr>
<td>EXPNO=78288</td>
<td>VSN=P04323</td>
<td>NUMREC= 122</td>
<td>CRECL= 8</td>
</tr>
</tbody>
</table>

MCSTATS
NSTATE=00000007400100000003 ON UNIT 9

<table>
<thead>
<tr>
<th>IOSTAT=0000</th>
<th>EXIST= T</th>
<th>OPENED= T</th>
<th>NAMED= T</th>
</tr>
</thead>
<tbody>
<tr>
<td>RECL= 0</td>
<td>MAXREC= 0</td>
<td>ACCESS=SEQUENTIAL</td>
<td>SEQUENTIAL= YES</td>
</tr>
<tr>
<td>DIRECT=YES</td>
<td>FORM=UNIFORM</td>
<td>FORMATTED= YES</td>
<td>UNFORMATTED= YES</td>
</tr>
<tr>
<td>MODE= 0</td>
<td>NEXTREC= 3</td>
<td>SEQFILE= 2</td>
<td>GEN= 0</td>
</tr>
<tr>
<td>EXPNO=78288</td>
<td>VSN=P04323</td>
<td>NUMREC= 16</td>
<td>CRECL= 8</td>
</tr>
</tbody>
</table>

MDMPLSTT
NSTATE=00000007400100000003 ON UNIT 9

<table>
<thead>
<tr>
<th>IOSTAT=0000</th>
<th>EXIST= T</th>
<th>OPENED= T</th>
<th>NAMED= T</th>
</tr>
</thead>
<tbody>
<tr>
<td>RECL= 0</td>
<td>MAXREC= 0</td>
<td>ACCESS=SEQUENTIAL</td>
<td>SEQUENTIAL= YES</td>
</tr>
<tr>
<td>DIRECT=YES</td>
<td>FORM=UNIFORM</td>
<td>FORMATTED= YES</td>
<td>UNFORMATTED= YES</td>
</tr>
<tr>
<td>MODE= 0</td>
<td>NEXTREC= 4</td>
<td>SEQFILE= 3</td>
<td>GEN= 0</td>
</tr>
<tr>
<td>EXPNO=78288</td>
<td>VSN=P04323</td>
<td>NUMREC= 140</td>
<td>CRECL= 8</td>
</tr>
</tbody>
</table>

* The BLANK= specifier is not yet implemented.
*The BLANK= specifier is not yet implemented.*
EXAMPLE OF USER CATALOG PROGRAM
(continued)

*The BLANK= specifier is not yet implemented.
### SUMMARY TABLE OF FORTRAN SYNTAX

#### Synchronous Input/Output Statements

- **READ**
  - `READ ([UNIT=]u[,,[FMT=]f][,[REC=]rn][,[ERR=s]][,END=s][,IOSTAT=ios]) iolist`
- **WRITE**
  - `WRITE ([UNIT=]u[,,[FMT=]f][,[REC=]rn][,MODE=md][,[ERR=s]][,IOSTAT=ios]) iolist`
- **READ**
  - `READ f,iolist`
- **PRINT**
  - `PRINT f,iolist`
- **PUNCH**
  - `PUNCH f,iolist`
- **WRITE**
  - `(u,f)iolist`
- **READ**
  - `(u,f)iolist`

#### Asynchronous Input/Output Statements

- **READ**
  - `READ ([UNIT=]u[,REC=rn],NWORDS=n[,NSTATE=ns]) fwa`
- **WRITE**
  - `WRITE ([UNIT=]u[,REC=rn][,MODE=md],NWORDS=n[,NSTATE=ns]) fwa`
- **BUFFER IN**
  - `(u,p)(fwa,lwa)`
- **BUFFER OUT**
  - `(u,p)(fwa,lwa)`

#### ENCODE/DECODE Statements

- **ENCODE**
  - `(cc,f,v)iolist`
- **DECODE**
  - `(cc,f,v)iolist`

#### Dataset Support Statements

- **OPEN**
  - `OPEN ([UNIT=]u[,FILE=dsi][,FILESEQ=n[,ERR=s][,FORM=fm][,STATUS=sta][,RECL=rlen]
    [,MAXREC=maxr][,ACCESS=acc][,GEN=gen][,EXPDT=exp][,PASS=pass][,IOSTAT=ios]
    [,BLANK=blnk])`
- **CLOSE**
  - `CLOSE ([UNIT=]u[,STATUS=sta][,ERR=s][,IOSTAT=ios])`
- **INQUIRE**
  - `INQUIRE ([UNIT=]u[,FILE=dsi][,FILESEQ=n][,ERR=s][,IOSTAT=ios][,EXIST=ex][,OPENED=od]
    [,NAMED=nmd][,NUMBER=r][,NAME=fn][,RECL=rlen][,MAXREC=maxr][,ACCESS=acc]
    [,SEQUENTIAL=seq][,DIRECT=dir][,FORM=fmt][,FORMATTED=fmt][,UNFORMATTED=unf]
    [,NSTATE=ns][,MODE=md][,NEXTREC=nr][,SEQFIL=n][,GEN=gen][,EXPDT=exp][,VSN=name]
    [,NUMREC=nr][,CRECL=c1][,GENTIME=gn][,BLANK=blnk][,CRDT=crdt])`
- **SETPASS**
  - `SETPASS ([UNIT=]u,PASS=pass[,RWPASS=rwpass][,RPPASS=rpass][,WPPASS=wpass][,K=LENGTHF(u)]
    [,(UNIT,u)S_1,S_2,S_3,S_4])`
SUMMARY TABLE OF FORTRAN SYNTAX
(continued)

File Positioning Statements
BACKSPACE u
BACKSPACE ([UNIT=]u[,IOSTAT=ios][,ERR=s])
ENDFILE u
ENDFILE ([UNIT=]u[,IOSTAT=ios][,ERR=s])
REWIND u
REWIND ([UNIT=]u[,IOSTAT=ios][,ERR=s])
SKIPFILE u
SKIPFILE (u[,COUNT=m][,ERR=s])
BACKFILE u
BACKFILE (u[,COUNT=m][,ERR=s])
IF(EOF,u)s1,s2
INTRODUCTION

The FORTRAN statement list includes allowable forms which statements may take using NCAR FORTRAN, CRAY-1 FORTRAN, FORTRAN 66 and FORTRAN 77. The list summarizes a FORTRAN vocabulary in each case. The statements have been subdivided into several categories: Program and subprogram statements, subprogram reference and transfer, arithmetic statement function, type declaration, storage allocation, replacement, branching and IF logic, looping and control statements, format, data input, data output, file positioning, core to core transfer, and dataset support.

The CRAY FORTRAN Manual, published by Cray Research, Inc., describes the FORTRAN compiler in use on the CRAY-1 computer. These statements are included for comparison with NCAR 7600 FORTRAN and the older and current standards. The symbols used by NCAR FORTRAN have been used in all of the statement lists. A table of symbol definitions appears at the end of this chapter.
NCAR FORTRAN

PROGRAM AND SUBPROGRAM STATEMENTS: (non-executable)

PROGRAM pgm
SUBROUTINE sub [(d1[,d2]...)]
[typ] FUNCTION fun (d1[,d2]...)
-----
BLOCK DATA [sub]
ENTRY en
EXTERNAL en1[,en2]...

SUBPROGRAM REFERENCE AND TRANSFER: (executable)

CALL en [(a1[,a2]...)]
RETURN
fun (a1[,a2]...)

ARITHMETIC STATEMENT FUNCTION: (definition)

fun (d1[,d2]...) = exp

TYPE DECLARATION: (non-executable)

COMPLEX nlist
DOUBLE PRECISION nlist
REAL nlist
INTEGER nlist
LOGICAL nlist
IMPLICIT typ (let1[,let2]...)
[typ (let3[,let4]...)]...
PARAMETER (param1=exp1[,param2=exp2]...)

-----

CRAY FORTRAN

PROGRAM pgm
SUBROUTINE sub [(d1[,d2]...)]
[typ] FUNCTION fun (d1[,d2]...)
-----
BLOCK DATA [sub]
-----
EXTERNAL en1[,en2]...

CALL en [(a1[,a2]...)]
RETURN
fun (a1[,a2]...)

fun (d1[,d2]...) = exp

COMPLEX nlist
DOUBLE PRECISION nlist
REAL nlist
INTEGER nlist
LOGICAL nlist
IMPLICIT typ (let1[,let2]...)
[typ (let3[,let4]...)]...
PARAMETER (param1=exp1[,param2=exp2]...)

-----
FORTRAN 66

PROGRAM AND SUBPROGRAM STATEMENTS: (non-executable)

```fortran
PROGRAM pgm
SUBROUTINE sub [(d1[,d2]...)] [typ] FUNCTION fun (d1[,d2]...)
BLOCK DATA
EXTERNAL en1[,en2]...

EXTERNAL en1[,en2]...
```

SUBPROGRAM REFERENCE AND TRANSFER: (executable)

```fortran
CALL en [(a1[,a2]...)]
RETURN
fun (a1[,a2]...)

CALL en [(a1[,a2]...)]
RETURN [exp]
fun ([a1[,a2]...])
```

ARITHMETIC STATEMENT FUNCTION: (definition)

```fortran
fun (d1[,d2]...) = exp

fun ([d1[,d2]...]) = exp
```

TYPE DECLARATION: (non-executable)

```fortran
COMPLEX nlist
DOUBLE PRECISION nlist
REAL nlist
INTEGER nlist
LOGICAL nlist

PARAMETER (param1=exp1[,param2=exp2]...)

INTRINSIC fun1[,fun2]...

CHARACTER [*len[,"]nam1[,nam2]...
```
10.4
FORTRAN Statement List

NCAR FORTRAN

STORAGE ALLOCATION: (non-executable)
DIMENSION v_1(n_1[,n_2][,n_3])
   [,v_2(n_1[,n_2][,n_3])]...
COMMON [/blk_1/]nlist[/blk_2/nlist]...
DATA klist_1/clist_1/
   [,klist_2/clist_2/]...
EQUIVALENCE (nlist)[,(nlist)]...

-----

REPLACEMENT: (executable)
x = arithmetic expression
x = relational/logical expression
x = masking expression

-----

ASSIGN s to lab

BRANCHING AND IF LOGIC: (executable)
GO TO s
GO TO lab [, (s_1[,s_2]...)]
GO TO (s_1[,s_2]...)[,]i
IF(exp) s_1,s_2,s_3
IF(exp) s_1,s_2
IF(exp) st
ELSE
ELSE IF(exp) THEN
END IF
IF(exp) THEN

CRAY FORTRAN

STORAGE ALLOCATION: (non-executable)
DIMENSION v_1(n_1[,n_2][,n_3]...[,n_7])
   [,v_2(n_1[,n_2][,n_3]...[,n_7])]...
COMMON [/blk_1/]nlist[/blk_2/nlist]...
DATA klist_1/[j*]dlist_1/
   [,klist_2/[j*]dlist_2/]...
EQUIVALENCE (nlist)[,(nlist)]...

-----

REPLACEMENT: (executable)
x = arithmetic expression
x = relational/logical expression
x = masking expression

-----

ASSIGN s to lab

BRANCHING AND IF LOGIC: (executable)
GO TO s
GO TO lab [, (s_1[,s_2]...)]
GO TO (s_1[,s_2]...)[,]i
IF(exp) s_1,s_2,s_3
IF(exp) s_1,s_2
IF(exp) st
ELSE
ELSE IF(exp) THEN
END IF
IF(exp) THEN
FORTRAN 66

**STORAGE ALLOCATION:** (non-executable)

```
DIMENSION v1(n1[,n2][,n3])
    [,v2(n1[,n2][,n3])]...
COMMON [/[blk1]/nlist[/[blk2]/nlist]... COMMON [/[blk1]/nlist[[,]/[blk2]/nlist]...
DATA klist1[[j*]dlist1/ [,klist2[[j*]dlist2]...
EQUIVALENCE (nlist)[,nlist]...
```

---

**REPLACEMENT:** (executable)

```
x = arithmetic expression
x = relational/logical expression
```

---

```
ASSIGN s to lab
```

---

**BRANCHING AND IF LOGIC:** (executable)

```
GO TO s
GO TO lab [,,(s1[,s2]...)]
GO TO (s1[,s2]),i
IF(exp) s1,s2,s3
-------
IF(exp) st
-------
-------
-------
```

FORTRAN 77

**STORAGE ALLOCATION:**

```
DIMENSION v1(n1[,n2][,n3]...[,n7])
    [,v2(n1[,n2][,n3]...[,n7])]...
COMMON [/[blk1]/nlist[[,]/[blk2]/nlist]... COMMON [/[blk1]/nlist[[,]/[blk2]/nlist]...
DATA klist1/[list1/
    [,klist2[/list2]/...]
EQUIVALENCE (nlist)[,nlist]...
SAVE [a1[,a2]...]
```

---

**REPLACEMENT:** (executable)

```
x = arithmetic expression
x = relational/logical expression
```

---

```
ASSIGN s to lab
```

---

**BRANCHING AND IF LOGIC:** (executable)

```
GO TO s
GO TO lab [[,](s1[,s2]...)]
GO TO (s1[,s2])[,i
IF(exp)s1,s2,s3
-------
IF(exp) st
ELSE
ELSE IF(exp) THEN
END IF
IF(exp) THEN
```
NCAR FORTRAN

LOOPING AND CONTROL STATEMENTS: (executable)

DO s[i] = exp1,exp2[,exp3] 
CONTINUE 
PAUSE [string] 
STOP [string] 
END

FORMAT: (non-executable)

FORMAT (spec) 

DATA INPUT: (executable)

READ f[,iolist] 
READ (u,f)[iolist] 
READ (u)[iolist] 
READ (ciston)[iolist] 
READ (rlist) fwa 
BUFFER IN (u,p)(fwa,lwa)

DATA OUTPUT: (executable)

PRINT f[,iolist] 
PUNCH f[,iolist] 
WRITE (u,f)[iolist] 
WRITE (u)[iolist] 
WRITE (ciston)[iolist] 
WRITE (wlist) fwa 
BUFFER OUT (u,p)(fwa,lwa)

CRAY FORTRAN

DO s i = m1,m2[,m3] 
CONTINUE 
PAUSE [string] 
STOP [string] 
END

FORMAT (spec) 

READ f,iolist 
READ (u,f)iolist 
READ (u)iolist 
----- 
----- 
BUFFER IN (u,p)(fwa,lwa)

PRINT f,iolist 
PUNCH f,iolist 
WRITE (u,f)iolist 
WRITE (u)iolist 
----- 
----- 
BUFFER OUT (u,ms)(fwa,lwa)
FORTRAN 66

LOOPING AND CONTROL STATEMENTS: (executable)

DO s i = m1,m2[,m3]
CONTINUE
PAUSE [string]
STOP [string]
END

FORMAT: (non-executable)

FORMAT (spec)

DATA INPUT: (executable)

-----
READ (u,f)iolist
READ (u)iolist
-----
-----

DATA OUTPUT: (executable)

-----
-----
WRITE (u,f)iolist
WRITE (u)iolist
-----
-----

FORTRAN 77

LOOPING AND CONTROL STATEMENTS: (executable)

DO s [,]i = exp1,exp2[,exp3]
CONTINUE
PAUSE [string]
STOP [string]
END

FORMAT: (spec)

FORMAT (spec)

DATA INPUT: (executable)

READ f[,iolist]
READ (u,f)[iolist]
READ (u)[iolist]
READ (cilist)[iolist]
-----

DATA OUTPUT: (executable)

PRINT f[,iolist]
-----
WRITE (u,f)[iolist]
WRITE (u)[iolist]
WRITE (cilist)[iolist]
NCAR FORTRAN

FILE POSITIONING: (executable)
ENDFILE (alist) ENDFILE u
REWIND (alist) REWIND u
BACKSPACE (alist) BACKSPACE u
BACKFILE (flist) -----
SKIPFILE (flist) -----
IF(EOF,u) s1,s2 IF(IEOF(u)) s1,s2,s3
IF(UNIT,u) s1,s2,s3,s4 IF(UNIT(u)) s1,s2,s3

