8-BIT SUBROUTINES
REFERENCE MANUAL
VERSION 1.0

CDC© OPERATING SYSTEMS:
SCOPE 3.4
KRONOS 2.1
<table>
<thead>
<tr>
<th>REVISION</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Preliminary edition.</td>
</tr>
<tr>
<td>(12-10-72)</td>
<td>Original printing.</td>
</tr>
<tr>
<td>(4-6-73)</td>
<td>To incorporate corrections resulting from documentation use and evaluation. Pages affected: Title page; 3-3, 3-7, 3-10, 3-19; 4-1, 4-4, 4-10, 4-14, 4-16; Comment Sheet</td>
</tr>
<tr>
<td>(3-08-74)</td>
<td>To incorporate references to NOS and KRONOS and information relative to the use of the subroutines in programs executed under those operating systems.</td>
</tr>
<tr>
<td>(11-15-74)</td>
<td>References to the NOS operating system have been deleted. This is the last scheduled revision to this manual. It will be a regularly stocked item at LDS for a minimum of 12 months after this release. All technical corrections and comment sheet responses have been incorporated.</td>
</tr>
</tbody>
</table>

REVISION LETTERS I, O, Q AND X ARE NOT USED

Control Data Corporation
Printed in the United States of America

Address comments concerning this manual to:

CONTROL DATA CORPORATION
Publications and Graphics Division
215 MOFFETT PARK DRIVE
SUNNYVALE, CALIFORNIA 94086

or use Comment Sheet in the back of this manual
LIST OF EFFECTIVE PAGES

New features, as well as changes, deletions, and additions to information in this manual are indicated by bars in the margins or by a dot near the page number if the entire page is affected. A bar by the page number indicates pagination rather than content has changed.

<table>
<thead>
<tr>
<th>Page</th>
<th>Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cover</td>
<td>—</td>
</tr>
<tr>
<td>Title Page</td>
<td>—</td>
</tr>
<tr>
<td>ii</td>
<td>D</td>
</tr>
<tr>
<td>iii/iv</td>
<td>D</td>
</tr>
<tr>
<td>v thru viii</td>
<td>D</td>
</tr>
<tr>
<td>1-1</td>
<td>C</td>
</tr>
<tr>
<td>1-2, 1-3</td>
<td>D</td>
</tr>
<tr>
<td>1-4</td>
<td>A</td>
</tr>
<tr>
<td>2-1</td>
<td>D</td>
</tr>
<tr>
<td>2-2 thru 2-9</td>
<td>A</td>
</tr>
<tr>
<td>2-10</td>
<td>D</td>
</tr>
<tr>
<td>2-11, 2-12</td>
<td>A</td>
</tr>
<tr>
<td>2-13</td>
<td>D</td>
</tr>
<tr>
<td>2-14 thru 2-23</td>
<td>A</td>
</tr>
<tr>
<td>3-1, 3-2</td>
<td>D</td>
</tr>
<tr>
<td>3-3</td>
<td>B</td>
</tr>
<tr>
<td>3-4</td>
<td>D</td>
</tr>
<tr>
<td>3-5, 3-6</td>
<td>C</td>
</tr>
<tr>
<td>3-7</td>
<td>B</td>
</tr>
<tr>
<td>3-8, 3-9</td>
<td>A</td>
</tr>
<tr>
<td>3-10</td>
<td>B</td>
</tr>
<tr>
<td>3-11, 3-12</td>
<td>D</td>
</tr>
<tr>
<td>3-13 thru 3-16</td>
<td>A</td>
</tr>
<tr>
<td>3-17, 3-18</td>
<td>C</td>
</tr>
<tr>
<td>3-19</td>
<td>B</td>
</tr>
<tr>
<td>4-1</td>
<td>D</td>
</tr>
<tr>
<td>4-2, 4-3</td>
<td>A</td>
</tr>
<tr>
<td>4-4</td>
<td>B</td>
</tr>
<tr>
<td>4-5 thru 4-9</td>
<td>A</td>
</tr>
<tr>
<td>4-10</td>
<td>B</td>
</tr>
<tr>
<td>4-11 thru 4-13</td>
<td>A</td>
</tr>
<tr>
<td>4-14</td>
<td>B</td>
</tr>
<tr>
<td>4-15</td>
<td>A</td>
</tr>
<tr>
<td>4-16</td>
<td>B</td>
</tr>
<tr>
<td>4-17</td>
<td>A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Page</th>
<th>Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-1, 5-2</td>
<td>D</td>
</tr>
<tr>
<td>5-3, 5-4</td>
<td>A</td>
</tr>
<tr>
<td>5-5</td>
<td>C</td>
</tr>
<tr>
<td>6-1</td>
<td>D</td>
</tr>
<tr>
<td>6-2</td>
<td>C</td>
</tr>
<tr>
<td>6-3</td>
<td>D</td>
</tr>
<tr>
<td>6-4</td>
<td>A</td>
</tr>
<tr>
<td>6-5</td>
<td>D</td>
</tr>
<tr>
<td>7-1 thru 7-20</td>
<td>A</td>
</tr>
<tr>
<td>8-1 thru 8-3</td>
<td>A</td>
</tr>
<tr>
<td>8-4</td>
<td>D</td>
</tr>
<tr>
<td>8-5, 8-6</td>
<td>A</td>
</tr>
<tr>
<td>A-1, A-2</td>
<td>C</td>
</tr>
<tr>
<td>A-3 thru A-6</td>
<td>A</td>
</tr>
<tr>
<td>A-7</td>
<td>C</td>
</tr>
<tr>
<td>A-8</td>
<td>A</td>
</tr>
<tr>
<td>A-9 thru A-11</td>
<td>C</td>
</tr>
<tr>
<td>B-1 thru B-5</td>
<td>A</td>
</tr>
<tr>
<td>C-1</td>
<td>A</td>
</tr>
<tr>
<td>C-2, C-3</td>
<td>C</td>
</tr>
<tr>
<td>D-1 thru D-3</td>
<td>A</td>
</tr>
<tr>
<td>E-1 thru E-4</td>
<td>C</td>
</tr>
<tr>
<td>F-1</td>
<td>D</td>
</tr>
<tr>
<td>F-2, F-3</td>
<td>A</td>
</tr>
<tr>
<td>G-1 thru G-8</td>
<td>A</td>
</tr>
<tr>
<td>H-1</td>
<td>A</td>
</tr>
<tr>
<td>Index-1</td>
<td>C</td>
</tr>
<tr>
<td>Index-2</td>
<td>A</td>
</tr>
<tr>
<td>Index-3</td>
<td>C</td>
</tr>
<tr>
<td>Index-4</td>
<td>C</td>
</tr>
<tr>
<td>Index-5</td>
<td>C</td>
</tr>
<tr>
<td>Index-6</td>
<td>C</td>
</tr>
<tr>
<td>Index-7</td>
<td>C</td>
</tr>
<tr>
<td>Index-8</td>
<td>A</td>
</tr>
</tbody>
</table>

Page    | Revision |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Comment Sheet</td>
<td>D</td>
</tr>
<tr>
<td>Mailer</td>
<td>—</td>
</tr>
<tr>
<td>Back Cover</td>
<td>—</td>
</tr>
</tbody>
</table>
The 8-Bit Subroutine package is a group of user-callable subroutines for processing 8-bit data on the CONTROL DATA® CYBER 70 Models 72, 73, and 74 and 6000 Series computers. The subroutines are accessed by FORTRAN CALL and COBOL ENTER statements and by COMPASS return jump instructions. No modification to FORTRAN Extended, COBOL, or COMPASS is required.