CORE TO CORE TRANSFER: (executable)
ENCODE (cc,f,v)[iolist] ENCODE (cc,f,v)[iolist]
DECODE (cc,f,v)[iolist] DECODE (cc,f,v)[iolist]

DATASET SUPPORT: (executable)
OPEN (olist) -----
INQUIRE (iulist) -----
CLOSE (cilist) -----
SETPASS (passlist) -----
FORTRAN 66

FILE POSITIONING: *(executable)*

ENDFILE u
REWIND u
BACKSPACE u
-----
-----
-----
-----

CORE TO CORE TRANSFER: *(executable)*

-----
-----

DATASET SUPPORT: *(executable)*

-----
-----
-----
-----

FORTRAN 77

FILE POSITIONING: (alist)

ENDFILE (alist)
REWIND (alist)
BACKSPACE (alist)
-----
-----
-----

CORE TO CORE TRANSFER: *(executable)*

WRITE (corlist)[iolist]
READ (corlist)[iolist]

DATASET SUPPORT: (alist)

OPEN (olist)
INQUIRE (iulist)
INQUIRE (iflist)
CLOSE (cilist)
**Definition of Symbols in Statement Syntax**

a variable or array name; an actual argument expression

alist a list of specifiers containing \([UNIT=u]\); may also contain \(\text{IOSTAT}=\text{ios}\) and/or \(\text{ERR}=\text{s}\)

blk common block identifier

cc character count

cilist control information list

cclist list of constants or symbolic names of constants used in NCAR FORTRAN and FORTRAN 77

coorlist control information list with \(\text{UNIT}=\text{nam}\) (a character variable or array name) and no direct access

d dummy argument

dlist list of constants or symbolic names of constants used in FORTRAN 66 and CRAY FORTRAN

en entry point name

exp an expression

f format statement label or array with Hollerith format data

fclist a list of specifiers: \((u[,\text{COUNT}=n][,\text{ERR}=s])\)

fun function name

fwa first word address of I/O buffer area; an array element or array name

i integer variable

iflist file specifier list which may contain inquiry specifiers

iolist an input/output list; may contain implied DO

iulist external unit specifier which may also contain inquiry specifiers

ios input/output status specifier

j* a repeat factor

klist variable, array identifiers and DO implied lists

lab variable assigned a statement label

len a specifier for the length of a character variable

let a single letter or a range of single letters in alphabetic order
lwa  last word address of I/O buffer area; an array element
m   incrementation parameter of a DO statement
ms  made specifier for full or partial record processing
n   integer constant greater than zero, or symbolic name of such a constant
nam character variable or array element
nlist list of variables, array elements and array names
clist specifier list for OPEN statement
p   parity
param a symbolic name
passlist password specifiers
pgm  a program name
rlist asynchronous READ specifier list
s   statement label
st  a FORTRAN statement
spec format specifiers
string a digit string
sub  a subroutine name
typ  type: REAL, INTEGER, DOUBLE PRECISION, COMPLEX or LOGICAL
u   unit number, unsigned integer constant or variable
v   variable name
wlist asynchronous WRITE specifier list
x   variable or array element
XI

DIAGNOSTICS

ERRORS FATAL TO EXECUTION
ERRORS FATAL TO COMPILATION
NONFATAL ERRORS
Errors in FORTRAN program compilation are pointed out to the user by the compiler in diagnostic messages following a program or subprogram listing. They are found following the END card of the program or subprogram in which the error appears.

The form of the diagnostic message is

<table>
<thead>
<tr>
<th>Card Number</th>
<th>FORTRAN Error Message</th>
<th>Compiler Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>SOME DIAGNOSTIC MESSAGE</td>
<td>m</td>
</tr>
</tbody>
</table>

The first column appearing on the FORTRAN program contains the card number. In the diagnostic message, n refers to the first card of the statement in error. m, the compiler location, is used for system debugging.
Example

COMPILATION DIAGNOSTICS

(continued)

CARD NUMBER  APPROXIMATE  PROGRAM HEAT

1          0           C  PROGRAM NEXT ILLUSTRATES THE USE OF FORTRAN ERROR MESSAGES
2          0           C
3          0           101 READ 1001 X * EP4 + K
4          0           WRITE (6+1005); X, EP4 + K
5          0           READ 1004 = I(A(I)); K
6          0           WRITE (6+1006): I(A(I)+X, K)
7          0  102 W(I) = 12 * A(I)
8          0  103 WRITE (6,1005) X, EPS.
9          0  104 READ (A(I), I=1, K)
10         0  105 WRITE (6,1006) (A(I), I=1, K)
11         0           L = K+1
12         0           DEFINED FOR DERIV
13         0           L = K+1
14         0  106 DO I = 1, K
15         0  107 I2 = I-1
16         0  108 T(I) = I*A(I)
17         0  109 ITERATE MAXIMUM OF 20 TIMES
18         0  110 DO 101, N = 1
19         0  111 FX = A(K)
20         0  112 DO 102, M = K-I
21         0  113 FFx = X*FX*A(M)
22         0  114 FPx = R(K)
23         0  115 DO 103, M = K-I+1
24         0  116 CONTINUE
25         0  117 WRITE (6,1103)
26         0  118 NN = N
27         0  119 X = X - FX/FFx
28         0  120 CONTINUE
29         0  121 WRITE (6,1007)
30         0  122 IF (ABS(FX) .LT. EPS) GO TO 101
31         0  123 NN = N
32         0  124 X = X - FX/FFx
33         0  125 CONTINUE
34         0  126 WRITE (6,1103)
35         0  127 NN = N
36         0  128 X = X - FX/FFx
37         0  129 CONTINUE
38         0  130 WRITE (6,1103)
39         0  131 CONTINUE
40         0  132 WRITE (6,1103)
41         0  133 CONTINUE
42         0  134 WRITE (6,1103)
43         0  135 CONTINUE
44         0  136 WRITE (6,1103)
45         0  137 CONTINUE
46         0  138 WRITE (6,1103)
47         0  139 CONTINUE

LENGTH OF PROGRAM: 170

VARIABLE ASSIGNMENTS

X = 140 EPS = 137 K = 136 I = 135 L = 134 N = 133
N = 132 FX = 131 M = 130 FPX = 127 FX = 126 N = 125

SUBROUTINES CALLED

INPUT QUITC H EXIT

COMPILE TIME = 714 MILLISECONDS

THESE STATEMENTS ARE MISSING

103

Fortran Error Message  Card Number  Compiler Location

7  PARENTHESES USAGE ON DO LOGIC OR TYPE IDENTIFIER IS ILLEGAL IN I/O DATA LIST  41104
8  PARENTHESES USAGE ON DO LOGIC OR TYPE IDENTIFIER IS ILLEGAL IN I/O DATA LIST  41105
13  A SUBSCRIBED VARIABLE HAS NOT BEEN DIMENSIONED  41556
13  ILLEGAL USE OF A REPLACEMENT STATEMENT  42222
25  A DO VARIABLE IS USED IN AN OUTER DO LOOP  47409
28  A DO LOOP WHICH TERMINATES AT THIS STATEMENT INCLUDES AN UNTERMINATED DO  47114
43  A FORMAT SPECIFICATION HAS NO FIELD WIDTH  42363
47  THIS STATEMENT DOES NOT FOLLOW A DO WHICH IT TERMINATES  41616
47  THIS STATEMENT DOES NOT FOLLOW A DO WHICH IT TERMINATES  41617

THE CARD NUMBER IS THE FIRST CARD OF THE STATEMENT IN WHICH THE ERROR OCCURRED.

THE COMPILER LOCATION IS FOR SYSTEM DEBUGGING.
In the example, the diagnostic for card number 7 points to

\begin{verbatim}
READ 1002,(A(I),I=1,K)
\end{verbatim}

The message says that the I/O list is illegal. The reason is that A is not dimensioned. A nondimensioned variable is assumed to be a function subprogram and may not appear in an I/O list.

Card number 13 locates the statement

\begin{verbatim}
DO 102 I=1,K(J)
\end{verbatim}

This is incorrect first because K has not been dimensioned, but further, a subscripted variable may not be a parameter in a DO. The dimension error came up first. The correction is not to dimension K, but to remove the subscript.

Card number 28 is 104 CONTINUE. The message says that an outer DO contains an inner DO not terminated. Statement 113 (card number 23) was mispunched: it should have been 103, which would terminate the DO 103 loop.

Often one FORTRAN mistake will generate more than one diagnostic. Card numbers 25, 28, and 47 relate to the same unterminated DO loop (DO 103 I=1,L).

The example shows that the compiler finds errors in a certain order. One error can cause several diagnostics, none of which necessarily point to the particular card that must be changed. A program is an ordered set of instructions which are interrelated, and the compiler looks at the program as a unified whole which should make sense. Whenever a statement appears which does not fit in with what has been previously set down, an error may be flagged.
Some errors are caused by omission. Several errors in this program were caused by a missing DIMENSION statement.

A list of compiler diagnostics is included below. There are three kinds of errors:

- Those that cause compilation to stop ("fatal to compilation").
- Those that allow compilation to continue, but abort execution of the program ("fatal to execution").
- Those that are merely flags to sloppy programming and do not stop the job ("nonfatal").
A VARIABLE NEEDS DIMENSIONING
AN EQUIVALENCED VARIABLE APPEARS WITH SUBSCRIPTS BUT HAS NOT BEEN DIMENSIONED OR REFERENCED VARIABLE IS NOT THE COMMON VARIABLE
AN UNDEFINED FUNCTION APPEARS WITHIN THIS DO LOOP
ARRAY SIZE DECLARATION ALLOWED IN EITHER DIMENSION OR TYPE STATEMENT, NOT BOTH

COMMON/EQUIVALENCE ERROR (EQVLST1 IN COMLST1)
COMPILER-BLOCK NAME TABLE OVERFLOW
COMPILER-COMMON TABLE OVERFLOW
COMPILER-DIMENSION LIST OVERFLOW
COMPILER-DIMENSION LIST OVERFLOW FROM EXTERNAL STATEMENT
COMPILER-DIMENSION LIST OVERFLOW FROM TYPE ROUTINE
COMPILER-DIMENSION VARIABLE LIST OVERFLOW
COMPILER-EQUIVALENCE LIST OVERFLOW
COMPILER ERROR DATA STATEMENT
COMPILER FAILURE
COMPILER-TYPE TABLE LIMIT EXCEEDED
COMPILER TABLE OVERFLOW
COMPILER TABLE OVERFLOW (PROCESS PI WLIS)

DIMENSION SIZE OF ZERO NOT ALLOWED
DIMENSION TOO LARGE FOR CORE
DIMENSION TOTAL .GT. 65K
EQUIVALENCE ATTEMPTS TO RE-ORDER COMMON
EQUIVALENCE OVERLAPS BLOCK ORIGIN
EQUIVALENCE OVERLAPS BLOCKS

FORMAT POINTER TABLE OVERFLOW
FTN-4 DATA STATEMENT FORMAT ERROR
FTN-4 DATA STATEMENT PRE-SCAN TABLE OVERFLOW
GREATER THAN THREE DIMENSIONS IN DIMENSION STATEMENT
11.6
Diagnostics

ERRORS FATAL TO COMPILATION
(continued)

ILLEGAL EXPONENT
ILLEGAL PROGRAM CARD
IMPROPER DIMENSIONED VARIABLE LIST
IMPROPER FORMAT OF PROGRAM STATEMENT
IMPROPER SUBROUTINE OR FUNCTION STATEMENT OR PARAMETER ERROR

MULTIPLE EQUIVALENCE BETWEEN SAME NAME VARIABLES IS AMBIGUOUS

NON-INTEGER EXPRESSION NOT ALLOWED IN ASA FORTRAN STANDARDS. EXECUTION MAY CONTINUE.

PARENTHESIS MISSING FROM DIMENSIONED TYPE STATEMENT
PROGRAM CARD, SUBROUTINE CARD, OR END CARD MISSING
PROGRAM LENGTH EXCEEDS MEMORY SIZE (1777777B)
PROGRAM OR SUBROUTINE CARD MISSING
PROGRAM OR SUBROUTINE CARD NOT FIRST CARD

STORAGE DEFINITIONS EXCEED COMPILER CAPACITY

TABLE OVERFLOW (ASF)
THIS EQUIVALENCE CAUSES A REORIGIN OF THE COMMON BLOCK
TOO MANY CONSTANTS
TOO MANY INDEX VARIABLES
TOO MANY PARAMETERS IN PARAMETER LIST
TRIED TO EQUATE DIFFERENT VARIABLE TYPES
A CONSTANT IN A COMPLEX PARAMETER IS NOT A REAL NUMBER
A DATA LIST ENTRY MAY NOT BE DEFINED BY FORMAL PARAMETER
A DO EXPRESSION CANNOT BE EVALUATED CORRECTLY
A DO LOOP IMPROPERLY DEFINED
A DO LOOP LIMIT EXCEEDS MACHINE CAPACITY
A DO LOOP MAY NOT TERMINATE AT THE END STATEMENT
A DO LOOP TERMINATES AT THIS STATEMENT
A DO LOOP WHICH TERMINATES AT THIS STATEMENT INCLUDES AN UNTERMINATED DO
A DO OR DO-IMPLIED LOOP OUT OF RANGE
A DO VARIABLE IS USED IN AN OUTER DO LOOP
A DO VARIABLE NOT SET IN DATA STATEMENT
A EXPRESSION FOR A DO PARAMETER IS IMPROPERLY FORMED
A FUNCTION NAME WAS NOT USED AS A REPLACEMENT STATEMENT
A MULTIPLY OPERATOR HAS BEEN INSERTED
A NUMBERED STATEMENT HAS A CONTINUATION MARK
A PARAMETER ARGUMENT MUST BE DEFINED AS A CONSTANT
A PARAMETER NAME WAS PREVIOUSLY DEFINED AS A DIMENSIONED VARIABLE
A PREVIOUS DO TERMINATES ON THIS DO STATEMENT
A REPETITION COUNT IN A DATA STATEMENT CONSTANT LIST MUST BE TYPE INTEGER
A REPLACEMENT CANNOT BE FORMED
A RIGHT PARENTHESES MISSING IN DATA STATEMENT
A SIGN ON A REPETITION COUNT IN A CONSTANT LIST IS MEANINGLESS
A SUBSCRIPTED VARIABLE HAS NOT BEEN DIMENSIONED
A VARIABLE IN DATA STATEMENT HAS VARIABLE DIMENSIONS
A ZERO INCREMENT IN A DO OR DO-IMPLIED LOOP
AN ARRAY NAME PREVIOUSLY USED
AN ENTRY STATEMENT MAY NOT OCCUR INSIDE A DO LOOP
AN EQUAL, SIGN MUST FOLLOW THE DO VARIABLE
AN IDENTIFIER HAS MORE THAN SEVEN CHARACTERS
AN IMPLIED DO VARIABLE MUST BE A NON-SUBSCRIPTED INTEGER VARIABLE
AN INDEX IN DATA LIST VARIABLE IS NOT DEFINED BY AN IMPLIED DO VARIABLE
11.8
Diagnostics

ERRORS FATAL 10
EXECUTION (continued)

AN OCTAL CONSTANT HAS MORE THAN 20 DIGITS
AN OPERATOR IS MISSING OR USED IMPROPERLY
.AND., .OR. MUST BE FOLLOWED BY (. .NOT., OR OPND
ARGUMENT NOT DEFINED IN I/O LIST
ARGUMENTS IN IMPLICIT STATEMENT MUST BE A SINGLE ALPHABETIC LETTER
ARGUMENTS IN IMPLICIT STATEMENT MUST BE ALPHABETIC CHARACTER BETWEEN A AND Z
ARITHMETIC STATEMENT FUNCTION DOUBLY DEFINED
ARITHMETIC STATEMENT FUNCTION PARAMETER LIST NOT ENDED
ARITHMETIC STATEMENT FUNCTION PARAMETERS DO NOT AGREE IN NUMBER
ASF INVOLVES ITSELF
ASF PARAMETER ERROR
ATTEMPT TO DOUBLY TYPE A VARIABLE
ATTEMPT TO PRESET DATA INTO BLANK COMMON

BLANK COMMON MAY NOT BE ASSIGNED BY A DATA STATEMENT
BLANK USED AS A BLOCK NAME MORE THAN ONCE
BRANCH DESTINATION INVALID OR A SEPARATOR IS MISSING
BUFFER TAPE MODE PARAMETER MUST BE SIMPLE INTEGER VARIABLE OR INTEGER CONSTANT

CALLED SUBPROGRAM AND VARIABLE USE SAME NAME
CANNOT FIND END OF IMPLIED DO
CANNOT FIND LIMITING / IN CONSTANT LIST
CLOSE STATEMENT-ERROR IN (STATUS). MUST BE EITHER KEEP OR DELETE
COMMA IN DATA STATEMENT MISSING
COMMA MISSING IN EQUIVALENCE STATEMENT
COMMA MISSING IN OVERLAY STATEMENT
COMPILER - CANNOT PROCESS COMMON STATEMENTS
COMPILER ERROR DATA STATEMENT
COMPILER - TOO MANY ARITHMETIC STATEMENT FUNCTION ENTRIES
COMPILER - TOO MANY FUNCTIONS
COMPILER - TOO MANY IDENTIFIERS
COMPILER - TOO MANY INDEX FUNCTIONS
COMPILER - TOO MANY NON-STANDARD INDICES
CONSTANT LIST SYNTAX ERROR PROBABLY OPERATOR ERROR
CONSTANT LIST SYNTAX ERROR, PROBABLY TOO LONG
CONSTANT OUT OF RANGE
CONTENTS OF DATA AND CONSTANT LIST MUST AGREE IF EITHER IS TYPE COMPLEX
COUNT VALUE PREVIOUSLY DEFINED

DATA LIST AND CONSTANT LIST MUST BE ONE TO ONE
DATA LIST CONTAINS A FORMAL PARAMETER
DATA LIST CONTAINS THE FUNCTION SUBROUTINE NAME
DATA LIST SYNTAX ALLOWS ONLY VARIABLE, ARRAY, OR SUBSCRIPTED VARIABLE
DATA TO TYPE LOGICAL VARIABLE
DATA TO VARIABLE DIMENSIONED ARRAY
DECLARATIVE STATEMENTS MUST PRECEDE ALL ARITHMETIC STATEMENTS
DIMENSIONED ARRAY TOO LARGE FOR CORE
DO INDEX IN DATA STATEMENT NOT STANDARD
DO NOT LABEL AN ENTRY STATEMENT
DOUBLE PRECISION NUMBER TO SINGLE PRECISION VARIABLE
DOUBLY DEFINED VARIABLE NAME IN COMMON
DOUBLY DEFINED VARIABLE OR DIMENSION IN COMMON
DUPLICATE BLOCK NAME
DUPLICATE STATEMENT NUMBER
ENCODE COUNT PARAMETER MUST BE SIMPLE INTEGER VARIABLE OR INTEGER CONSTANT
END CARD MISSING
END OF FILE ERROR FROM INPUT FILE
EQUAL SIGN IN DATA STATEMENT MISSING
EQUAL SIGN MUST DELIMIT KEYWORD
ERROR EXIT PREVIOUSLY SET
ERROR IN ASF SETUP
ERROR STATEMENT LABEL TOO LARGE
EXCESS LEFT PARENTHESIS
EXPRESSION TOO LARGE

FIRST ELEMENT OF BOOLEAN EXP MUST BE OPERAND, (, OR .NOT.
FIRST WORD OF SIGMA STRING NOT AN IDENTIFIER
FORMAL PARAMETER ERROR IN EQUIVALENCE
FORMAL PARAMETER IN COMMON DECLARATION
FORMAT MAY NOT HAVE A VARIABLE INDEX
FORMAT MUST BE SIMPLE INTEGER VARIABLE OR INTEGER CONSTANT
FUNCTION FORMED INCORRECTLY

.GT. # DIMENSIONS FOR VARIABLE IN DATA STATEMENT

IF FORMED INCORRECTLY
IF STATEMENT FORMAT ERROR
ILLEGAL CHARACTER APPEARS IN A CONSTANT
ILLEGAL COMPLEX NO. OR SUBSCRIPTED SUBSCRIPTS
ILLEGAL CONSTANT TYPE
ILLEGAL EXPONENT
ILLEGAL FORMAT IN TYPE STATEMENT
ILLEGAL MARK IN COLUMN SIX
ILLEGAL REPLACEMENT APPEARS IN AN EXPRESSION
ILLEGAL REPLACEMENT IN ARITHMETIC STATEMENT
ILLEGAL SEQUENCE OR USE OF OPERATORS
ILLEGAL SUBSCRIPT IN I/O DATA LIST
ILLEGAL TYPE DECLARED
ILLEGAL USE OF A PERIOD
ILLEGAL USE OF A REPLACEMENT STATEMENT
ILLEGAL USE OF AN OPERATOR
ILLEGAL USE OF FORMAL PARAMETER IN DATA STATEMENT
ILLEGAL VARIABLE LIST FORMAT IN DATA STATEMENT
ILLEGAL VARIABLE NAME IN TYPE STATEMENT
IMPLICIT MUST BE DEFINED BEFORE ALL STATEMENTS EXCEPT PARAMETER
IMPLIED DO DEPTH EXCEEDED
IMPROPER INDEXING IN A DO STATEMENT
IMPROPER LENGTH OF A HOLLERITH CONSTANT
IMPROPER SUBSCRIPT IN AN INPUT OR OUTPUT STATEMENT
IMPROPER TERMINATION OF OVERLAY STATEMENT
IMPROPER USE OF A FUNCTION NAME
IMPROPER USE OF A PROGRAM, SUBROUTINE, OR FUNCTION NAME
IMPROPER USE OF FUNCTION NAME
IMPROPERLY FORMED IMPLIED DO
IN COMPUTED GO-TO ONLY INTEGER, REAL, OR DOUBLE PRECISION EXPRESSIONS ARE VALID
INCORRECT ARITHMETIC SUB-EXP IN BOOLEAN EXP
INCORRECT FORM FOR THE ENTRY STATEMENT
INCORRECT LOGICAL EXPRESSION
INCORRECT RELATIONAL SUB-EXPRESSSION
INPUT OF DATA INTO A CONSTANT IS ILLEGAL
INQUIRE STATEMENT - ERROR IN (EXIST). MUST BE EITHER .TRUE. OR .FALSE.
INQUIRE STATEMENT - ERROR IN (NAME)
INTRINSIC FUNCTION HAS TOO MANY ARGUMENTS
INTRINSIC FUNCTION HAS WRONG NUMBER OF ARGUMENTS
INTRINSIC OR LIBRARY FUNCTION CAN NOT BE DECLARED EXTERNAL
IT IS NOT STANDARD TO INCLUDE DATA FROM LABELED COMMON OTHER IN BLOCK DATA
Diagnostics

ERRORS FATAL TO EXECUTION
(continued)

LABEL MUST BE INTEGER CONSTANT
LEFT PAREN MUST START ARGUMENT LIST
LEFT PAREN OR COMMA MISSING IN DATA STATEMENT
LEFT PARENTHESIS MISSING IN EQUIVALENCE STATEMENT
LEFT PARENTHESIS MISSING IN OVERLAY STATEMENT
LEFT SIDE REPLACEMENT SYMBOL IN PARAMETER MUST BE A SYMBOLIC NAME
LETTER SEQUENCE IN IMPLICIT STATEMENT NOT IN ALPHABETIC ORDER
LIST ITEMS AND CONSTANTS NOT 1 TO 1 CORRESPONDENCE
LOGICAL CONNECTIVE MUST BE FOLLOWED BY ( OR AN OPERAND
LOGICAL OPERATOR INCORRECTLY USED
LOGICAL SUB-EXPRESSION MAY NOT BEGIN WITH AN OPERATOR

MAIN PROGRAMS SHOULD NOT CONTAIN ENTRY STATEMENTS
MASKING EXPRESSIONS MAY NOT CONTAIN ARITHMETIC OPERATORS
MASKING OPNDS MUST BE REAL OR INTEGER
MISSING OR MISPLACED SLASH IN DATA STATEMENT
MIXED MODE-TYPE 5 AND/OR 6 AND/OR 7
MODE MUST BE DEFINED FOR BUFFER STATEMENT
MORE THAN 64 PARAMETERS IN CALL OR FUNCTION
MULTIPLE DATA TO NON-DIMENSIONED VARIABLE

NAME NOT STARTING WITH AN ALPHABETIC CHARACTER
NEGATIVE DO COUNT NOT ALLOWED IN IMPLIED DO
NO PARENTHESIS AFTER ARRAY NAME IN A DIMENSION STATEMENT
NO SEPARATOR IN COMMON STATEMENT
NO SLASH (/) SEPARATOR IN BLOCK DESIGNATION
NO STATEMENT TO DEFINE COUNT
NO TERMINAL PARENTHESIS IN DIMENSION STATEMENT
NO TERMINAL RIGHT PARENTHESIS IN COMMON STATEMENT
NON-CONSTANT DATA IN DATA STATEMENT
NON-CONSTANT IN DATA STATEMENT DO LOGIC
NON-CONSTANT OVERLAY NUMBER
NON-CONSTANT SUBSCRIPT IN COMMON STATE
NON-CONSTANT SUBSCRIPT IN EQUIVALENCE
NON-DIMENSIONED VARIABLE IN DATA LIST
NON-NCAR STANDARD INDEXING
.NOT. MUST BE FOLLOWED BY ( OR OPERAND
NOT STANDARD TO MIX TYPE LOGICAL DATA AND CONSTANT
NUMBERED COMMON NAME .GT. 99999999

OPEN STATEMENT - ERROR IN (ACCESS). VALUE MUST BE DIRECT,
SEQUENTIAL, OR MIXED
OPEN STATEMENT - ERROR IN (EXPDT). MUST BE ALPHANUMERIC DATE
OPEN STATEMENT - ERROR IN (FILE). MUST BE 17 CHARACTERS OR
LESS ALPHANUMERIC CONSTANT OR VAR
OPEN STATEMENT - ERROR IN (FILESEQ)
OPEN STATEMENT - ERROR IN (FORM)
OPEN STATEMENT - ERROR IN (STATUS)
OPERAND MAY BE FOLLOWED BY OPERATOR OR )
OUTPUT OF TYPE 5, 6, OR 7 PROHIBITED
OVERLAY NUMBER .GT. 20B

PARAMETER NAME HAS BEEN PREVIOUSLY DEFINED
PAREN GROUP NOT CLOSED IN PREAMBLE LIST
PARENTHESIS USAGE OR DO LOGIC OR TYPE IDENTIFIER IS ILLEGAL
IN I/O DATA LIST
PARENTHETICAL EXPRESSION VOID
PARITY IN I/O STATEMENT NOT 0 OR 1
POSSIBLE COMPILER ERROR
POSSIBLE MACHINE ERROR

REPEAT COUNT IN DATA STATEMENT OUT OF RANGE
REPLACE NOT A VARIABLE
REPLACEMENT VARIABLE MUST BE REAL OR INTEGER
RETURN NOT ALLOWED IN MAIN PROGRAM
RIGHT PARENTHESIS MISSING IN EQUIVALENCE STATEMENT OR
ILLEGAL SUBSCRIPT
ERRORS FATAL TO EXECUTION
(continued)

SEGMENT NUMBER .GT. 20B
SEGMENT NUMBER MUST BE A CONSTANT
STATEMENT ENDS WITH AN ASTERISK
STATEMENT LABEL INCORRECT
STATEMENT LABELS IN GO TO REQUIRE INTEGER VARIABLES
STATEMENT NUMBER TOO LARGE
STATEMENT TOO LONG
SUBROUTINE NAME USED AS A VARIABLE
SUBSCRIPT ON A NON-DIMENSIONED VARIABLE
SUBSCRIPTED VARIABLE IN DATA STATEMENT NOT DIMENSIONED
SUBSCRIPTED SUBSCRIPTS NOT ALLOWED
SUBSCRIPTED SUBSCRIPTS NOT ALLOWED
SUCCESSIVE COMMAS OR IMPROPER CONSTANT IN PARAMETER LIST
SYNTAX ERROR IN COMPUTED GO-TO. COMMA OR PAREN LIKELY
SYNTAX OF IMPLICIT STATEMENT IS NOT CORRECT

TAPE PARAMETER MUST BE SIMPLE INTEGER VARIABLE OR INTEGER
CONSTANT
THE ARGUMENT MUST BE AN INTEGER
THE IF STATEMENT UNRECOGNIZED
THE LEFT HAND SIDE OF A REPLACEMENT STATEMENT IS MISSING
THE NESTING CAPACITY OF THE COMPILER HAS BEEN EXCEEDED
THE OBJECT OF A COMPUTED GO TO MUST BE AN INTEGER CONSTANT
OR VARIABLE
THE OBJECT OF AN ASSIGN OR ASSIGNED GO MUST BE A SIMPLE
INTEGER VARIABLE
THE PARAMETER TO THIS STATEMENT MUST BE AN INTEGER CONSTANT
OR VARIABLE
THE PARAMETERS OF A DO LOOP MUST BE AN UNSIGNED INTEGER
CONSTANT OR SIMPLE INTEGER VARIABLE
THE RUNNING SUBSCRIPT IN A DO LOOP MUST BE A SIMPLE INTEGER
VARIABLE
THE TERMINAL LABEL OF A DO MUST BE AN INTEGER CONSTANT
THIS FORMAT HAS NO STATEMENT NUMBER
THIS STATEMENT DOES NOT FOLLOW A DO WHICH IT TERMINATES
TOO MANY ARGUMENTS IN PREAMBLE LIST
TOO MANY CHARACTERS IN AN IDENTIFIER
TOO MANY RELATIONAL OPERATORS
TOO MANY SUBSCRIPT INDICES
TYPE COMPLEX IN IMPLICIT IS MULTIPLY DEFINED
TYPE DECLARATION NOT RECOGNIZED IN IMPLICIT STATEMENT
TYPE DOUBLE IN IMPLICIT IS MULTIPLY DEFINED
TYPE LOGICAL IN IMPLICIT IS MULTIPLY DEFINED
TYPE REAL OR INTEGER IN IMPLICIT IS MULTIPLY DEFINED

UNDEFINED SEPARATOR IN COMMON STATEMENT
UNIT MUST BE DEFINED FOR BUFFER STATEMENT
UNIT NUMBER INCORRECT
UNIT NUMBER MUST BE FOLLOWED BY A )
UNIT VALUE NOT INTEGER VARIABLE OR INTEGER CONSTANT
UNRECOGNIZABLE I/O STATEMENT
UNRECOGNIZED DELIMITER IN PARAMETER = EXPECTING A COMMA OR EQUALS SIGN
UNRECOGNIZED KEYWORD, SCAN TERMINATED
UNRECOGNIZED STATEMENT

VARIABLE DIMENSION ERROR
VARIABLE DIMENSIONED IDENTIFIER NOT IN FORMAL PARAMETER LIST
VARIABLE EQUIVALENCED TO ITSELF + N
VARIABLE IDENTIFIER IN EXTERNAL STATEMENT

WRONG FORMAT OF I/O STATEMENT, DATA LIST WAS NOT YET PROCESSED

ZERO IS NOT A VALID STATEMENT NUMBER

1ST (through 31ST) EXPRESSION NOT RECOGNIZABLE
1ST (through 31ST) POSITION KEYWORD UNRECOGNIZABLE
NONFATAL ERRORS

A FORMAT SPECIFICATION HAS NO FIELD WIDTH
ARGUMENT NOT DEFINED IN I/O LIST

E OR F OR D DECIMAL FIELD UNDEFINED
ENCODING/DECODING APPEARS WITHOUT A LIST
END MUST REFER TO STATEMENT LABEL
ERR MUST REFER TO STATEMENT LABEL
ERR STATEMENT LABEL OUT OF RANGE