Input/output operations are under the control of Record Manager for all the subroutines except COPY8P. A knowledge of Record Manager and FORM (File Organizer and Record Manager) is desirable. A general knowledge of character sets, IBM files (record types and blocking formats), and CDC files is essential. We assume knowledge of COBOL or FORTRAN and hexadecimal notation.

This reference manual makes numerous references to the SCOPE 3.4 operating system and illustrates some of its features. Users of KRONOS 2.1 will notice significant differences between SCOPE and KRONOS in the following areas:

Use of nT parameter on SCOPE 3.4 job card

Use of ACCOUNT card following KRONOS 2.1 job card

Use of RESOURCE card in KRONOS 2.1 jobs when more than one tape is required

Differences in REQUEST card syntax and parameters

Differences in DISPOSE card syntax and parameters

Differences in the definition and use of flag cards for reading SCOPE 3.4 free-form binary cards versus KRONOS 2.1 literal input

KRONOS 2.1 requirement to punch columns 79-80 of job card to select IBM 026 or 029 punch code interpretation

Use of an extended print file in conjunction with SCOPE 3.4 usage of 595-6 print train; such capacity does not exist in KRONOS 2.1

Each of these differences should be resolved by referring to either the SCOPE 3.4 or KRONOS 2.1 operating system reference manuals.
This reference manual deals only with the manipulation of files; the opening and closing of the files is done by the interface language.

The following CDC publications may be of interest to the user:

<table>
<thead>
<tr>
<th>Titles</th>
<th>Publication Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCOPE 3.4 Reference Manual</td>
<td>60307200</td>
</tr>
<tr>
<td>KRONOS 2.1 Reference Manual</td>
<td>60407000</td>
</tr>
<tr>
<td>FORTRAN Extended Reference Manual</td>
<td>60305601</td>
</tr>
<tr>
<td>COBOL Reference Manual</td>
<td>60384100</td>
</tr>
<tr>
<td>COMPASS Reference Manual</td>
<td>60360900</td>
</tr>
<tr>
<td>Record Manager File Organization User's Guide</td>
<td>60359600</td>
</tr>
<tr>
<td>Record Manager Reference Manual</td>
<td>60307300</td>
</tr>
<tr>
<td>FORM Reference Manual</td>
<td>60307000</td>
</tr>
<tr>
<td>CYBER LOADER</td>
<td>60344200</td>
</tr>
</tbody>
</table>

The following IBM publications may be of value when it is necessary for the user to understand the nature of the file data being processed by the 8-bit subroutines:

- IBM 360/370 FORTRAN IV Programmer's Guide
- IBM 360/370 ANSI COBOL Programmer's Guide
- IBM 360/370 PL/I Programmer's Guide

CDC manuals can be ordered from Control Data Literature Distribution Services, 8001 East Bloomington Freeway, Minneapolis, Minnesota 55420.

This product is intended for use only as described in this document. Control Data cannot be responsible for the proper functioning of undescribed features or parameters.
CONTENTS

1 INTRODUCTION
   Maintaining 8-Bit Significance 1-1
   Overview of Routines 1-2
   Notations Used in Calling Sequences 1-3

2 DATA CONVERSION STRINGS
   String Position 2-1
   Field Alignment 2-1
   Conversion String Syntax 2-2
   Conversion Items 2-2
      Repeat Count 2-3
      Simple Item Conversion 2-3
   Conversion Strings Used As Conversion Items 2-5
   Selector Expressions 2-5
      Item Locators 2-5
      Value Field 2-6
      Relationships 2-7
   Conversion Specifications 2-8
   Conversion Strings 2-9
      Conversion String Punctuation 2-9
      Nested Conversion Strings 2-10
   Special Conversion Rules 2-12
   Conversion Tables 2-14

3 INPUT-OUTPUT SUBROUTINES
   Input/Output Parameters 3-1
   XFILE File Definition 3-2
      File-String Parameters 3-3
      Workspace Area Size 3-5
      File Usage 3-5
   XWRITE Write a Record 3-5
      Forcing Termination of File Output 3-7
   XREAD/XREREAD Read a Record 3-8
      Skipping Records 3-11
   Error Handling 3-12
   COBOL Input/Output Examples 3-12
      COBOL Tape Reading Example 3-13
      Conversion-String Variations for Input 3-13
      COBOL Tape Writing Example 3-15
      Conversion-String Variations for Output 3-15
   FORTRAN Input/Output Examples 3-17
   COMPASS Input/Output Examples 3-18

4 UTILITY SUBROUTINES
   Utility Subroutine Parameters 4-1
   XCOMP String Comparison 4-1
      Collating Sequence 4-5
      Comparison Examples 4-5
   XMOVE String Move 4-7
      Move Examples 4-11
   XPACK String Compression 4-12
      Pack Examples 4-14
   XPAND String Expansion 4-15
      Expand Examples 4-17

5 COPY8P PROGRAM
   Calling COPY8P 5-1
   COPY8P Options 5-1
   Character Set Restrictions 5-3
      COPY8P Print Conversion Tables 5-4

6 SYSTEM INTERFACES
   Operating System Interface 6-1
      Tape Files 6-1
      Card Files 6-2
      Print Files 6-3
   Record Manager Interface 6-3
      Tape Files 6-3
      Card Files 6-4
      Print Files 6-4
   Loader Interface 6-4
      Omitting Unneeded 8-Bit Modules 6-4
      Loader Considerations 6-5

7 COBOL USAGE
   Internal CDC COBOL Data Formats 7-1
   Example Conversions -- IBM to CDC Files 7-6
   Sample CDC COBOL Job 7-15

8 FORTRAN EXTENDED USAGE
   IBM FORTRAN Data Formats 8-1
   CDC FORTRAN Extended Data Formats 8-3
   Sample FORTRAN Job 8-4
## Appendixes