FORMAT MUST BE INTEGER CONSTANT
FORMAT NOT TERMINATED BY RIGHT PARENTHESES

IN EW,D OR DW.D FORMAT SPECIFICATION, W SHOULD BE .GE. D+7
INQUIRE STATEMENT - ERROR IN (ACCESS). VALUE MUST BE DIRECT,
SEQUENTIAL, OR MIXED
INQUIRE STATEMENT - ERROR IN (EXPT)
INQUIRE STATEMENT - ERROR IN (EXISTS). MUST BE EITHER .TRUE.
OR .FALSE.
INQUIRE STATEMENT - ERROR IN (FILESEQ)
INQUIRE STATEMENT - ERROR IN (GEN)
INQUIRE STATEMENT - ERROR IN (MODE)
INQUIRE STATEMENT - ERROR IN (NAMED). IF FILE HAS A NAME,
(NAMED) MUST BE .TRUE.
INQUIRE STATEMENT - ERROR IN (NEXTREC)
INQUIRE STATEMENT - ERROR IN (NSTATE)
INQUIRE STATEMENT - ERROR IN (NUMBER)
INQUIRE STATEMENT - ERROR IN (OPENED). MUST BE EITHER .TRUE.
OR .FALSE.
INQUIRE STATEMENT - ERROR IN (RECL). MUST BE AN INTEGER
VARIABLE
INQUIRE STATEMENT - ERROR IN (SEQFIL)
INVALID CHARACTER IN A FORMAT SPECIFICATION

OPEN STATEMENT - ERROR IN (ERR). MUST BE STATEMENT LABEL
SETPASS - CPASS KEYWORD IN ERROR
SETPASS - PASS KEYWORD IN ERROR
SETPASS - RPASS KEYWORD IN ERROR
SETPASS - WPASS KEYWORD IN ERROR

THE FIRST ELEMENT IN A FORMAT STATEMENT MUST BE A ( 
THERE IS NO PATH TO THIS STATEMENT
THERE MUST BE A SEPARATOR BETWEEN FIELD SPECIFICATIONS
THIS CONSTANT STATEMENT PRECEDES DO

UNIT NOT DEFINED

2 BRANCH IF(UNIT) TEST WILL CAUSE UNPREDICTABLE RESULTS IN
CASE OF PARITY ERROR, ADVISE USING 4 BRANCH TEST
Several of the compiler diagnostics above indicate that a compiler table limit has been exceeded; e.g., COMPILER-TOO MANY FUNCTIONS. The following table gives the table sizes for both the FORTRAN and FORTRAN1 compilers. This table shows that certain of these errors can be eliminated by using the FORTRAN1 compiler. This is done by replacing *FORTRAN control cards with *FORTRAN1.

Table 11-1. Compiler Table Size Limits

<table>
<thead>
<tr>
<th>DESCRIPTION</th>
<th>LIMIT FORTRAN</th>
<th>FORTRAN1</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMON block names</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>Constants/routine</td>
<td>592</td>
<td>2048</td>
</tr>
<tr>
<td>Dimensioned variables/routine</td>
<td>284</td>
<td>512</td>
</tr>
<tr>
<td>DO-loop nests</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>Equivalenced variables/routine</td>
<td>192</td>
<td>192</td>
</tr>
<tr>
<td>Format statements/routine</td>
<td>64</td>
<td>256</td>
</tr>
<tr>
<td>Functions/routine</td>
<td>256</td>
<td>512</td>
</tr>
<tr>
<td>Index functions/routine</td>
<td>224</td>
<td>224</td>
</tr>
<tr>
<td>Index variables/routine</td>
<td>192</td>
<td>192</td>
</tr>
<tr>
<td>Nonstandard indices/routine</td>
<td>32</td>
<td>128</td>
</tr>
<tr>
<td>Parameters in parameter list</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>Typed variables/routine</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>Variables in COMMON blocks</td>
<td>284</td>
<td>512</td>
</tr>
</tbody>
</table>

† These table size limits are current as of the publication of this manual. They are subject to change.
Appendix A: 7600 Series Character Set ........................................ 12.2
Appendix B: Table of Powers of Two ........................................ 12.4
Appendix C: Octal-Decimal Integer Conversion Table ....................... 12.5
Appendix D: Internal Floating-Point Decimal Table .......................... 12.9
Appendix E: Decimal/Binary Position Table ................................... 12.10
Appendix F: Computer Word Structure of Constants--7600 ................... 12.11
Appendix G: Computer Word Structure of Constants--CRAY-1 ................ 12.12
Appendix H: FORTRAN Optimization Modifiers ............................... 12.13
### APPENDIX A: CONTROL DATA 7600 SERIES CHARACTER SET

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>:</td>
<td>00</td>
<td>00</td>
<td>8-2</td>
</tr>
<tr>
<td>A</td>
<td>01</td>
<td>61</td>
<td>12-1</td>
</tr>
<tr>
<td>B</td>
<td>02</td>
<td>62</td>
<td>12-2</td>
</tr>
<tr>
<td>C</td>
<td>03</td>
<td>63</td>
<td>12-3</td>
</tr>
<tr>
<td>D</td>
<td>04</td>
<td>64</td>
<td>12-4</td>
</tr>
<tr>
<td>E</td>
<td>05</td>
<td>65</td>
<td>12-5</td>
</tr>
<tr>
<td>F</td>
<td>06</td>
<td>66</td>
<td>12-6</td>
</tr>
<tr>
<td>G</td>
<td>07</td>
<td>67</td>
<td>12-7</td>
</tr>
<tr>
<td>H</td>
<td>10</td>
<td>70</td>
<td>12-8</td>
</tr>
<tr>
<td>I</td>
<td>11</td>
<td>71</td>
<td>12-9</td>
</tr>
<tr>
<td>J</td>
<td>12</td>
<td>41</td>
<td>11-1</td>
</tr>
<tr>
<td>K</td>
<td>13</td>
<td>42</td>
<td>11-2</td>
</tr>
<tr>
<td>L</td>
<td>14</td>
<td>43</td>
<td>11-3</td>
</tr>
<tr>
<td>M</td>
<td>15</td>
<td>44</td>
<td>11-4</td>
</tr>
<tr>
<td>N</td>
<td>16</td>
<td>45</td>
<td>11-5</td>
</tr>
<tr>
<td>O</td>
<td>17</td>
<td>46</td>
<td>11-6</td>
</tr>
<tr>
<td>P</td>
<td>20</td>
<td>47</td>
<td>11-7</td>
</tr>
<tr>
<td>Q</td>
<td>21</td>
<td>50</td>
<td>11-8</td>
</tr>
<tr>
<td>R</td>
<td>22</td>
<td>51</td>
<td>11-9</td>
</tr>
<tr>
<td>S</td>
<td>23</td>
<td>22</td>
<td>0-2</td>
</tr>
<tr>
<td>T</td>
<td>24</td>
<td>23</td>
<td>0-3</td>
</tr>
<tr>
<td>U</td>
<td>25</td>
<td>24</td>
<td>0-4</td>
</tr>
<tr>
<td>V</td>
<td>26</td>
<td>25</td>
<td>0-5</td>
</tr>
<tr>
<td>W</td>
<td>27</td>
<td>26</td>
<td>0-6</td>
</tr>
<tr>
<td>X</td>
<td>30</td>
<td>27</td>
<td>0-7</td>
</tr>
<tr>
<td>Y</td>
<td>31</td>
<td>30</td>
<td>0-8</td>
</tr>
<tr>
<td>Z</td>
<td>32</td>
<td>31</td>
<td>0-9</td>
</tr>
<tr>
<td>0</td>
<td>33</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>34</td>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>35</td>
<td>02</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>36</td>
<td>03</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>37</td>
<td>04</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>40</td>
<td>05</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>41</td>
<td>06</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>42</td>
<td>07</td>
<td>7</td>
</tr>
</tbody>
</table>
### Appendix

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>43</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>44</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td>+</td>
<td>45</td>
<td>60</td>
<td>12</td>
</tr>
<tr>
<td>-</td>
<td>46</td>
<td>40</td>
<td>11</td>
</tr>
<tr>
<td>#</td>
<td>47</td>
<td>54</td>
<td>11-8-4</td>
</tr>
<tr>
<td>/</td>
<td>50</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>(</td>
<td>51</td>
<td>24</td>
<td>0-8-4</td>
</tr>
<tr>
<td>)</td>
<td>52</td>
<td>74</td>
<td>12-8-4</td>
</tr>
<tr>
<td>$</td>
<td>53</td>
<td>53</td>
<td>11-8-3</td>
</tr>
<tr>
<td>=</td>
<td>54</td>
<td>13</td>
<td>8-3</td>
</tr>
<tr>
<td>space</td>
<td>55</td>
<td>20</td>
<td>space</td>
</tr>
<tr>
<td>,</td>
<td>56</td>
<td>33</td>
<td>0-8-3</td>
</tr>
<tr>
<td>;</td>
<td>57</td>
<td>73</td>
<td>12-8-3</td>
</tr>
<tr>
<td>:</td>
<td>60</td>
<td>36</td>
<td>0-8-6</td>
</tr>
<tr>
<td>[</td>
<td>61</td>
<td>17</td>
<td>8-7</td>
</tr>
<tr>
<td>]</td>
<td>62</td>
<td>32</td>
<td>0-8-2</td>
</tr>
<tr>
<td>%</td>
<td>63</td>
<td>16</td>
<td>8-6</td>
</tr>
<tr>
<td>#</td>
<td>64</td>
<td>14</td>
<td>8-4</td>
</tr>
<tr>
<td>\</td>
<td>65</td>
<td>35</td>
<td>0-8-5</td>
</tr>
<tr>
<td>\</td>
<td>66</td>
<td>52</td>
<td>11-0</td>
</tr>
<tr>
<td>^</td>
<td>67</td>
<td>37</td>
<td>0-8-7</td>
</tr>
<tr>
<td>+</td>
<td>70</td>
<td>55</td>
<td>11-8-5</td>
</tr>
<tr>
<td>-</td>
<td>71</td>
<td>56</td>
<td>11-8-6</td>
</tr>
<tr>
<td>&lt;</td>
<td>72</td>
<td>72</td>
<td>12-0</td>
</tr>
<tr>
<td>&gt;</td>
<td>73</td>
<td>57</td>
<td>11-8-7</td>
</tr>
<tr>
<td>=</td>
<td>74</td>
<td>15</td>
<td>8-5</td>
</tr>
<tr>
<td>7</td>
<td>75</td>
<td>75</td>
<td>12-8-5</td>
</tr>
<tr>
<td>?</td>
<td>76</td>
<td>76</td>
<td>12-8-6</td>
</tr>
<tr>
<td>:</td>
<td>77</td>
<td>77</td>
<td>12-8-7</td>
</tr>
</tbody>
</table>

**Notes:**
- + 11-0 and 11-8-2 are equivalent
- ++ 12-0 and 12-8-2 are equivalent
## APPENDIX B: TABLE OF POWERS OF TWO

<table>
<thead>
<tr>
<th>(2^n)</th>
<th>(2^{-n})</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>2</td>
<td>0.5</td>
</tr>
<tr>
<td>4</td>
<td>0.25</td>
</tr>
<tr>
<td>8</td>
<td>0.125</td>
</tr>
<tr>
<td>16</td>
<td>0.0625</td>
</tr>
<tr>
<td>32</td>
<td>0.03125</td>
</tr>
<tr>
<td>64</td>
<td>0.015625</td>
</tr>
<tr>
<td>128</td>
<td>0.0078125</td>
</tr>
<tr>
<td>256</td>
<td>0.00390625</td>
</tr>
<tr>
<td>512</td>
<td>0.001953125</td>
</tr>
<tr>
<td>1024</td>
<td>0.0009765625</td>
</tr>
<tr>
<td>2048</td>
<td>0.00048828125</td>
</tr>
<tr>
<td>4096</td>
<td>0.000244140625</td>
</tr>
<tr>
<td>8192</td>
<td>0.0001220703125</td>
</tr>
<tr>
<td>16384</td>
<td>0.00006103515625</td>
</tr>
<tr>
<td>32768</td>
<td>0.000030517578125</td>
</tr>
<tr>
<td>65536</td>
<td>0.0000152687896625</td>
</tr>
<tr>
<td>131072</td>
<td>0.0000076289394375</td>
</tr>
<tr>
<td>262144</td>
<td>0.000003814697265625</td>
</tr>
<tr>
<td>524288</td>
<td>0.00000190734863215625</td>
</tr>
<tr>
<td>1048576</td>
<td>0.00000095367431640625</td>
</tr>
<tr>
<td>2097152</td>
<td>0.0000004768371583023125</td>
</tr>
<tr>
<td>4194304</td>
<td>0.0000002384185791015625</td>
</tr>
<tr>
<td>8388608</td>
<td>0.00000011920928945578125</td>
</tr>
<tr>
<td>16777216</td>
<td>0.000000059004464775390625</td>
</tr>
<tr>
<td>33554322</td>
<td>0.0000000290082023278693125</td>
</tr>
<tr>
<td>67108644</td>
<td>0.000000014901090161938476625</td>
</tr>
<tr>
<td>134217288</td>
<td>0.000000007450580596923828125</td>
</tr>
<tr>
<td>268434576</td>
<td>0.0000000037262989264619140625</td>
</tr>
<tr>
<td>536870912</td>
<td>0.00000000186264514932309503125</td>
</tr>
<tr>
<td>1073741824</td>
<td>0.0000000009313227544184755185625</td>
</tr>
<tr>
<td>2147483648</td>
<td>0.00000000046566128730739258125</td>
</tr>
<tr>
<td>4294967296</td>
<td>0.00000000023283064365368968980825</td>
</tr>
<tr>
<td>8589934592</td>
<td>0.0000000001164153218269341465125</td>
</tr>
<tr>
<td>17179869184</td>
<td>0.0000000000582076609134674072265625</td>
</tr>
<tr>
<td>34359738368</td>
<td>0.0000000000291038304567337036128125</td>
</tr>
<tr>
<td>68719476736</td>
<td>0.00000000001551915226366858066425</td>
</tr>
<tr>
<td>137438953472</td>
<td>0.0000000000072759576141834259033203125</td>
</tr>
<tr>
<td>274877906944</td>
<td>0.000000000003637978807909171395166015625</td>
</tr>
<tr>
<td>549755813888</td>
<td>0.00000000000181898940354585673583078125</td>
</tr>
<tr>
<td>1099511627776</td>
<td>0.0000000000009094947017729282379150396025</td>
</tr>
<tr>
<td>219902355552</td>
<td>0.00000000000045474735088646411898759193125</td>
</tr>
<tr>
<td>439804611104</td>
<td>0.000000000000227373765443223549775765625</td>
</tr>
<tr>
<td>879609310208</td>
<td>0.0000000000001136883877216027397888125</td>
</tr>
<tr>
<td>17592186044416</td>
<td>0.00000000000005684341886080801486868949140625</td>
</tr>
<tr>
<td>351843720863245</td>
<td>0.000000000000028421709430400007434844970703125</td>
</tr>
<tr>
<td>703687441776646</td>
<td>0.000000000000014210854715200037174224883515625</td>
</tr>
<tr>
<td>1407348835532847</td>
<td>0.000000000000007105427357601001858711247678125</td>
</tr>
<tr>
<td>2814797671065648</td>
<td>0.00000000000000355271367880000592355621337890625</td>
</tr>
<tr>
<td>562949534213129</td>
<td>0.0000000000000017763563894000250644678108689453125</td>
</tr>
<tr>
<td>1125890906842624</td>
<td>0.0000000000000007761375643843968257001252323389053344725665625</td>
</tr>
<tr>
<td>2251799813685248</td>
<td>0.000000000000000444089209850862161864525667236328125</td>
</tr>
<tr>
<td>4503599627370496</td>
<td>0.00000000000000034444604462950313080847263361816460625</td>
</tr>
<tr>
<td>900719925470992</td>
<td>0.00000000000000011102230246251564504346316800803125</td>
</tr>
<tr>
<td>1801439850481984</td>
<td>0.0000000000000000551151121235287282181583404541015625</td>
</tr>
<tr>
<td>36028797018963968</td>
<td>0.00000000000000002775557561662891351059079127027058125</td>
</tr>
<tr>
<td>7205759403927936</td>
<td>0.00000000000000001378778780815466757552953958511362590625</td>
</tr>
<tr>
<td>144115180758557287</td>
<td>0.00000000000000000693893909308827736497925567626553125</td>
</tr>
<tr>
<td>28823037647117448</td>
<td>0.000000000000000003469446951953614668824869678314765625</td>
</tr>
<tr>
<td>57646075230342248</td>
<td>0.000000000000000001734723475878709441192448139109073828125</td>
</tr>
</tbody>
</table>