<table>
<thead>
<tr>
<th>Letter</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>STANDARD CDC CHARACTER SETS</td>
<td>A-1</td>
</tr>
<tr>
<td>B</td>
<td>IBM TAPE FILE RECORD AND BLOCK FORMATS</td>
<td>B-1</td>
</tr>
<tr>
<td>C</td>
<td>IBM DATA FORMATS</td>
<td>C-1</td>
</tr>
<tr>
<td>D</td>
<td>CDC DATA FORMATS</td>
<td>D-1</td>
</tr>
<tr>
<td>E</td>
<td>FREE-FORM BINARY CARD FORMAT</td>
<td>E-1</td>
</tr>
<tr>
<td>F</td>
<td>PRINT FORMAT – 595-6 PRINT TRAIN</td>
<td>F-1</td>
</tr>
<tr>
<td>G</td>
<td>DIAGNOSTIC MESSAGES</td>
<td>G-1</td>
</tr>
<tr>
<td>H</td>
<td>8-BIT SUBROUTINES/FORM COMPARISON</td>
<td>H-1</td>
</tr>
</tbody>
</table>

## Tables

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-1</td>
<td>Valid T and m Values</td>
<td>2-14</td>
</tr>
<tr>
<td>2-2</td>
<td>Default Conversions If Tm2 Is Not Specified</td>
<td>2-15</td>
</tr>
<tr>
<td>2-3</td>
<td>Comparison Modes</td>
<td>2-16</td>
</tr>
<tr>
<td>2-4</td>
<td>Conversion Rules</td>
<td>2-17</td>
</tr>
<tr>
<td>5-1</td>
<td>Print Conversion Tables</td>
<td>5-4</td>
</tr>
<tr>
<td>7-1</td>
<td>IBM COBOL Tm Values</td>
<td>7-3</td>
</tr>
<tr>
<td>7-2</td>
<td>CDC COBOL Tm Values</td>
<td>7-5</td>
</tr>
<tr>
<td>8-1</td>
<td>IBM FORTRAN Constant and Variable Sizes</td>
<td>8-1</td>
</tr>
<tr>
<td>8-2</td>
<td>IBM FORTRAN Tm Values</td>
<td>8-2</td>
</tr>
<tr>
<td>8-3</td>
<td>CDC FORTRAN Constant and Variable Sizes</td>
<td>8-3</td>
</tr>
<tr>
<td>8-4</td>
<td>CDC FORTRAN Tm Values</td>
<td>8-4</td>
</tr>
</tbody>
</table>
INTRODUCTION

Programmers using either COMPASS, COBOL, or FORTRAN Extended may use the 8-bit subroutines described in this reference manual to perform any of the following three functions:

To convert IBM 360/370 sequential 8-bit tape or card files to CDC internal format, maintaining 8-bit significance where necessary.

To perform data moves, comparisons, packing, and expanding of the converted data in which 8-bit significance has been maintained.

To copy an IBM 360/370 print file to a 595-6 extended print file, under SCOPE, maintaining upper and lower case characters. Printing of such a file requires the use of the 95-character ASCII character set available in the CDC 595-6 print train. Use of an extended print file with the 595-6 print train is not available under KRONOS.

EBCDIC is the Extended Binary Coded Decimal Information Code developed by IBM to represent a set of 256 characters. It is an 8-bit code used in sequential 8-bit tape files produced on IBM 360/370 computer systems. A complete listing of the EBCDIC character set appears on page A-6.

ASCII is the American Standard Code for Information Interchange. It is an 8-bit code representing a prescribed set of 128 characters. The full ASCII character set appears in table A-5. A standard 64-character subset of ASCII has been adopted by Control Data Corporation to comply with Federal regulations and appears on page A-4. It is a 6-bit code that can be used on existing 6-bit equipment and software systems.

Display code is a 6-bit code established for the representation of a 64-character set on external display devices, such as line printers and cathode ray tube display units. A list of display codes in octal form with corresponding graphics appears on page A-7.

IBM 360/370 8-bit sequential tape files may contain:

- EBCDIC character string data
- ASCII character string data
- Bit stream data
- Arithmetic data in IBM format

Such tape files may be in any one of the record/block format types described in appendix B. A full description of IBM arithmetic data appears in appendix C.
The internal CDC format used in conjunction with the 8-bit subroutines may contain the following data types:

6-bit display code character string data

12-bit ASCII character string data

12-bit EBCDIC character string data

Bit stream data

Arithmetic data in CDC format

The 12-bit ASCII and EBCDIC data format is unique to 8-bit subroutines. Both 12-bit data formats and CDC format arithmetic data are defined in appendix D.

An IBM 360/370 print file is written entirely in either 8-bit ASCII or 8-bit EBCDIC character string data. CDC extended print files consist of an ASCII code comparable to IBM 360/370 print file code. It contains a maximum of 95 characters; each is represented internally in a 12-bit byte. The code is printed through use of the CDC 595-6 print train. A list of the 95-character ASCII code appears on page 5-4, in the discussion of the COPY8P utility program. CDC extended print files are discussed further in appendix F.

An IBM 360/370 sequential 8-bit tape can be written by the 8-bit subroutines with a few restrictions. Whenever a block is written, the block size is rounded up to the next multiple of 12 bits as it goes through the PP channel. The block size is rounded up again to the next multiple of 8 as it gets written on the tape. This may cause up to two extra characters to be written on the tape.

For example, suppose a block to be written contains eight 8-bit characters or 64 bits, the PP rounds this to 72 bits and the tape drive further rounds it to 80 bits for a full frame. Eighty bits equals ten 8-bit characters.

When reading this tape using the 8-bit subroutines, the extra characters are ignored as padding.

MAINTAINING 8-BIT SIGNIFICANCE

Maintaining 8-bit significance in data converted from IBM 360/370 tape files is necessary when such files contain character codes not included in the CDC 64-character graphic set. Lower case characters and several special characters, such as @ ? ! and # are included in the IBM EBCDIC and ASCII character sets used (see page A-2), but not in the CDC 64-character set. Control characters included in the 8-bit ASCII and EBCDIC character sets (page A-6) are not members of the 95-character ASCII graphic set used by the CDC 595-6 print train. These characters can be processed by the 8-bit subroutines, but such control characters cannot be printed on CDC equipment. The user must decide the necessity of maintaining special character codes. In many cases, the upper case equivalent (page A-7) will suffice.

In a number of cases, 8-bit significance need not be maintained. Files containing only characters that appear in the CDC 64-character set will convert to CDC 6-bit display codes. Files containing packed decimal data, in which each digit occupies 4 bits, will convert to CDC 6-bit numeric display fields. IBM files containing binary arithmetic data can be converted to CDC binary arithmetic or display numeric data per user specification. For IBM arithmetic data that does not exceed CDC double precision format, accuracy need not be maintained for 8-bit significance. When double precision significance is exceeded, accuracy can be maintained by using bit image conversion; but the user must provide his own routines to process such data.
OVERVIEW OF ROUTINES

The 8-bit subroutines fall into two distinct groups:

Input/output subroutines that operate on a record-by-record basis, providing translation capabilities between internal and external data types and character sets.

Utility subroutines that manipulate, compress, or expand character strings in display code, ASCII, or EBCDIC forms.

A stand-alone program (COPY8P) provides the capability to copy an IBM 360/370 print file into a CDC compatible print file without loss of 8-bit (upper and lower case) significance.

Input/output subroutines:

XFILE Must be used to define a file for subsequent use by the 8-bit input/output subroutines; it performs no input/output function by itself.

XREAD XREREAD Reads or rereads one record from an input file and places the data into the user's buffer area. Optional data conversion may be performed during reading or rereading.

XWRITE Writes one record at a time to an output file; optional data conversion may be performed as specified.

Utility subroutines include the following:

XPACK Packs 8-bit data from 12-bit form into seven characters per 60-bit word for file storage.

XPAND Unpacks 8-bit data (packed into 60-bit words for file storage by the XPACK subroutine) into 12-bit form (5 characters per 60-bit word) for subsequent internal processing.

XMOVE Moves character strings internally and optionally converts ASCII/EBCDIC/Display Code during the move.

XCOMP Compares two strings of like or differing character sets. Status information indicating the result of the comparison is returned to the user.

The print routine COPY8P is a stand-alone control-card callable program not related to the subroutines described above. It copies an IBM 360/370 print file to a 595-6 extended print file without loss of 8-bit significance.
NOTATIONS USED IN CALLING SEQUENCES

UPPER CASE  
words are reserved words. They must be spelled correctly and may not be used in a source program except as specified in the calling sequences.

Lower case words  
are generic terms which represent the words or symbols supplied by the user. When generic terms are repeated in a calling sequence, a number or letter is appended to the term for identification.

[ ] Brackets  
enclose optional portions of a calling sequence. All entries within brackets may be omitted or included at the user's option.

{ } Braces  
enclose two or more vertically stacked items in a calling sequence. When only one of the enclosed items must be used; also enclose required entries that may be repeated.

... Ellipses  
immediately following a pair of brackets or braces indicate that the enclosed material may be repeated at the user's option.

:=  
represents the phrase "is defined as."

Punctuation symbols shown in the calling sequences are required, unless enclosed in brackets and specifically noted as optional.
Data conversion strings are used as input parameters to the XREAD, XREREAD, and XWRITE subroutines. The conversion string specifies how data items in a record are to be translated. Through the data conversion string, the user may specify conversion between any IBM 360/370 data format and CDC internal data format.

STRING POSITION

Each record in the file is considered to be a string of variable length bytes with the length determined by the storage device in use. When the file is stored on tape, the string contains 8-bit bytes; on card or print files, the string contains 12-bit bytes; and when stored internally, the string contains 6-bit bytes. In 6-bit or 8-bit bytes, bits within each byte are numbered 1 through 6 or 8, from left to right. In 12-bit bytes (card input files) bits are numbered 1 to 12, from left to right.

For binary card files, all 12 bits are used. These card files are described in section 6.

FIELD ALIGNMENT

When data conversion is initiated, internal pointers are established for the source and destination record areas, each initially pointing to bit 1 of byte 1 of its record string. These bits are the initial ‘next’ field positions. Single quotes are used, as the word ‘next’ has special meaning in this regard.

When a ‘next’ source item is converted to a ‘next’ destination item, these pointers may be modified as follows:

1. Prior to conversion, if the bit pointer for a byte does not equal 1, it is set to 1; and character position is incremented by 1 (rounded up to the next byte). If the destination pointer is so affected, skipped bit positions are filled with binary zeros.

Exception: No rounding takes place for a type B (bit) source or destination item.

2. When conversion is complete, the pointers are updated to point to the bit succeeding the last bit read or written – the ‘next’ field position. When conversion terminates mid-word, the remainder of the word is unchanged.

Alignment never is forced to a boundary more significant than a byte position. If word boundary or other alignment is needed, the proper fill items must be supplied explicitly. Data alignment requirements are given in section 7; IBM in table 7-1, CDC in table 7-2.
CONVERSION STRING SYNTAX

A conversion string consists of 6-bit Display Code characters constructed in accordance with the following rules. With the exception of literal string parameters, blanks are ignored and may be used freely to improve readability. The notations used in the definitions are given in section 1.

conversion-string:=
(conversion-specification-1 ;conversion-specification-2) . . .

conversion-specification:=
[selector expression:] conversion-item-1 [, conversion-item-2] . . .

selector-expression:=
{ item-locator-1 relationship value }
{ item-locator-1 relationship item-locator-2 }

conversion-item:=
[repeat count] 
{ simple-item-conversion }
{ conversion-string }

A conversion-string may contain one or more conversion specifications. A conversion-specification consists of an optional selector expression followed by one or more conversion-items. A conversion-item may consist of a simple-item-conversion or a conversion-string. A simple-item-conversion is the only utility that can cause data translation. Since a conversion-item may consist of a conversion-string, nested conversion strings are legal. Nesting may occur up to seven levels.

Multiple conversion specifications are separated by semicolons. A selector expression is separated from its conversion items in a conversion-specification by a colon. Multiple conversion items are separated by commas. The scope of a selector expression is a single conversion-specification and is terminated by a semicolon; when the selector expression is true, the semicolon causes the rest of the conversion string, up to the matching right parenthesis, to be ignored.

A conversion-item may comprise part or all of a conversion-specification or a conversion-string. The most basic form is a conversion-item which specifies only one conversion; this form would be a simple-item-conversion.

CONVERSION ITEMS

Conversion items provide directions for translating data items from a source record to a destination record. The items may be dependent upon selector expressions which determine whether or not conversion is to take place.

Conversion items are written:

[repeat count] simple-item-conversion

[repeat count] conversion-string

2-2
REPEAT COUNT

A decimal integer is used as the optional repeat count to indicate the number of times the conversion item is to be repeated. This form is equivalent to writing the conversion item n times, separated by commas. No repetition occurs if the repeat count is zero or omitted.

SIMPLE ITEM CONVERSION

A simple item conversion specifies how the 'next' source record field is to be translated to the 'next' destination record field. The 'next' field is defined under Field Alignment. Only a simple-item-conversion specification causes data to be converted. Other parts of the conversion string provide control information, as they determine the kind of conversions to be performed.

A simple-item-conversion must be written in one of the following formats:

Format 1: Tm₁ Tm₂ Format 2: Tm₁ Format 3: Q

Format 1

Tm₁ must be a valid item descriptor type (table 2-1) for the source medium, which may be a tape file, internal file, or card file. Tm₂ must be a valid item descriptor type for the destination medium, which may be a tape file, internal file, print file, or card file. Table 2-4 gives translation rules for all possible combinations of simple-item-conversion.