1,525,921,504,606,846,976
12.5
Appendices

APPENDIX C:

OCTAL-DECINAL INTEGER CONVERSION TABLE

0

1

2

3

4

5

6

7

0

1

2

3

4

5

0256

0257

0258

0259

0260

0261

6

7

0262

0263

0000

0000

0000 0000 0001

to;
0777

to
0511

0010 0008 0009 0010 0011 0012 0013 0014 0015
0020 0016 0017 0018 0019 0020 0021 0022 0023
0030 0024 0025 0026 0027 0028 0029 0030 0031

0410 0264 0265 0266 0267 0268 0269 0270 0271
0420 0272 0273 0274 0275 0276 0277 0278 0279
0430 0280 0281 0282 0283 0284 0285 0286 0287

0040
0050
0060
0070

0032
0040
0048
0056

0033
0041
0049
0057

0034
0042
0050
0058

0035
0043
0051
0059

0036
0044
0052
0060

0037
0045
0053
0061

0038
0046
0054
0062

0039
0047
0055
0063

0440
0450
0460
0470

0288
0296
0304
0312

0289
0297
0305
0313

0290
0298
0306
0314

0291
0299
0307
0315

0292
0300
0308
0316

0293
0301
0309
0317

0294
0302
0310
0318

0295
0303
0311
0319

0100
0110
0120
0130
0140
0150
0160
0170

0064
0072
0080
0088
0096
0104
0112
0120

0065
0073
0081
0089
0097
0105
0113
0121

0066
0074
0082
0090
0098
0106
0114
0122

0067
0075
0083
0091
0099
0107
0115
0123

0068
0076
-0084
0092
0100
0108
0116
0124

0069
0077
0085
0093
0101
0109
0117
0125

0070
0078
0086
0094
0102
0110
0118
0126

0071
0079
'0087
0095
0103
0111
0119
0127

0500
0510
0520
0530
0540
0550
0560
0570

0320
0328
0336
0344
0352
0360
0368
0376

0321
0329
0337
0345
0353
0361
0369
0377

0322
0330
0338
0346
0354
0362
0370
0378

0323
0331
0339
0347
0355
0363
0371
0379

0324
0332
0340
0348
0356
0364
0372
0380

0325
0333
0341
0349
0357
0365
0373
0381

0326
0334
0342
0350
0358
0366
0374
0382

0327
0335
0343
0351
0359
0367
0375
0383

0200
0210
0220
0230
0240
0250
0260
0270

0128
0136
0144
0152
0160
0168
0176
0184

0129
0137
0145
0153
0161
0169
0177
0185

0130
0138
0146
0154
0162
0170
0178
0186

0131
0139
0147
0155
0163
0171
0179
0187

0132
0140
0148
0156
0164
0172
0180
0188

0133
0141
0149
0157
0165
0173
0181
0189

0134
0142
0150
0158
.0166
0174
0182
0190

0135
0143
0151
0159
0167
0175
0183
0191

0600
0610
0620
0630
0640
0650
0660
0670

0384
0392
0400
0408
0416
0424
0432
0440

0385
0393
0401
0409
0417
0425
0433
0441

0386 0387
0394 0395
0402 0403
0410 0411
0418 0419
0426 0421
0434 0435
0442 .0443

0388
0396
0404
0412
0420
0428
0436
0444

0389
0397
0405
0413
0421
0429
0437
0445

0390
0398
0406
0414
0422
0430
0438
0446

0391
0399
0407
0415
0423
0431
0439
0447

0300
0310
0320
0330
0340
0350
0360
0370

0192
0200
0208
0216
0224
0232
0240
0248

0193
0201
0209
0217
0225
0233
0241
0249

0194
0202
0210
0218
0226
0234
0242
0250

0195
0203
0211
0219
0227
0235
0243
0251

0196
0204
0212
0220
0228
0236
0244
0252

0197
0205
0213
0221
0229
0237
0245
0253

0198
0206
0214
0222
0230
0238
0246
0254

0199
0207
0215
0223
0231
0239
0247
0255

0700
0710
0720
0730
0740
0750
0760
0770

0448
0456
0464
0472
0480
0488
0496
0504

0449
0457
0465
0473
0481
0489
0497
0505

0450
0458
0466
0474
0482
0490
0498
0506

0452
0460
0468
0476
0484
0492
0500
0508

0453
0461
0469
0477
0485
0493
0501
0509

0454
0462
0470
0478
0486
0494
0502
0510

0455
0463
0471
0479
0487
0495
0503
0511

0

1

2

3

0

1

4

5

6

7

Octal Decimal
10000- 4096
20000- 8192
30000-12288
40000-163840
60000-24576
70000-28672

0002 0003

0004

4

0005

5

0006

6

0007

0400

7

2

0451
0459
0467
0475
0483
0491
0499
0507

3

1000
to

0512
to

1000 0512 0513 0514 0515 0516 0517 0518 0519
1010 0520 0521 0522 0523 0524 0525 0526 0527

1400 0768 0769 0770 0771 0772 0773 0774 0775
1410 0776 0777 0778 0779 0780 0781 0782 0783

1777

1023

1020 0528 0529 0530 0531 0532 0533 0534 0535

1420 0784 0785 0786 0787 0788 0789 0790 0791

1030
1040
1050
1060
1070

0536
0544
0552
0560
0568

0537
0545
0553
0561
0569

0538
0546
0554
0562
0570

0539
0547
0555
0563
0571

0540
0548
0556
0564
0572

0541
0549
0557
0565
0573

0542
0550
0558
0566
0574

0543
0551
0559
0567
0575

1430
1440
1450
1460
1470

0792
0800
0808
0816
0824

0793
0801
0809
0817
0825

0794
0802
0810
0818
0826

0795
0803
0811
0819
0827

0796
0804
0812
0820
0828

0797
0805
0813
0821
0829

0798
0806
0814
0822
0830

0799
0807
0815
0823
0831

1100
1110
1120
1130
1140
1150
1160
1170

0576
0584
0592
0600
0608
0616
0624
0632

0577 0578
0585 0586
0593 0594
0601 0602
0609 0610
0617 0618
0625- 0626
0633 0634

0579
0587
0595
0603
0611
0619
0627
0635

0580
0588
0596
0604
0612
0620
0628
0636

0581
0589
0597
0605
0613
0621
0629
0637

0582
0590
0598
0606
0614
0622
0630
0638

0583
0591
0599
0607
0615
0623
0631
0639

1500
1510
1520
1530
1540
1550
1560
1570

0832
0840
0848
0856
0864
0872
0880
0888

0833
0841
0849
0857
0865
0873
0881
0889

0834
0842
0850
0858
0866
0874
0882
0890

0835
0843
0851
0859
0867
0875
0883
0891

0836
0844
0852
0860
0868
0876
0884
0892

0837
0845
0853
0861
0869
0877
0885
0893

0838
0846
0854
0862
0870
0878
0886
0894

0839
0847
0855
0863
0871
0879
0887
0895

1200
1210
1220
1230
1240

0640
0648
0656
0664
0672

0641
0649
0657
0665
0673

0643
0651
0659
0667
0675

0644
0652
0660
0668
0676

0645
0653
0661
0669
0677

0646
0654
0662
0670
0678

0647
0655
0663
0671
0679

1600
1610
1620
1630
1640

0896
0904
0912
0920
0928

0897
0905
0913
0921
0929

0898
0906
0914
0922
0930

0899
0907
0915
0923
0931

0900
0908
0916
0924
0932

0901
0909
0917
0925
0933

0902
0910
0918
0926
0934

0903
0911
0919
0927
0935

0642
0650
0658
0666
0674

1250 0680 0681 0682 0683 0684 0685 0686 0687
1260 0688 0689 0690 0691 0692 0693 0694 0695

1650 0936 0937 0938 0939 0940 0941 0942 0943
1660 0944 0945 0946 0947 0948 0949 0950 0951

1270 0696 0697 0698 0699 0700 0701 0702 0703

1670 0952 0953 0954 0955 0956 0957 0958 0959

1300 0704 0705 0706 0707 0708 0709 0710 0711

1700 0960 0961 0962 0963 0964 0965 0966 0967

1310 0712 0713 0714 0715 0716 0717 0718 0719
1320 0720 0721 0722 0723 0724 0725 0726 0727
1330 0728 0729 0730 0731 0732 0733 0734 0735

1710 0968 0969 0970 0971 0972 0973 0974 0975
1720 0976 0977 0978 0979 0980 0981 0982 0983
1730 0984 0985 0986 0987 0988 0989 0990 0991

1340
1350
1360
1370

1740
1750
1760
1770

0736
0744
0752
0760

0737 0738 0739
0745 0746 0747
0753 0754 0755
0761 0762 0763

0740
0748
0756
0764

0741
0749
0757
0765

0742
0750
0758
0766

0743
0751
0759
0767

0992
1000
1008
1016

0993 0994 0995 0996 0997 0998 0999
1001 1002 1003 1004 1005 1006 1007
1009 1010 1011 1012 1013 1014 1015
1017 1018 1019 1020 1021 1022 1023


**APPENDIX C: OCTAL-DECIMAL INTEGER CONVERSION TABLE (continued)**

<table>
<thead>
<tr>
<th>Octal</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>1024</td>
</tr>
<tr>
<td>0001</td>
<td>1025</td>
</tr>
<tr>
<td>0002</td>
<td>1026</td>
</tr>
<tr>
<td>0003</td>
<td>1027</td>
</tr>
<tr>
<td>0004</td>
<td>1028</td>
</tr>
<tr>
<td>0005</td>
<td>1029</td>
</tr>
<tr>
<td>0006</td>
<td>1030</td>
</tr>
<tr>
<td>0007</td>
<td>1031</td>
</tr>
<tr>
<td>0010</td>
<td>1032</td>
</tr>
<tr>
<td>0011</td>
<td>1033</td>
</tr>
<tr>
<td>0012</td>
<td>1034</td>
</tr>
<tr>
<td>0013</td>
<td>1035</td>
</tr>
<tr>
<td>0014</td>
<td>1036</td>
</tr>
<tr>
<td>0015</td>
<td>1037</td>
</tr>
<tr>
<td>0016</td>
<td>1038</td>
</tr>
<tr>
<td>0017</td>
<td>1039</td>
</tr>
<tr>
<td>0020</td>
<td>1040</td>
</tr>
<tr>
<td>0021</td>
<td>1041</td>
</tr>
<tr>
<td>0022</td>
<td>1042</td>
</tr>
<tr>
<td>0023</td>
<td>1043</td>
</tr>
<tr>
<td>0024</td>
<td>1044</td>
</tr>
<tr>
<td>0025</td>
<td>1045</td>
</tr>
<tr>
<td>0026</td>
<td>1046</td>
</tr>
<tr>
<td>0027</td>
<td>1047</td>
</tr>
<tr>
<td>0030</td>
<td>1050</td>
</tr>
<tr>
<td>0031</td>
<td>1051</td>
</tr>
<tr>
<td>0032</td>
<td>1052</td>
</tr>
<tr>
<td>0033</td>
<td>1053</td>
</tr>
<tr>
<td>0034</td>
<td>1054</td>
</tr>
<tr>
<td>0035</td>
<td>1055</td>
</tr>
<tr>
<td>0036</td>
<td>1056</td>
</tr>
<tr>
<td>0037</td>
<td>1057</td>
</tr>
<tr>
<td>0040</td>
<td>1060</td>
</tr>
<tr>
<td>0041</td>
<td>1061</td>
</tr>
<tr>
<td>0042</td>
<td>1062</td>
</tr>
<tr>
<td>0043</td>
<td>1063</td>
</tr>
<tr>
<td>0044</td>
<td>1064</td>
</tr>
<tr>
<td>0045</td>
<td>1065</td>
</tr>
<tr>
<td>0046</td>
<td>1066</td>
</tr>
<tr>
<td>0047</td>
<td>1067</td>
</tr>
<tr>
<td>0050</td>
<td>1070</td>
</tr>
<tr>
<td>0051</td>
<td>1071</td>
</tr>
<tr>
<td>0052</td>
<td>1072</td>
</tr>
<tr>
<td>0053</td>
<td>1073</td>
</tr>
<tr>
<td>0054</td>
<td>1074</td>
</tr>
<tr>
<td>0055</td>
<td>1075</td>
</tr>
<tr>
<td>0056</td>
<td>1076</td>
</tr>
<tr>
<td>0057</td>
<td>1077</td>
</tr>
<tr>
<td>0060</td>
<td>1080</td>
</tr>
<tr>
<td>0061</td>
<td>1081</td>
</tr>
<tr>
<td>0062</td>
<td>1082</td>
</tr>
<tr>
<td>0063</td>
<td>1083</td>
</tr>
<tr>
<td>0064</td>
<td>1084</td>
</tr>
<tr>
<td>0065</td>
<td>1085</td>
</tr>
<tr>
<td>0066</td>
<td>1086</td>
</tr>
<tr>
<td>0067</td>
<td>1087</td>
</tr>
<tr>
<td>0070</td>
<td>1090</td>
</tr>
<tr>
<td>0071</td>
<td>1091</td>
</tr>
<tr>
<td>0072</td>
<td>1092</td>
</tr>
<tr>
<td>0073</td>
<td>1093</td>
</tr>
<tr>
<td>0074</td>
<td>1094</td>
</tr>
<tr>
<td>0075</td>
<td>1095</td>
</tr>
<tr>
<td>0076</td>
<td>1096</td>
</tr>
<tr>
<td>0077</td>
<td>1097</td>
</tr>
<tr>
<td>0080</td>
<td>1100</td>
</tr>
<tr>
<td>0081</td>
<td>1101</td>
</tr>
<tr>
<td>0082</td>
<td>1102</td>
</tr>
<tr>
<td>0083</td>
<td>1103</td>
</tr>
<tr>
<td>0084</td>
<td>1104</td>
</tr>
<tr>
<td>0085</td>
<td>1105</td>
</tr>
<tr>
<td>0086</td>
<td>1106</td>
</tr>
<tr>
<td>0087</td>
<td>1107</td>
</tr>
<tr>
<td>0090</td>
<td>1110</td>
</tr>
<tr>
<td>0091</td>
<td>1111</td>
</tr>
<tr>
<td>0092</td>
<td>1112</td>
</tr>
<tr>
<td>0093</td>
<td>1113</td>
</tr>
<tr>
<td>0094</td>
<td>1114</td>
</tr>
<tr>
<td>0095</td>
<td>1115</td>
</tr>
<tr>
<td>0096</td>
<td>1116</td>
</tr>
<tr>
<td>0097</td>
<td>1117</td>
</tr>
<tr>
<td>0100</td>
<td>1120</td>
</tr>
<tr>
<td>0101</td>
<td>1121</td>
</tr>
<tr>
<td>0102</td>
<td>1122</td>
</tr>
<tr>
<td>0103</td>
<td>1123</td>
</tr>
<tr>
<td>0104</td>
<td>1124</td>
</tr>
<tr>
<td>0105</td>
<td>1125</td>
</tr>
<tr>
<td>0106</td>
<td>1126</td>
</tr>
<tr>
<td>0107</td>
<td>1127</td>
</tr>
<tr>
<td>0110</td>
<td>1130</td>
</tr>
<tr>
<td>0111</td>
<td>1131</td>
</tr>
<tr>
<td>0112</td>
<td>1132</td>
</tr>
<tr>
<td>0113</td>
<td>1133</td>
</tr>
<tr>
<td>0114</td>
<td>1134</td>
</tr>
<tr>
<td>0115</td>
<td>1135</td>
</tr>
<tr>
<td>0116</td>
<td>1136</td>
</tr>
<tr>
<td>0117</td>
<td>1137</td>
</tr>
<tr>
<td>0120</td>
<td>1140</td>
</tr>
<tr>
<td>0121</td>
<td>1141</td>
</tr>
<tr>
<td>0122</td>
<td>1142</td>
</tr>
<tr>
<td>0123</td>
<td>1143</td>
</tr>
<tr>
<td>0124</td>
<td>1144</td>
</tr>
<tr>
<td>0125</td>
<td>1145</td>
</tr>
<tr>
<td>0126</td>
<td>1146</td>
</tr>
<tr>
<td>0127</td>
<td>1147</td>
</tr>
<tr>
<td>0130</td>
<td>1150</td>
</tr>
<tr>
<td>0131</td>
<td>1151</td>
</tr>
<tr>
<td>0132</td>
<td>1152</td>
</tr>
<tr>
<td>0133</td>
<td>1153</td>
</tr>
<tr>
<td>0134</td>
<td>1154</td>
</tr>
<tr>
<td>0135</td>
<td>1155</td>
</tr>
<tr>
<td>0136</td>
<td>1156</td>
</tr>
<tr>
<td>0137</td>
<td>1157</td>
</tr>
<tr>
<td>0140</td>
<td>1160</td>
</tr>
<tr>
<td>0141</td>
<td>1161</td>
</tr>
<tr>
<td>0142</td>
<td>1162</td>
</tr>
<tr>
<td>0143</td>
<td>1163</td>
</tr>
<tr>
<td>0144</td>
<td>1164</td>
</tr>
<tr>
<td>0145</td>
<td>1165</td>
</tr>
<tr>
<td>0146</td>
<td>1166</td>
</tr>
<tr>
<td>0147</td>
<td>1167</td>
</tr>
<tr>
<td>0150</td>
<td>1170</td>
</tr>
<tr>
<td>0151</td>
<td>1171</td>
</tr>
<tr>
<td>0152</td>
<td>1172</td>
</tr>
<tr>
<td>0153</td>
<td>1173</td>
</tr>
<tr>
<td>0154</td>
<td>1174</td>
</tr>
<tr>
<td>0155</td>
<td>1175</td>
</tr>
<tr>
<td>0156</td>
<td>1176</td>
</tr>
<tr>
<td>0157</td>
<td>1177</td>
</tr>
</tbody>
</table>
### APPENDIX C: OCTAL-DECIMAL INTEGER CONVERSION TABLE (continued)

#### Octal to Decimal

<table>
<thead>
<tr>
<th>Octal</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>0010</td>
<td>2</td>
</tr>
<tr>
<td>0011</td>
<td>3</td>
</tr>
<tr>
<td>0100</td>
<td>4</td>
</tr>
<tr>
<td>0101</td>
<td>5</td>
</tr>
<tr>
<td>0110</td>
<td>6</td>
</tr>
<tr>
<td>0111</td>
<td>7</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>1001</td>
<td>9</td>
</tr>
<tr>
<td>1010</td>
<td>10</td>
</tr>
<tr>
<td>1011</td>
<td>11</td>
</tr>
<tr>
<td>1100</td>
<td>12</td>
</tr>
<tr>
<td>1101</td>
<td>13</td>
</tr>
<tr>
<td>1110</td>
<td>14</td>
</tr>
<tr>
<td>1111</td>
<td>15</td>
</tr>
</tbody>
</table>

#### Decimal to Octal

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Octal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Octal</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0011</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0101</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0110</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0111</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1011</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1101</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1110</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1111</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

#### 5000-5999

<table>
<thead>
<tr>
<th>Octal</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0011</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0101</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0110</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0111</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1011</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1101</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1110</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1111</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

#### 6000-6999

<table>
<thead>
<tr>
<th>Octal</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0011</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0101</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0110</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0111</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1011</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1101</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1110</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1111</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

#### 7000-7999

<table>
<thead>
<tr>
<th>Octal</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0011</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0101</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0110</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0111</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1011</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1101</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1110</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1111</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
### APPENDIX C: OCTAL-DECIMAL INTEGER CONVERSION TABLE (continued)

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>8000</td>
<td>3072</td>
<td>3073</td>
<td>3074</td>
<td>3075</td>
<td>3076</td>
<td>3077</td>
<td>3078</td>
</tr>
<tr>
<td>8010</td>
<td>3080</td>
<td>3081</td>
<td>3082</td>
<td>3083</td>
<td>3084</td>
<td>3085</td>
<td>3086</td>
</tr>
<tr>
<td>8020</td>
<td>3090</td>
<td>3091</td>
<td>3092</td>
<td>3093</td>
<td>3094</td>
<td>3095</td>
<td>3096</td>
</tr>
<tr>
<td>8030</td>
<td>3100</td>
<td>3101</td>
<td>3102</td>
<td>3103</td>
<td>3104</td>
<td>3105</td>
<td>3106</td>
</tr>
<tr>
<td>8040</td>
<td>3110</td>
<td>3111</td>
<td>3112</td>
<td>3113</td>
<td>3114</td>
<td>3115</td>
<td>3116</td>
</tr>
<tr>
<td>8050</td>
<td>3120</td>
<td>3121</td>
<td>3122</td>
<td>3123</td>
<td>3124</td>
<td>3125</td>
<td>3126</td>
</tr>
<tr>
<td>8060</td>
<td>3130</td>
<td>3131</td>
<td>3132</td>
<td>3133</td>
<td>3134</td>
<td>3135</td>
<td>3136</td>
</tr>
<tr>
<td>8070</td>
<td>3140</td>
<td>3141</td>
<td>3142</td>
<td>3143</td>
<td>3144</td>
<td>3145</td>
<td>3146</td>
</tr>
<tr>
<td>8080</td>
<td>3150</td>
<td>3151</td>
<td>3152</td>
<td>3153</td>
<td>3154</td>
<td>3155</td>
<td>3156</td>
</tr>
<tr>
<td>8090</td>
<td>3160</td>
<td>3161</td>
<td>3162</td>
<td>3163</td>
<td>3164</td>
<td>3165</td>
<td>3166</td>
</tr>
<tr>
<td>8100</td>
<td>3170</td>
<td>3171</td>
<td>3172</td>
<td>3173</td>
<td>3174</td>
<td>3175</td>
<td>3176</td>
</tr>
<tr>
<td>8110</td>
<td>3180</td>
<td>3181</td>
<td>3182</td>
<td>3183</td>
<td>3184</td>
<td>3185</td>
<td>3186</td>
</tr>
<tr>
<td>8120</td>
<td>3190</td>
<td>3191</td>
<td>3192</td>
<td>3193</td>
<td>3194</td>
<td>3195</td>
<td>3196</td>
</tr>
<tr>
<td>8130</td>
<td>3200</td>
<td>3201</td>
<td>3202</td>
<td>3203</td>
<td>3204</td>
<td>3205</td>
<td>3206</td>
</tr>
<tr>
<td>8140</td>
<td>3210</td>
<td>3211</td>
<td>3212</td>
<td>3213</td>
<td>3214</td>
<td>3215</td>
<td>3216</td>
</tr>
<tr>
<td>8150</td>
<td>3220</td>
<td>3221</td>
<td>3222</td>
<td>3223</td>
<td>3224</td>
<td>3225</td>
<td>3226</td>
</tr>
<tr>
<td>8160</td>
<td>3230</td>
<td>3231</td>
<td>3232</td>
<td>3233</td>
<td>3234</td>
<td>3235</td>
<td>3236</td>
</tr>
<tr>
<td>8170</td>
<td>3240</td>
<td>3241</td>
<td>3242</td>
<td>3243</td>
<td>3244</td>
<td>3245</td>
<td>3246</td>
</tr>
<tr>
<td>8180</td>
<td>3250</td>
<td>3251</td>
<td>3252</td>
<td>3253</td>
<td>3254</td>
<td>3255</td>
<td>3256</td>
</tr>
<tr>
<td>8190</td>
<td>3260</td>
<td>3261</td>
<td>3262</td>
<td>3263</td>
<td>3264</td>
<td>3265</td>
<td>3266</td>
</tr>
<tr>
<td>8200</td>
<td>3270</td>
<td>3271</td>
<td>3272</td>
<td>3273</td>
<td>3274</td>
<td>3275</td>
<td>3276</td>
</tr>
<tr>
<td>8210</td>
<td>3280</td>
<td>3281</td>
<td>3282</td>
<td>3283</td>
<td>3284</td>
<td>3285</td>
<td>3286</td>
</tr>
<tr>
<td>8220</td>
<td>3290</td>
<td>3291</td>
<td>3292</td>
<td>3293</td>
<td>3294</td>
<td>3295</td>
<td>3296</td>
</tr>
<tr>
<td>8230</td>
<td>3300</td>
<td>3301</td>
<td>3302</td>
<td>3303</td>
<td>3304</td>
<td>3305</td>
<td>3306</td>
</tr>
<tr>
<td>8240</td>
<td>3310</td>
<td>3311</td>
<td>3312</td>
<td>3313</td>
<td>3314</td>
<td>3315</td>
<td>3316</td>
</tr>
<tr>
<td>8250</td>
<td>3320</td>
<td>3321</td>
<td>3322</td>
<td>3323</td>
<td>3324</td>
<td>3325</td>
<td>3326</td>
</tr>
<tr>
<td>8260</td>
<td>3330</td>
<td>3331</td>
<td>3332</td>
<td>3333</td>
<td>3334</td>
<td>3335</td>
<td>3336</td>
</tr>
<tr>
<td>8270</td>
<td>3340</td>
<td>3341</td>
<td>3342</td>
<td>3343</td>
<td>3344</td>
<td>3345</td>
<td>3346</td>
</tr>
<tr>
<td>8280</td>
<td>3350</td>
<td>3351</td>
<td>3352</td>
<td>3353</td>
<td>3354</td>
<td>3355</td>
<td>3356</td>
</tr>
<tr>
<td>8290</td>
<td>3360</td>
<td>3361</td>
<td>3362</td>
<td>3363</td>
<td>3364</td>
<td>3365</td>
<td>3366</td>
</tr>
<tr>
<td>8300</td>
<td>3370</td>
<td>3371</td>
<td>3372</td>
<td>3373</td>
<td>3374</td>
<td>3375</td>
<td>3376</td>
</tr>
<tr>
<td>8310</td>
<td>3380</td>
<td>3381</td>
<td>3382</td>
<td>3383</td>
<td>3384</td>
<td>3385</td>
<td>3386</td>
</tr>
<tr>
<td>8320</td>
<td>3390</td>
<td>3391</td>
<td>3392</td>
<td>3393</td>
<td>3394</td>
<td>3395</td>
<td>3396</td>
</tr>
<tr>
<td>8330</td>
<td>3400</td>
<td>3401</td>
<td>3402</td>
<td>3403</td>
<td>3404</td>
<td>3405</td>
<td>3406</td>
</tr>
<tr>
<td>8340</td>
<td>3410</td>
<td>3411</td>
<td>3412</td>
<td>3413</td>
<td>3414</td>
<td>3415</td>
<td>3416</td>
</tr>
<tr>
<td>8350</td>
<td>3420</td>
<td>3421</td>
<td>3422</td>
<td>3423</td>
<td>3424</td>
<td>3425</td>
<td>3426</td>
</tr>
<tr>
<td>8360</td>
<td>3430</td>
<td>3431</td>
<td>3432</td>
<td>3433</td>
<td>3434</td>
<td>3435</td>
<td>3436</td>
</tr>
<tr>
<td>8370</td>
<td>3440</td>
<td>3441</td>
<td>3442</td>
<td>3443</td>
<td>3444</td>
<td>3445</td>
<td>3446</td>
</tr>
</tbody>
</table>

For converting octal to decimal, you can use the following formula:

\[ \text{Decimal} = \sum_{i=0}^{n} a_i \times 8^i \]

where \( a_i \) is the digit at position \( i \) and \( n \) is the highest power of 8 that is less than or equal to the number of digits in the octal number. For example, to convert the octal number 123 to decimal:

\[ 1 \times 8^2 + 2 \times 8^1 + 3 \times 8^0 = 64 + 16 + 3 = 83 \]

The decimal equivalent of the octal number 123 is 83.
## APPENDIX D: INTERNAL FLOATING-POINT DECIMAL TABLE

<table>
<thead>
<tr>
<th>Floating Point</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1574</td>
<td>7571</td>
</tr>
<tr>
<td>1615</td>
<td>5716</td>
</tr>
<tr>
<td>1636</td>
<td>4401</td>
</tr>
<tr>
<td>1656</td>
<td>6676</td>
</tr>
<tr>
<td>1677</td>
<td>5174</td>
</tr>
<tr>
<td>1702</td>
<td>6433</td>
</tr>
<tr>
<td>1706</td>
<td>4061</td>
</tr>
<tr>
<td>1711</td>
<td>5075</td>
</tr>
<tr>
<td>1713</td>
<td>6314</td>
</tr>
<tr>
<td>1714</td>
<td>6314</td>
</tr>
<tr>
<td>1717</td>
<td>4000</td>
</tr>
<tr>
<td>1720</td>
<td>0000</td>
</tr>
<tr>
<td>1722</td>
<td>5000</td>
</tr>
<tr>
<td>1723</td>
<td>5000</td>
</tr>
<tr>
<td>1725</td>
<td>6200</td>
</tr>
<tr>
<td>1726</td>
<td>6200</td>
</tr>
<tr>
<td>1731</td>
<td>7640</td>
</tr>
<tr>
<td>1735</td>
<td>4704</td>
</tr>
<tr>
<td>1740</td>
<td>6065</td>
</tr>
<tr>
<td>1761</td>
<td>4520</td>
</tr>
<tr>
<td>2002</td>
<td>7065</td>
</tr>
<tr>
<td>2023</td>
<td>5327</td>
</tr>
<tr>
<td>2044</td>
<td>4105</td>
</tr>
</tbody>
</table>

**Note:** A negative number is represented as the 7's complement of a positive number.

**Example**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7777</td>
<td>7777</td>
<td>7777</td>
<td>7777</td>
<td>7777</td>
<td></td>
</tr>
<tr>
<td>+.01 is</td>
<td>1711</td>
<td>5075</td>
<td>3412</td>
<td>1727</td>
<td>0243</td>
</tr>
<tr>
<td>-.01 is</td>
<td>6066</td>
<td>2702</td>
<td>4365</td>
<td>6050</td>
<td>7534</td>
</tr>
</tbody>
</table>
## APPENDIX E: DECIMAL/BINARY POSITION TABLE