Examples of format 1:

\[
\begin{array}{c}
\text{X5X5} \\
\text{Tm₁} \quad \text{Tm₂}
\end{array}
\]

Translates five 8-bit characters on an ASCII tape to five 6-bit internal Display Code characters.

\[
\begin{array}{c}
\text{X5C5} \\
\text{Tm₁} \quad \text{Tm₂}
\end{array}
\]

Translates five 8-bit characters on an EBCDIC tape to five 12-bit internal EBCDIC characters.

\[
\begin{array}{c}
\text{GU} \\
\text{Tm₁} \quad \text{Tm₂}
\end{array}
\]

Translates one IBM 64-bit integer on tape to a 60-bit word containing a CDC unnormalized floating point number.

\[
\begin{array}{c}
\text{B60I} \\
\text{Tm₁} \quad \text{Tm₂}
\end{array}
\]

Translates a 60-bit stream on tape to an internal 60-bit integer field.
60B6B10
repeat count \[\text{Tm}_1 \text{Tm}_2\]

Makes 60 moves in which consecutive internal 6-bit fields are moved to consecutive 10-bit fields on a tape file. Each 10-bit destination field will contain 4 bits of binary zero fill on the right.

4C5X10
repeat count \[\text{Tm}_1 \text{Tm}_2\]

Translates 4 times consecutive 5-character, internal 12-bit EBCDIC fields to consecutive 10-character, 8-bit fields on an ASCII tape. Each destination field will contain 5 ASCII space characters as fill on the right.

X80C80
repeat count \[\text{Tm}_1 \text{Tm}_2\]

80X1C1

Translates an 80-character field on tape containing 8-bit EBCDIC characters to an internal 80-character field containing 12-bit EBCDIC characters. These two examples result in the same data conversion, as they are logically equivalent. The former will produce more rapid conversion, since it is faster to move 80 characters at once rather than one character at a time repeated 80 times.

Format 2

\text{Tm}_1\text{ is defined the same as in format 1. A default value is selected for the absent }\text{Tm}_2\text{ descriptor, as specified in table 2-2. Translation rules for all possible simple item conversion combinations appear in table 2-4.}

Examples of format 2:

B60
repeat count \[\text{Tm}_1\]

Moves a 60-bit internal CDC field to a 60-bit IBM tape file field.

U
repeat count \[\text{Tm}_1\]

Translates a 60-bit internal CDC unnormalized floating point number to an IBM 32-bit floating point field.

40 P5
repeat count \[\text{Tm}_1\]

Translates sequential 9-digit IBM packed decimal fields to internal CDC 10-digit signed overpunch numeric 6-bit Display Code fields 40 times.
Translates 10 internal CDC 12-bit ASCII characters to 10 extended print format characters.

Format 3

The Q (Quit) specification is not a true conversion, but a control code. Execution of a Q specification terminates all conversion for the record. Conversion up to that point is not lost, but no further conversion takes place.

CONVERSION STRINGS USED AS CONVERSION ITEMS

Wherever a simple conversion item is allowed, a conversion string may be used, following the same syntax rules for conversion strings. This feature allows specification of alternative conversions at interior positions of a record, such as a fixed initial record segment followed by a variable format segment, or where certain alternatives may themselves have alternatives.

Conversion strings may occur as items within conversion strings, to a maximum depth of seven levels. Examples of nested conversion strings appear later in this section.

SELECTOR EXPRESSIONS

A selector expression is written in one of two formats:

item-locator-1 relationship value field

item-locator-1 relationship item-locator-2

A selector expression appears in a conversion specification to indicate a relational test is to be made. If the result of the test is true, all conversion items associated with the selector expression are executed. If the result is false, all associated conversion items are ignored.

ITEM LOCATORS

Item locators specify which data fields in the current source record are to be used in the relational test. Item locators may be written in the selector expression in any of the following forms:

Tm  iTm  i/wTm

T is an item type specification; it may be any value representing a legal data type for the record media.

m is a decimal integer specifying the size in bytes (12, 8, 6, or 1-bit if a type B item) of a variable length data item whose size is not determined uniquely by type. If the data item cannot be variable in length, the m specification must be omitted. When m is omitted for a variable length data item, it is assumed to be 1.
Tables 2-1 and 2-2 contain allowable and default values for T and m.

i is a byte index, absolute or relative. As an absolute index, i must be written as an unsigned decimal integer. As a relative index, i is written as a signed integer, the sign denoting the direction of the move. If i is omitted, the index is assumed to be plus zero, designating the current byte.

The current byte, from which relative positions are computed, is the current 'next' byte position. The current bit position within a byte is set to 1 if it is not already equal to 1. (See Field Alignment.)

When the item type specification is for bit fields (B), the item locator may have the form i/wTm, where i is a byte index as described above, and w is an absolute bit position within the byte (the leftmost bit in the byte is numbered 1). The value given for w must not exceed the bit size of the byte in its source medium: 6 bits in memory, 8 bits on tape, 12 bits on cards.

VALUE FIELD

Selector expressions permit the user to test the relationship between the contents of an item-locator field and the contents of a value field or another item-locator field. The value field describes a literal character string or a numeric value to be used in the comparison.

A character string is written in the value field as a literal enclosed in identical delimiter characters which are not considered part of the string. Either * or $ may be used as the delimiter. When the literal contains either * or $, the other character should be used as the delimiter pair. If the delimiter character must be part of the string, each enclosed occurrence must be doubled. Within the string, doubled delimiters are counted as a single character. Blanks within delimiters are retained.

Example:

String ABC*DEF

may be specified as: $ABC*DEFS$

or: *ABC**DEF*

A literal string must not exceed 80 characters, excluding delimiters, and may be composed of any characters in the display code character set.

A numeric value is written in the value field in a form that closely follows the numeric notation used in FORTRAN. The general format of such a numeric field is:

\[
\left[\left(\begin{array}{c}
+ \\
- \\
\end{array}\right)\right. \text{digit-1}[\text{digit-2}] \ldots \left[\left(\begin{array}{c}
+ \\
- \\
\end{array}\right)\right. \text{digit-4}[\text{digit-5}] \ldots ]
\]

If E is present, the decimal point must also appear.
Numeric value may be expressed in any of the following forms:

\[ \pm n \quad \pm n.n \quad \pm n.E \pm s \quad \pm n.nE \pm s \quad \pm nE \pm s \]

where \( n \) is the numeric value and \( s \) is the value of the exponent. The omission of a plus sign implies a positive value or exponent.

Examples:

\begin{align*}
0 & \quad 452.E6 & \text{(represents 452,000,000)} \\
2.5 & \quad -818.62E3 & \text{(represents -818,620.0)} \\
-10. & \quad .57E-10 & \text{(represents .000,000,000,057)}
\end{align*}

RELATIONSHIPS

Relationships for which elements in a selector expression may be evaluated are expressed by the following mnemonics:

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

Only the EQ and NE relationships are legal for value fields containing character strings (literals).

Before a comparison between the elements in the selector expression can be performed, both elements must be reduced to a common mode: either a character string literal or a numeric value (table 2-3). Numeric quantities are kept to an accuracy of at least 96 bits. A and C character strings are folded to 6-bit form for comparison against X strings (6-bit). Folding is the process of mapping more than one source character to a single destination character. Upper and lowercase alphabetic characters are mapped to a single uppercase character set. The table on page A-7 provides the translations for display code, EBCDIC, and ASCII characters that would occur during mapping. Shorter strings are treated as if they were extended on the right with blanks so that both strings will be the same length. (Exception: a string deriving from a bit string will be extended with zeros.)

Examples of Selector Expressions:

\[
\begin{array}{ccc}
X6 & \text{EQ} & SABCDEFs \\
Tm & \text{condition} & \text{value string}
\end{array}
\]

item-locator-1

A string of six 8-bit characters in a tape file record are compared with the literal ABCDEF; if they are equivalent, the associated conversion item list is processed.
The current byte in the record begins a long floating point field which is compared with the constant numeric value \(-467.0\); if the item tested is less than the constant value, the associated conversion item list is processed.

\[
\begin{align*}
\text{item-locator-1} & \quad \text{LT} \quad \text{value} \\
\text{condition} & \quad -467E+02
\end{align*}
\]

One 8-bit character in byte 6 of the record is compared with one character in byte 10 of the record. If equal, the associated conversion item list is processed.

\[
\begin{align*}
\text{item-locator-1} & \quad \text{condition} \quad \text{item-locator-2} \\
\text{EQ} & \quad \text{EQ} \\
+4W & \quad -SG
\end{align*}
\]

A full-word integer starting 4 bytes beyond the current byte position is compared with the double-word integer starting 8 bytes preceding the current byte. If equal, the associated conversion item list is processed.

\[
\begin{align*}
\text{item-locator-1} & \quad \text{condition} \quad \text{item-locator-2} \\
\text{EQ} & \quad \text{EQ} \\
6/4B2 & \quad -10S
\end{align*}
\]

A 2-bit field starting with bit 4 of byte 6 is compared with the binary literal value 10. If equal, the associated conversion item list is processed.

**CONVERSION SPECIFICATIONS**

A conversion specification consists of an optional selector expression followed by a list of conversion items. The selector expression is a conditional expression which must be true for the associated conversion items to be executed. If the selector expression is missing, the conversion specification is treated as though it were prefixed by a selector expression which is always true.

The conversion items of a conversion specification are executed in sequence, left to right.

```
Format:=[selector-expression:]conversion-item-1 [conversion-item-2] . . .
```
Examples:

X80X80
   conversion-item-1
   conversion-specification

X5X10,X10X5
   conversion-item-1 conversion-item-2
   conversion-specification

P5 EQ -456 : P5S9
   selector expression conversion-item-1
   conversion-specification

P3 GT 0 : 6P3Z10 , 60B8B6
   selector expression conversion-item-2
   conversion-item-1

CONVERSION STRINGS

Format: (conversion-specification-1 [; conversion-specification-2] . . . )

A conversion string is parenthesized and consists of one or more alternative conversion specifications. During execution, when a conversion string is encountered, each conversion specification is tested in turn, from left to right, until one is found with a selector expression that is true. That conversion specification is executed and all alternatives to its right in the conversion string are ignored. If none is true, no conversion is performed.

Selection of the alternative to be executed occurs anew on each entry to the conversion string.

CONVERSION STRING PUNCTUATION

A colon separates a selector expression from its associated conversion items in a conversion specification; multiple conversion items are separated by commas. The scope of a selector expression is a single conversion specification, and it is terminated by a semicolon. When the selector expression is true, the semicolon causes the rest of the parenthesized conversion string, up to the matching right parentheses, to be ignored. This feature is of special significance when items are nested in parentheses in a conversion string. When a conversion item is executed, all remaining parts of the conversion string are ignored.
NESTED CONVERSION STRINGS

A conversion string may be used wherever a simple item conversion is allowed in a conversion item. Conversion strings may occur within conversion items to a maximum depth of seven levels.

This feature permits specification of alternate conversions at various positions within a record, such as a fixed initial record followed by a variable format segment, or where certain alternatives may themselves have alternates.

Conversion String Examples:

All valid T and m values are given in table 2-1 for conversion items, normally written Tm₁ Tm₂. When the default value given in table 2-3 is acceptable, Tm₂ may be omitted. In many cases, m must not be specified.

Table 2-4 gives further details concerning conversion for T values.

The following examples illustrate elements of a conversion string:

\[
(X80) \\
\text{conversion-string} \\
\hline
(4B6B8 \quad, \quad X0X10) \\
\text{conversion-string} \\
(\quad X1 \text{ EQ $AS$} \quad : \quad X1X0 \quad , \quad 50H1 \quad ; \quad X1X0 \quad , \quad 10H1 \quad ) \\
\text{conversion-specification-1} \\
\text{conversion-specification-2} \\
\hline
(\quad X1 \text{ EQ 1 : 10W1} \quad ; \quad X1 \text{ EQ 2 : 20W1} \quad ; \quad 50W1 \quad ) \\
\text{conversion-specification-1} \\
\text{conversion-specification-2} \\
\text{conversion-specification-3} \\
\hline
(\quad X20 \quad , \quad (X4 \text{ EQ *MASH* : X10X0 : X10}) \quad ) \\
\text{conversion-item-1} \\
\text{conversion-specification-1} \\
\text{conversion-string used as conversion-item-2} \\
\text{conversion-specification-2} \\
\text{conversion-specification-1} \\
\text{conversion-string}
All conversion items are based on the concept that data is moved from the source medium to the destination medium. The type of input (ASCII or EBCDIC, tape or card) or output must be specified by the code parameter in an XFILE call. In the conversion item, a T value of X specifies source media characters for a read operation or destination coded characters for a write operation.

The following examples assume a 9-track tape containing only 8-bit EBCDIC characters as the external source of data to be read to an internal record. The tape contains multiple repetitions of the alphabet in uppercase only:

<table>
<thead>
<tr>
<th>Example</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(X5C5)</td>
<td>Moves the 5 characters ABCDE from tape to internal record. X5 specifies the first 5 characters on the tape; C5 specifies the first 5 characters of the internal record as 12-bit EBCDIC.</td>
</tr>
<tr>
<td>(X5A5)</td>
<td>Converts five 8-bit EBCDIC characters to five 12-bit ASCII characters.</td>
</tr>
<tr>
<td>(X1 EQ <em>A</em> : X5C5)</td>
<td>Because the first character on the tape is A, the first 5 characters ABCDE are moved from the 8-bit EBCDIC tape record to the 12-bit EBCDIC internal format record.</td>
</tr>
<tr>
<td>(X1 EQ <em>1</em> : X5C5)</td>
<td>Since the first character on the tape is not 1, the conversion item is ignored.</td>
</tr>
<tr>
<td>(X1 EQ <em>1</em> : X5C5 ; X1 EQ <em>A</em> : X3C3)</td>
<td>Since the first selector expression is false, the X5C5 conversion item is ignored. The second selector expression is true; therefore, the characters ABC are moved.</td>
</tr>
<tr>
<td>(X3C3,(X1 EQ $D$ : Q ; X3C3),X5C5)</td>
<td>The first 3 characters ABC are moved. Since the fourth character is D, the Q conversion item terminates conversion string execution at that point. The selector expression refers to the current byte of the source record which contains the character D.</td>
</tr>
<tr>
<td>(3X1 EQ <em>C</em> : X5C5)</td>
<td>Since the third character in the source record is C, the first 5 characters ABCDE are moved from tape to internal 12-bit EBCDIC format field.</td>
</tr>
<tr>
<td>(+3X1 EQ <em>D</em> : X5C5)</td>
<td>The selector expression is true, as D is the fourth character in the record; therefore, the first five characters ABCDE are moved.</td>
</tr>
<tr>
<td>(X3C3,(-3X1 EQ <em>A</em> : X23C23))</td>
<td>The first 3 characters on tape record ABC are moved before the first character on the record is tested. Since the first character is A, the selector expression is true; and the remaining 23 characters in the alphabetic sequence are moved.</td>
</tr>
</tbody>
</table>
Example

(X26C26,-26X26 EQ X26 : 2X26C26)

Explanation

Moves one entire alphabet group from the tape to the internal record then compares the next 26 characters with the first 26. Since they match, two more entire alphabet groups are moved to the internal record.

SPECIAL CONVERSION RULES

Rules pertaining to all possible conversions appear in table 2-4. Some capabilities deserving special emphasis are itemized below.

SPECIAL CONVERSION, BIT TO STRING

When a bit field is converted to a character string, the result is a string equal in length (measured in characters) to the bit field (measured in bits). Conversion is left to right, each zero bit is translated to the character 0 and each one bit to the character 1.

SPECIAL CONVERSION, BIT TO NUMERIC

When a bit field is converted to a numeric value, the bit field is considered to be a positive binary integer. The binary point is assumed to follow the rightmost bit of the field.

SPECIAL CONVERSION, STRING TO NUMERIC

This situation can arise when a literal string or an X, C, or A item is to be compared with a numeric item. It must be a character string following the rules for a numeric value as described under Value Field. An error will result if the string is not in this format. Spaces in the string are ignored.

CHARACTER SKIPPING AND BLANK/ZERO FILL

To specify bit or character skipping, the source field size must be specified as greater than destination size in the conversion items. For example:

B10B0 causes 10 bits to be skipped.

X5X0 causes 5 characters to be skipped.

X10X5 causes 5 characters to be transferred and the next 5 to be skipped.
To insert blanks or zeros in the destination record, destination field size must be greater than the source field size in the conversion item. For example:

XOC5 causes 5 EBCDIC spaces to be placed in the destination field.

B0B60 causes 60 bits of zero to be placed in the destination field.

X10X100 causes 10 characters to be transferred to the destination field with 90 blanks on the right.

An algorithm for converting 8-bit tapes with the high bit on is:

\[ n(B1B0,B0B1,B7B7) \]

B1B0 skips one bit of the input record, B0B1 skips one bit of the output record, and B7B7 copies seven bits from the input record to the output record.

CONVERSION OF FLOATING POINT AND INTEGER DATA

Conversions are possible between the valid formats listed in table 2-1 within the restrictions for each conversion noted in table 2-4, such as conversion between the IBM floating point formats of 32, 64, and 128 bits and the CDC floating and double-precision floating formats of 60 and 120 bits. Conversions to single precision floating point are rounded to 48-bit precision; conversions to double-precision are rounded to 96-bit precision.

Conversion from the internal record to an external IBM floating-point format yields a minimum precision of 21 bits for floating point, 53 bits for long floating point, and 109 bits for extended-precision floating point.

CONVERSION OF BINARY DATA

Any data may be considered binary and manipulated on a bit-by-bit basis. Bits may be copied in strings, or selectively, by skipping bits or replacing bit groups in a string with zeros.

Bit strings may be converted to any other valid format within the limitations expressed in table 2-4.

Item locators in selector expressions may address any bit in a character or bit string.

For example:

\[ 2/5B1 \] will reference the fifth bit of the second byte in the source record.

\[ (2/5B1 \text{ EQ } \$1\text{S}:X1C0,X1C1) \] will translate the second character to the internal record only if its fifth bit is 1.

When the internal record is referenced using data type B, all references must be based on 6-bit bytes, although the 12-bit internal format is used to contain EBCDIC or ASCII characters. To refer to the fifth bit of the fifth character of an EBCDIC 12-bit internal record, the item locator must be written \[ 9/5B1 \] because all references by the item locator to internal format are limited to 6-bit bytes. This inconvenience reduces the number of problems that otherwise would be encountered in other comparisons.
Table 2-1. Valid T and m Values

<table>
<thead>
<tr>
<th>T</th>
<th>Description</th>
<th>m*</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>Bits</td>
<td>size of field in bits</td>
</tr>
<tr>
<td>X</td>
<td>8-bit characters</td>
<td>size of field in (8-bit) characters</td>
</tr>
<tr>
<td>H</td>
<td>Half-word (16-bit) integer</td>
<td>--</td>
</tr>
<tr>
<td>W</td>
<td>Whole-word (32-bit) integer</td>
<td>--</td>
</tr>
<tr>
<td>G</td>
<td>Double-word (64-bit) integer</td>
<td>--</td>
</tr>
<tr>
<td>F</td>
<td>Floating point (32-bit)</td>
<td>--</td>
</tr>
<tr>
<td>L</td>
<td>Long floating point (64-bit)</td>
<td>--</td>
</tr>
<tr>
<td>E</td>
<td>Extended-precision floating point (128-bit)</td>
<td>--</td>
</tr>
<tr>
<td>P</td>
<td>Packed decimal (IBM COMP-3 COBOL items)</td>
<td>size of field in (8-bit) bytes</td>
</tr>
<tr>
<td>S</td>
<td>Decimal signed numeric</td>
<td>size of field in (8-bit) bytes</td>
</tr>
</tbody>
</table>

Internal Media: (Internal CDC format)

| B     | Bits                                             | size of field in bits         |
| X     | 6-bit characters (Display Code)                  | size of field in (6-bit) characters |
| A     | 12-bit characters (EBCDIC)                       | size of field in (12-bit) characters |
| C     | 12-bit characters (EBCDIC)                       | size of field in (12-bit) characters |
| I     | Integer (60-bit)                                 | --                            |
| U     | Unnormalized floating point (60-bit)             | --                            |
| E     | Normalized floating point (60-bit)               | --                            |
| D     | Double precision floating point (120-bit)        | --                            |
| S     | Numeric, signed overpunch (Display Code)         | size of field in (6-bit) characters |
| N     | Numeric, unsigned (Display Code)                 | size of field in (6-bit) characters |
| Z     | Numeric, leading zeros suppressed (Display Code) | size of field in (6-bit) characters |