<table>
<thead>
<tr>
<th>Largest Decimal Integer</th>
<th>Decimal Digits Req'd*</th>
<th>Number of Binary Digits</th>
<th>Largest Decimal Fraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>.5</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>.75</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>.875</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>.937 5</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>5</td>
<td>.968 75</td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>6</td>
<td>.984 375</td>
<td></td>
</tr>
<tr>
<td>127</td>
<td>7</td>
<td>.992 187 5</td>
<td></td>
</tr>
<tr>
<td>255</td>
<td>8</td>
<td>.996 093 75</td>
<td></td>
</tr>
<tr>
<td>511</td>
<td>9</td>
<td>.998 046 875</td>
<td></td>
</tr>
<tr>
<td>1 023</td>
<td>10</td>
<td>.999 023 437 5</td>
<td></td>
</tr>
<tr>
<td>2 047</td>
<td>11</td>
<td>.999 511 718 75</td>
<td></td>
</tr>
<tr>
<td>4 095</td>
<td>12</td>
<td>.999 755 859 375</td>
<td></td>
</tr>
<tr>
<td>8 191</td>
<td>13</td>
<td>.999 877 929 687 5</td>
<td></td>
</tr>
<tr>
<td>16 383</td>
<td>14</td>
<td>.999 938 964 843 75</td>
<td></td>
</tr>
<tr>
<td>32 767</td>
<td>15</td>
<td>.999 969 482 421 875</td>
<td></td>
</tr>
<tr>
<td>65 535</td>
<td>16</td>
<td>.999 984 741 210 937 5</td>
<td></td>
</tr>
<tr>
<td>131 071</td>
<td>17</td>
<td>.999 992 370 605 468 75</td>
<td></td>
</tr>
<tr>
<td>262 143</td>
<td>18</td>
<td>.999 996 185 302 734 375</td>
<td></td>
</tr>
<tr>
<td>524 287</td>
<td>19</td>
<td>.999 998 092 651 367 187 5</td>
<td></td>
</tr>
<tr>
<td>1 048 575</td>
<td>20</td>
<td>.999 999 046 325 683 593 75</td>
<td></td>
</tr>
<tr>
<td>2 097 151</td>
<td>21</td>
<td>.999 999 523 162 841 796 875</td>
<td></td>
</tr>
<tr>
<td>4 194 303</td>
<td>22</td>
<td>.999 999 811 581 420 896 437 5</td>
<td></td>
</tr>
<tr>
<td>8 388 607</td>
<td>23</td>
<td>.999 999 989 700 710 449 218 75</td>
<td></td>
</tr>
<tr>
<td>16 777 215</td>
<td>24</td>
<td>.999 999 940 395 355 244 609 375</td>
<td></td>
</tr>
<tr>
<td>33 554 431</td>
<td>25</td>
<td>.999 999 970 197 677 612 304 687 5</td>
<td></td>
</tr>
<tr>
<td>67 108 863</td>
<td>26</td>
<td>.999 999 985 098 836 606 152 343 75</td>
<td></td>
</tr>
<tr>
<td>134 217 727</td>
<td>27</td>
<td>.999 999 989 824 410 076 171 875</td>
<td></td>
</tr>
<tr>
<td>268 435 456</td>
<td>28</td>
<td>.999 999 995 611 600 291 777 375</td>
<td></td>
</tr>
<tr>
<td>536 870 911</td>
<td>29</td>
<td>.999 999 998 137 354 850 769 042 968 75</td>
<td></td>
</tr>
<tr>
<td>1 073 741 823</td>
<td>30</td>
<td>.999 999 999 068 677 425 384 521 484 375</td>
<td></td>
</tr>
<tr>
<td>2 147 483 647</td>
<td>31</td>
<td>.999 999 999 534 338 712 692 260 742 187 5</td>
<td></td>
</tr>
<tr>
<td>4 294 967 295</td>
<td>32</td>
<td>.999 999 999 767 163 356 346 130 371 093 75</td>
<td></td>
</tr>
<tr>
<td>8 589 934 591</td>
<td>33</td>
<td>.999 999 999 883 584 678 173 065 185 546 875</td>
<td></td>
</tr>
<tr>
<td>17 179 669 183</td>
<td>34</td>
<td>.999 999 999 941 792 339 086 532 592 772 437 5</td>
<td></td>
</tr>
<tr>
<td>34 355 178 387</td>
<td>35</td>
<td>.999 999 999 978 896 189 543 266 298 366 718 75</td>
<td></td>
</tr>
<tr>
<td>69 719 478 735</td>
<td>36</td>
<td>.999 999 999 985 448 034 771 633 148 193 359 375</td>
<td></td>
</tr>
<tr>
<td>137 438 953 471</td>
<td>37</td>
<td>.999 999 999 992 724 042 385 816 574 096 679 687 5</td>
<td></td>
</tr>
<tr>
<td>274 877 906 943</td>
<td>38</td>
<td>.999 999 999 996 362 021 192 988 287 048 339 843 75</td>
<td></td>
</tr>
<tr>
<td>549 755 813 887</td>
<td>39</td>
<td>.999 999 999 998 181 010 596 454 143 524 169 821 75</td>
<td></td>
</tr>
<tr>
<td>1 099 511 627 775</td>
<td>40</td>
<td>.999 999 999 999 090 505 298 277 071 762 004 980 937 5</td>
<td></td>
</tr>
<tr>
<td>2 199 023 255 561</td>
<td>41</td>
<td>.999 999 999 999 545 252 649 113 635 881 042 480 468 75</td>
<td></td>
</tr>
<tr>
<td>4 398 046 111 103</td>
<td>42</td>
<td>.999 999 999 999 772 626 324 556 767 940 521 240 234 375</td>
<td></td>
</tr>
<tr>
<td>8 796 093 222 07</td>
<td>43</td>
<td>.999 999 999 999 886 313 162 278 383 970 260 620 117 187 5</td>
<td></td>
</tr>
<tr>
<td>17 592 186 044 415</td>
<td>44</td>
<td>.999 999 999 999 943 156 581 139 191 985 130 310 058 593 75</td>
<td></td>
</tr>
<tr>
<td>35 184 372 088 831</td>
<td>45</td>
<td>.999 999 999 999 971 578 290 589 585 992 565 155 029 296 875</td>
<td></td>
</tr>
<tr>
<td>70 388 744 177 663</td>
<td>46</td>
<td>.999 999 999 999 985 769 145 294 751 998 292 577 514 948 437 5</td>
<td></td>
</tr>
<tr>
<td>140 737 488 355 327</td>
<td>47</td>
<td>.999 999 999 999 992 894 572 942 398 998 141 268 757 324 218 75</td>
<td></td>
</tr>
</tbody>
</table>

* Larger numbers within a digit group should be checked for exact number of decimal digits required.

### Examples of Use

**Q.** What is the largest decimal value which can be expressed by 36 binary digits?

A. 68,719,476,735.

**Q.** How many decimal digits will be required to express a 22-bit number?

A. 7 decimal digits.
APPENDIX F: COMPUTER WORD STRUCTURE OF CONSTANTS--7600

INTEGER

REAL

HOLLERITH BCD AND DISPLAY CODE

DOUBLE-PRECISION

COMPLEX

REAL

IMAGINARY

OCTAL
APPENDIX 6: COMPUTER WORD STRUCTURE OF CONSTANTS--CRAY-1

**INTEGER**

```
SIGN

0 23
```

**REAL**

```
SIGN

0 63
```

**CHARACTER CODE--ASCII**

```
0 7 8 15 16 23 24 31 32 39 40 47 48 55 56 63
```

**DOUBLE-PRECISION**

```
SIGN

0 15 16 23 63
```

**COMPLEX**

```
SIGN

0 15 16 63
```

**OCTAL**

```
0 3 4 6 7 9
```

```
5756606I 63
```
**APPENDIX H: FORTRAN COMPILATION MODIFIERS**

*FORTRAN,0 Apply 14, 15, 16 to prevent reordering of arithmetic sequence.

*FORTRAN,1 Do not remove constant statements from DO loops.

*FORTRAN,2 Do not remove constant subexpressions from DO loops.

*FORTRAN,3 Do not precalculate addresses in B-registers.

*FORTRAN,4 Do not optimize variant global functions.

*FORTRAN,13 Do not set B-boxes to constants.

*FORTRAN,14 Do not optimize divide operators. (This option turns off 15 as well.)

*FORTRAN,15 Do not optimize multiply operators.

*FORTRAN,16 Do not optimize add and subtract operators.

*FORTRAN,18 DO loop optimization is removed for modifiers 1,2,3,4,13.

*FORTRAN,60 Do not optimize any of the program whatsoever.

*FORTRAN,25 Compile DO-loops according to FORTRAN 77 standards.

*FORTRAN,26 Compile DATA statements according to FORTRAN 77 standards.
INDEX

A

Absolute value, 7.19
Access, 9.5, 9.66-67
direct, example of, 9.69
direct and sequential,
example of, 9.68
sequential, example of, 9.68
ACCESS=acc, 9.41, 9.49
Actual parameters, 7.4
Address, relative, 5.16
Algorithms, in-line, 7.18
Allocation
array, 2.20
storage, 5.1, 5.28, 10.4-5
Alphanumeric
constant, 2.11
conversion, 8.23
identifier, 2.2
labels, 2.16
names, 2.16
Alternate entry points, 7.28-29
Apostrophe delimiters, 8.8
Appendices, 12.1
Argument
dummy, 5.39
list, 7.4
Arithmetic evaluation, 3.3
Arithmetic IF statement, 6.5
Array, 2.17, 2.19
allocation, 2.20
double precision, 2.21
elements, 2.19-20
integer, 2.21
mapping, 5.12
name, 5.11
structure, 2.20
transmission, 8.3-4
ASSIGN statement, 6.3
Assigned GO TO statement, 6.2

Asynchronous
statement, alternate form for, 9.23
data transfers, 9.4
READ/WRITE, 9.19
Argument list, 7.3
Aw on input and output, 8.23

B

BACKFILE statement, 9.62
BACKSPACE statement, 9.61
Base 8 system, 2.6
Binary, 9.18, 9.25
BLANK COMMON, 5.16, 5.21, 5.23
Blank specification, 8.29
BLANK=blnk, 9.42, 9.53
BLK parameter in COMMON statement, 5.20
BLOCK DATA statement, 7.26
Block, 6.30-32
common; see Common block
ELSE, 6.32
IF, 6.28-29
IF/THEN/ELSE, 6.28
Branching and IF logic, 10.4-5
BUFFER IN statement, 9.24
BUFFER OUT statement, 9.24-25

C

CALL EXIT statement, 6.27
CALL statement, 7.8
Calling program, argument list for, 7.4
Card
blank, 1.4
program, 7.2
punched, 1.4
Character mode, in BUFFER OUT, 9.25
C (continued)

Character set
FORTRAN, 2.1
7600 series, 12.2
Characters
alphabetic, 2.1
Hollerith, 2.11
maximum
  in ENCODE, 9.29
  in output record, 8.31
number of, in a variable, 2.13
numeric, 2.1
special, 2.1
string of, in PAUSE statement, 6.26
Classes of variables, 2.16
clist list in DATA statement, 5.35
  correspondence between nlist and, 5.36
CLOSE statement, 9.44
  format, 9.44
  error diagnostics for, 9.45
Code, display, 2.11
Coding
  forms, FORTRAN, 1.1
  line, 1.2
Combined expressions, 3.17, 3.19
Comment
  information, 1.3
  option, special, 1.4
Common
  blank, 5.16, 5.21, 5.23
  block, 5.16
    changing length of, 5.22
    length of, 5.22-23
  declaration, 5.14
  blank, 5.21
    unused variables in, 5.22
  equivalence, 5.28-29
  variables, 5.20
COMMON statement, 5.20
Compilation
  diagnostics, 11.1
  errors fatal to, 11.5
  errors in FORTRAN program, 11.1
Compiler, 1.1
  errors, 11.3
  table size limits, 11.18

Complementing in masking expressions, 3.15
Complex
  constants, 2.10
  number, imaginary part of, 2.10
  number, real part of, 2.10
  variable, 2.15
    type of, 2.15
    word length of, 2.15
COMPLEX nlist, 5.2
Computed GO TO statement, 6.3
Connectives, logical, 3.13
Constant
  alphanumeric, 2.11
  changing the value of, 5.7
  double-precision, 2.9
  expression, 5.8
  Hollerith, 2.11
  integer, 2.4
Continuation, 1.2-3
CONTINUE statement, 6.26
Conversion, 7.19
  alphanumeric, 8.23
  and editing information, 8.7-8
  general, 8.16
  logical, 8.25
  octal, 8.21
  specifications, 8.9
  table, octal-decimal, 12.5
Core
  dimensioned array too large for, 5.10
  size, programs approaching maximum, 5.18
  to core transfer, 9.4, 10.8-9
Count, iteration, in DO statement, 6.40
CPASS=cpass, 9.56
CRAY-1 FORTRAN, 10.1
  Manual, 10.1
CRDT=crdt, 9.53
CRECL=cl, 9.53
D

Data
  access to, 9.5
  block, 7.26
  declaration, 5.35-36
  input, 10.6-7
  mode of, definition, 9.10
  output, 10.6-7
  source, 9.2
  transfer, 9.1
    formatted and unformatted, 9.6
    synchronous and asynchronous, 9.4
    transmission of, 8.1
DATA statement, 5.35
  assigning initial values in, 5.36
  blank common and, 5.21
  block, 7.26
  implied DO loop in, 5.37
  variable format in, 8.37
Dataset support, 10.8-9
  statements, 9.38
Decimal
  /binary position table, 12.10
  internal floating-point table, 12.9
Deck, arrangement of FORTRAN, 7.35
Declaration
  blank common, 5.21
  common, 5.14
    unused variables in, 5.22
  data, 5.35
    rules for, 5.36
  dimension, 5.10
  equivalence, 5.24
    rules for, 5.27
  format, 8.7
  type, 5.1, 10.2-3
    and storage allocation, 5.1
  nlist parameter in, 5.2
    rules for, 5.2
  DECODE statement, 9.27, 9.33
    examples, 9.36
    parameters, 9.27
  Default, modifying, 5.4
  Delimiters, star and apostrophe, 8.8

Diagnostics, 11.1
  ARRAY NAME PREVIOUSLY USED, 5.11
  ATTEMPT TO CHANGE LENGTH, 5.22
  ATTEMPT TO DOUBLY TYPE A VARIABLE, 5.2
  compilation, 11.1
  DIMENSIONED ARRAY TOO LARGE FOR CORE, 5.10
  DUPLICATE BLOCK NAME, 5.21
  error, in subscript use, 2.19
    form of, 11.1
  GREATER THAN THREE DIMENSIONS, 5.10
  ILLEGAL USE OF REPLACEMENT STATEMENT, 4.5
  incorrect DO forms, 6.11
  rules and, in ENTRY statement, 7.30
  SCM DIRECT RANGE, 5.12
  syntax error, 9.22
    in CLOSE statement, 9.45
    in INQUIRE statement, 9.54
    in OPEN statement, 9.43
    in READ/WRITE statement, 9.11
    in SETPASS statement, 9.57
  UNRECOGNIZED STATEMENT, 6.3
  VARIABLE DIMENSIONED IN FORMAL PARAMETER LIST, 5.13
  Diagram, flow, for DO loop, 6.43
Digits, octal, 8.22
Dimension
  declaration, 5.10
  specification, 5.11
  DIMENSION statement, 5.10
  diagnostic, 5.10
Dimensioned
  array diagnostic, 5.10
  identifier diagnostic, 5.13
  variables, 5.12
Dimensions
  assigned through parameter list, 5.13
  greater than three, in DIMENSION statement, 5.10
  variable, 5.13
    correct usage of, 7.33
    in subprograms, 6.25, 7.33
Direct
  access
    comments, 9.67
    examples of, 9.67-9.68
  range diagnostic, SCM, 5.12
  DIRECT=dir, 9.50
Display code, 2.11
D (continued)

DO-loop
active and inactive, 6.42
diagnostics of incorrect, 6.11
execution, 6.13, 6.39
flow diagram for, 6.43
FORTRAN 77, 6.37
geneneral form of, 8.3
illegal entry in, 6.18
implied, in DATA statement, 5.37
incrementation processing, 6.41
nests, 6.15
levels of, 6.16
limitations of, 6.17
optimization of, 6.21
defining statements of, 6.20
operations, sequence of, 6.39
optimization, 6.2
optimizer, turning off, 6.21
programming sample of, 6.12
range, 6.37
extended, 6.18
execution of, 6.41
rules, 6.38
structure, 6.37
subscripts
nonstandard, 6.18
standard, 6.18
within, 6.18
terminnal statement in, 6.10, 6.38
transfer out of, 6.15
DO statement, 6.9, 6.37
execution, 6.40
form of, 6.9, 6.37
iteration count in, 6.40
largest possible value in, 6.9
parameter
incrementation, 6.10, 6.40
indexing, 6.9
initial, 6.9
terminal, 6.40
rules in, 6.9
Dominant operand type, 3.18-19
mixed-mode, 4.1
order, 3.18
Double-precision
arrays, 2.21
constants, 2.8-9
nlist, 5.2
variable, 2.15
Dummy arguments, 5.36
DUPLICATE BLOCK NAME diagnostic, 5.21
Dw.d input and output, 8.18
Evaluation
- arithmetic, 3.3
- logical, 3.12
- standard, 3.3
- with optimization, 3.6

F
- Factor
  - repetition, 8.34
  - scale, 8.26

False value logical evaluation, 3.12

Field
- identification, 1.3
- short, in FORMAT, 8.15

FILE=dsi, 9.39, 9.47
FILESEQ=n, 9.40, 9.47

Floating-point decimal table, internal, 12.9
Flow diagram for DO-loop, 6.43

Formal parameters, 5.13, 5.36, 7.3

Format, 10.6-7
- control, 9.12
- declaration, 8.7
- input/output, 8.1
- specifications, 8.8, 8.34-35
- variable, 8.7, 8.36

FORMAT statement
- form of, 8.7
- use of slash (/) in, 8.32

Formatted data transfers, 9.6
FORMATTED=fmt, 9.50

Forms, FORTRAN coding, 1.1
FORM=fm, 9.40, 9.50

*FORTRAN, 1, 6.21
*FORTRAN, 2, 6.22
*FORTRAN, 3, 6.23
*FORTRAN, 4, 6.23
*FORTRAN, 13, 6.24
*FORTRAN, 18, 6.24
*FORTRAN, 60, 6.24

Function, 7.1

fwa, 9.21

Fw.d
- input, 8.14
- output, 8.14
- scaling, 8.27-28
- summary table, 8.14

G

GEN=gen, 9.52

GO TO statement, 6.2
- assigned, 6.2
- computed, 6.3
- many-branch, 6.2-3

Gw.d input and output, 8.16
Index

H

Hierarchy of arithmetic evaluation, 3.3
Hollerith
character, 2.11
constant, 2.11-12

I

ident parameters, 9.39, 9.46
Identification field, 1.3
Identifier, 2.2
  alphanumeric, 2.2
  function, 7.12
  statement, 2.3
  variable dimensioned, 5.13
  zero as, 5.20
If
  block, 6.28-29
  logic, 10.4-5
IF statement, 6.5
  arithmetic, 3-branch, 6.5
  logical, 3.13-14
  relational, 6.7-8
IF, ELSE; see ELSE IF
IF/THEN statement, 6.30
IF/THENELSE statement, 6.28
sample, 6.35
Imaginary part of complex number, 2.10
Implicit typing, 5.4-6
IMPLICIT statement, 2.14, 5.4
Implied DO-loop, 5.37
Inactive DO-loop, 6.42
Incrementation processing in DO-loop, 6.10, 6.40-41
Indefinite conditions, integer, 2.5
Index
  check, 5.11
  values, 5.11
  variable, 8.4
Infinity, positive and negative, 2.7
Initial values, 5.35-36
Input, 8.27
  data, 10.6-7
  parameters, 9.9, 9.20, 9.44
Input/Output
  formats, 8.1
  list, 8.2, 9.3
  statements, 9.1, 9.16
  synchronous, 9.12, 9.58
  terminology, 9.4
INQUIRE statement, 9.46
  error diagnostics in, 9.54
  keylist parameters in, 9.48
Integer
  arrays, 2.21
  constants, 2.3-4
  conversion, octal-decimal, 12.5
  indefinite conditions, 2.5
  multiply, 2.5
  nlist, 5.2
  overflow, 2.5
  type, 5.3
  variable, 2.14
Internal files, 9.27
In-line, 7.1
  algorithms, 7.18
I STAT=ios, 9.42, 9.45, 9.48
Iteration count, 6.40
Iw input and output, 8.19-20

J

Job, 7.1

K

Keylist parameters, 9.40, 9.48
Keywords, 9.5

L

Label
  alphanumeric, 2.16
  statement, 1.3, 2.3
  error condition, 9.40, 9.45, 9.47
Left-justified Hollerith constant, 2.11
Length
  common block, 5.22-23
  Hollerith constant, 2.11
  record, 9.13
  word, 2.14-2.15
let parameter, 5.4
Mode
character, 9.25
data, 9.10, 9.20
mixed, 3.1; see also mixed mode
of variables in masking statement, 3.16
MODE=md, 9.20, 9.51
Modular design, 7.1
Modulus, 7.21
Multiple replacement statements, 4.4
Multiply, integer, 2.5

N
NAMED=nmd, 9.49
Names
alphanumeric, 2.16
symbolic, 2.2
variable, 2.15, 5.10
NAME=fn, 9.49
NCAR
forms of synchronous I/O, 9.12
FORTRAN, 10.1
printer/card reader units, 9.14
Negative
infinity, 2.7
subscripts, 5.12
Nested DO loops, 6.20-6.21
Nesting
capacity, 6.17
levels, 6.15-16, 6.20
NEXTREC=nr, 9.52
nHf, 2.11
nlist
and clist, 5.36
double precision, 5.2
integer, 5.2
logical, 5.2
parameter
in COMMON statement, 5.20
in EQUIVALENCE statement, 5.24
in type declaration, 5.2
real, 5.2
subscripts in, 5.3
Nonexecutable statement, 5.1
Nonfatal errors, 11.16
Nonstandard subscript, 2.18, 8.3
in DO loops, 6.18
limitations of, 8.3
Nonzero test in relational IF statement, 8.3

M
Machine program, 1.1
Main program, 7.2
Mapping, array, 5.12
Masking, 3.1
expressions, 3.14-16
statement, 3.16
Maximum
arguments, 7.20
characters in
ENCOD, 9.29
output record, 8.31
core size, 5.18
record length, 9.13
MAXREC=maxr, 9.41, 9.49
Minimum arguments, 7.20
Mixed mode, 3.1
expressions, 3.17-18
replacement statements, 4.1
Loop
control processing, 6.41
DO; see DO loop
Looping and control statements, 10.6-7
Lw input and output, 8.25
N (continued)

NSTATE=ns, 9.21, 9.51  
Number
  complex, 2.10
  indefinite, 2.7
  real, 2.7
  unit, 9.39
NUMBER=un, 9.49  
Numeric characters, 2.1
NUMREC=nr, 9.52  
NWORDS=n, 9.21

O

Octal
  constants, 2.6
  conversion, 8.21
  /decimal conversion, 12.5
digits, 8.22
  variable, 2.15
Odd parity, 9.23
OPEN statement, 9.38
  error condition in, 9.40
  error diagnostics in, 9.43
OPENED=od, 9.48
Operand
  in arithmetic expressions, 3.2
  in masking expressions, 3.15
type, 3.18-19
Operator, 3.19
  in arithmetic expressions, 3.2
  relational, 3.11
Optimization
  DO loop, 6.2, 6.21
  evaluation with, 3.6
  of arithmetic sequences, 3.6
Optimizer, DO loop, 6.21
Ordering, statement, 5.1
Output, 8.27-28
  Aw, 8.23
data, 10.6-7
Dw.d, 8.18
Ew.d, 8.12
Fw.d, 8.14
Gw.d, 8.16
Iw, 8.20
Lw, 8.25
Ow, 8.22
  parameters, 9.11, 9.21
  real data, 8.16
Rw, 8.24
  statements, 9.12, 9.17
wl, 8.31
Overflow, integer, 2.5
Overlay, 5.23
Ow input and output, 8.21-22

P

param parameter, 5.7
Parameter, 9.24, 9.55
  actual, 7.4
  communication of, 7.3
diagnostic, 5.13
  example of, 5.9
  formal, 5.36, 7.3
  ident, 9.44, 9.46
  in COMMON statement, 5.20
  in DO statement
    exp1, exp2, exp3, 6.10
    initial, 6.40
    indexing, 6.9
    m1, m2, m3, 6.40
    terminal, 6.40
  in ENCODE/DECODE, 9.27
  in EQUIVALENCE statement, 5.24
  in IMPLICIT statement, 5.4
  in PARAMETER statement, 5.7
  in TYPE declaration, 5.2
  input, 9.9, 9.20, 9.44
  keylist, 9.40, 9.48
  output, 9.11, 9.44
  return, 9.42, 9.45
PARAMETER statement, 5.7
  rules for using, 5.8
Parentheses, use of, in evaluating expressions, 3.20
Parenthetical expressions, 3.3
Parity, 9.23
PASS=pass, 9.42, 9.55
PAUSE statement, 6.26
Place holder, 5.14
Points, entry; see Entry points
Positioning
  file, 9.58, 10.8-9
  statements, 9.60
Positive
  difference, 7.21
  infinity, 2.7
Powers of two, table of, 12.4
Precision, double; see Double precision
PRINT statement, 9.13
Printer unit, 9.14
Program, 7.1
PROGRAM statement, 7.2
PUNCH statement, 9.14
Punched cards, 1.4

R
Range
  direct, 5.12
  error, 2.5, 2.8
  in DO loop, 6.37, 6.41
    extended, 6.18
  of constants, 2.7
    double precision, 2.8
    integer, 2.4
  of exponent in E specification, 8.9
  of unit numbers, 9.39
READ statement, 9.12, 9.16
  asynchronous, 9.19
  diagnostics, 9.11
  format of, 9.8
  synchronous, 9.8
Reader units, 9.14
Real
  constant, 2.7
  data, 8.16
  number, 2.7
  nlist, 5.2
  part of complex number, 2.10
    type, 5.3
  variables, 2.15
RECL=rlen, 9.41, 9.49

Record
  end of, 8.32
  external, 8.2
  length, 9.13
  logical, 9.18
  new, 8.32
  output, 8.31
  structure, 9.25
  tape, 9.18
REC=rn, 9.20
Reentry into a loop, 6.19
Reference
  function, 7.13-14
  identifier, 5.24
  label, statement, 6.9
  of array elements, 2.19
  of entire array, 2.19
  subprogram, 10.2-3
Relational
  IF statement, 6.7-8
  operators, 3.11
Relative address, 5.16
Repetition factor, 8.34
Replacement, 10.4-5
  statements, 4.1
    diagnostic, 4.5
    function identifier, 7.12
    function reference, 7.14
    logical variables in, 3.14
    mixed mode, 4.1
    multiple, 4.4
Reserved computer words, 5.10
Return parameter, 9.42, 9.45
RETURN, 7.27
REWIND statement, 9.60
Right-justified Hollerith constant, 2.11
RPASS=rpas, 9.56
Rw input and output, 8.24
RWPASS=rwpass, 9.56

S
Scale factor, 8.26
Scaling, Fw.d and Ew.d, 8.27-28
SCM DIRECT RANGE diagnostic, 5.12
SEQFIL=n, 9.52
Sequential access, 9.65-68
  file positioning with, 9.58
SEQUENTIAL=seq, 9.50
S (continued)

Set, character; see Character set
SETPASS statement, 9.55-57
Share, 5.24
Shifting, 7.22
Sign, 7.21
Significant part, 2.9
Simple
expressions, 3.17, 3.19
variable, 2.16
Size
core, maximum, 5.18
limits, compiler table, 11.18
SKIPFILE statement, 9.63
Slash (/) in FORMAT statement, 8.32
Source
data, 9.2
program, 1.1
Specification
blank, 8.29
conversion, 8.8-9
dimension, 5.11
E, 8.9-10
editing, 8.8, 8.29
format, 8.8, 8.34-35
H, 2.11, 8.30
scaled, 8.26
statements, 5.1
wX, 8.29
Standard
evaluation, 3.3
FORTRAN, 2.1
subscripts, 2.17, 6.18, 8.2
Star (*) delimiter, 8.8
STATUS=sta, 9.40, 9.44
STOP statement, 6.27
Storage
allocation, 5.1, 5.28, 10.4-5
equivalent, 5.27
location, 2.20, 5.24
String substitution, 5.8
Subfield combinations in FORMAT, 8.10
Subroutine, 7.1, 7.7
subprogram, 7.7, 7.28
Subscript, 2.17
error diagnostic in use of, 2.19
nonstandard, 2.18, 8.3
in DO loop, 6.18
limitations of, 8.3
standard, 2.17, 8.2
in DO loop, 6.18
in I/O list, 8.2
negative, 5.12
subscripted, 2.18
Subscripted
subscript, 2.18
variable, 2.17-18, 5.10
Substitution, string, 5.8
Support, dataset; see Dataset support
Symbolic
name, 2.2, 5.7
representation of a variable, 2.13
Synchronous
data transfers, 9.4
input/output, 9.12, 9.58
READ/WRITE, 9.8
Syntax
error diagnostics; see Diagnostics
FORTRAN, 9.70
statement, definition of symbols in, 10.10
T

Tables
compiler, 11.18
decimal/binary position, 12.10
Ew.d summary, 8.10
FORTRAN syntax, 9.70
Fw.d summary, 8.14
octal/decimal integer conversion, 12.5
powers of two, 12.4
variable assignments, 5.16
Tape
odd and even parity in writing, 9.23
1/2" tape recorded in binary unformatted, 9.18
Terminal
parameter in DO statement, 6.40
statement in DO loop, 6.10, 6.20, 6.38
Termination statements, 6.27
Tests in relational IF statement, 6.7
Three-branch arithmetic IF statement, 6.5
Transfer
control, 7.8
to a function, 7.13
core-to-core, 9.4, 10.8-9
data, 9.1
formatted and unformatted, 9.6
synchronous and asynchronous, 9.4
out of DO loop, 6.15
subprogram reference and, 10.2-3
unconditional, of statement, 6.2
Transmission
array, 8.3-4
data, 8.1
Two-branch relational IF statement, 6.7
typ parameter in IMPLICIT statement, 5.4
Type
complex variable, 2.15
declaration, 5.1-2, 10.2-3
dominant, 4.1
double-precision variable, 2.15
Hollerith constant, 2.12
integer, 5.3
logical, 5.3
octal variable, 2.15
operand, dominant, 3.18-19
real, 5.3
variable, 2.14
Typing, implicit, 5.4-6

U
Unconditional GO TO statement, 6.2
Unformatted
binary, 9.18
data transfers, 9.6
input statements, 9.16
output statements, 9.17
UNFORMATTED=unf, 9.51
UNRECOGNIZED STATEMENT diagnostic, 6.3
Unsigned integer constants, 2.3
User catalog program, example, 9.69

V
Value
absolute, 7.19
false, logical evaluation, 2.13
index, 5.11
initial, 5.35
in DATA statement, 5.36
true, logical evaluation, 3.12
Variable
complex, 2.15
dimension, 5.13
correct usage of, 7.33
in a subprogram, 8.25, 7.33
DO, 6.37
double-precision, 2.15
format, 8.36-37
in IMPLICIT statement, 2.14
in TYPE statement, 2.14
index, 8.4
integer, 2.14
multiply-subscripted, in EQUIVALENCE
statement, 5.24
name
in DIMENSION statement, 5.10
octal, 2.15
of location, 2.13
number of characters in, 2.13
octal, 2.15
simple, 2.16, 2.18, 5.10
subscripted, 2.16
symbolic representation of, 2.13
value of, 2.13
VSN=name

W
Wf input and output, 8.30-31
Word
computer, length on common block, 5.22
length on common block, 5.22
reserved for an array, 5.10
structure of constants, 12.11
length, 2.14-15
WPASS=wpass, 9.56
WRITE statement, 9.14
asynchronous, 9.19
diagnostics in, 9.11
form of, 9.8
synchronous, 9.8
W (continued)

Writing a tape, 9.23
wx specification, 8.29

X

Y

Z

Zero
as an identifier, 5.20
test in relational IF statement,
  6.7