Print Media: (Print format)

| B     | Bits                                             | size of field in bits         |
| X     | 12-bit characters (ASCII)                        | size of field in (12-bit) characters |

Card Media: (Card file format)

| B     | Bits                                             | size of field in bits         |
| X     | 12-bit characters (Hollerith)                    | size of field in (12-bit) characters |

Control:

| Q     | Terminate conversion (quit)                      | --                            |

*When m is --, no m specification may be made.
<table>
<thead>
<tr>
<th>Tm₁</th>
<th>Default Tm₂</th>
<th>Tm₁</th>
<th>Default Tm₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tape:</td>
<td>Internal:</td>
<td>Internal:</td>
<td>Tape:</td>
</tr>
<tr>
<td>Bm</td>
<td>Bm</td>
<td>Bm</td>
<td>Bm</td>
</tr>
<tr>
<td>Xm</td>
<td>Xm</td>
<td>Xm</td>
<td>Xm</td>
</tr>
<tr>
<td>H</td>
<td>I</td>
<td>Am</td>
<td>Xm</td>
</tr>
<tr>
<td>W</td>
<td>I</td>
<td>Cm</td>
<td>Xm</td>
</tr>
<tr>
<td>G</td>
<td>I</td>
<td>I</td>
<td>W</td>
</tr>
<tr>
<td>F</td>
<td>E</td>
<td>U</td>
<td>F</td>
</tr>
<tr>
<td>L</td>
<td>E</td>
<td>E</td>
<td>F</td>
</tr>
<tr>
<td>E</td>
<td>D</td>
<td>D</td>
<td>L</td>
</tr>
<tr>
<td>Pm</td>
<td>Sn (n=2m)</td>
<td>Sm</td>
<td>Sm</td>
</tr>
<tr>
<td>Sm</td>
<td>Sm</td>
<td>Nm</td>
<td>Sm</td>
</tr>
<tr>
<td>Zm</td>
<td>Xm</td>
<td>Xm</td>
<td>Xm</td>
</tr>
</tbody>
</table>

Card: Internal:

| Bm | Bm |
| Xm | Xm |

(Refer to table 2-4 for conversion rules.)
Table 2-3. Comparison Modes

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>B,X</td>
<td>S</td>
<td>N</td>
<td>S</td>
<td>N</td>
</tr>
<tr>
<td>H,W,G,F, L,E,P,S</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

**Internal:**

<table>
<thead>
<tr>
<th></th>
<th>String Literal</th>
<th>Numeric Literal</th>
<th>B,X,A,C</th>
<th>I,U,E,D, S,N,Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>B,X,A,C</td>
<td>S</td>
<td>N</td>
<td>S</td>
<td>N</td>
</tr>
<tr>
<td>I,U,E,D, S,N,Z</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

**Card:**

<table>
<thead>
<tr>
<th></th>
<th>String Literal</th>
<th>Numeric Literal</th>
<th>B,X</th>
</tr>
</thead>
<tbody>
<tr>
<td>B,X</td>
<td>S</td>
<td>N</td>
<td>S</td>
</tr>
</tbody>
</table>

*S = string literal mode
N = numeric mode

To use this table:

1. Select the subtable corresponding to the source-record to be converted. For reading tape, use Tape; for reading cards, use Card; for writing, use Internal.

2. Find the row in the subtable corresponding to the data type of item-locator-1.

3. Find the column in the subtable corresponding to the data type of item-locator-2 (or string literal type).

4. The intersection gives the comparison mode.

Example:  
\[ ... .X3 \text{ EQ } $123$ \]  
\[ ... .X3 \text{ EQ } 123 \]  
is a string literal mode comparison;  
is numeric mode.
<table>
<thead>
<tr>
<th>Ext.</th>
<th>Int.</th>
<th>B</th>
<th>X</th>
<th>A</th>
<th>C</th>
<th>I</th>
<th>U</th>
<th>E</th>
<th>D</th>
<th>S</th>
<th>N</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td></td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3,4</td>
<td>3,5</td>
<td>3,6</td>
<td>3,7</td>
<td>3,8</td>
<td>3,9</td>
<td>3,10</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>11</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>13,4</td>
<td>13,5</td>
<td>13,6</td>
<td>13,7</td>
<td>13,8</td>
<td>13,9</td>
<td>13,10</td>
</tr>
<tr>
<td>H</td>
<td></td>
<td>14</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>W</td>
<td></td>
<td>14</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>G</td>
<td></td>
<td>14</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>F</td>
<td></td>
<td>14</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>L</td>
<td></td>
<td>14</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>E</td>
<td></td>
<td>14</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>S</td>
<td></td>
<td>14,26</td>
<td>15,26</td>
<td>15,26</td>
<td>15,26</td>
<td>4,26</td>
<td>5,26</td>
<td>6,26</td>
<td>7,26</td>
<td>8,26</td>
<td>9,26</td>
<td>10,26</td>
</tr>
</tbody>
</table>

**Table 2-4. Conversion Rules**
(Numbers refer to notes following table)

<table>
<thead>
<tr>
<th>Ext.</th>
<th>Int.</th>
<th>B</th>
<th>X</th>
<th>H</th>
<th>W</th>
<th>G</th>
<th>F</th>
<th>L</th>
<th>E</th>
<th>P</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td></td>
<td>1</td>
<td>2</td>
<td>3,16</td>
<td>3,17</td>
<td>3,18</td>
<td>3,19, 20</td>
<td>3,19, 21</td>
<td>3,19, 22</td>
<td>3,23, 25</td>
<td>3,24, 26</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>11</td>
<td>12</td>
<td>13,16</td>
<td>13,17</td>
<td>13,18</td>
<td>13,19, 20</td>
<td>13,19, 21</td>
<td>13,19, 22</td>
<td>13,23, 25</td>
<td>13,24, 26</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td>11</td>
<td>12</td>
<td>13,16</td>
<td>13,17</td>
<td>13,18</td>
<td>13,19, 20</td>
<td>13,19, 21</td>
<td>13,19, 22</td>
<td>13,23, 25</td>
<td>13,24, 26</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>11</td>
<td>12</td>
<td>13,16</td>
<td>13,17</td>
<td>13,18</td>
<td>13,19, 20</td>
<td>13,19, 21</td>
<td>13,19, 22</td>
<td>13,23, 25</td>
<td>13,24, 26</td>
</tr>
<tr>
<td>I</td>
<td></td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19,20</td>
<td>19,21</td>
<td>19,22</td>
<td>23,25</td>
<td>24,26</td>
</tr>
<tr>
<td>U</td>
<td></td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19,20</td>
<td>19,21</td>
<td>19,22</td>
<td>23,25</td>
<td>24,26</td>
</tr>
<tr>
<td>E</td>
<td></td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19,20</td>
<td>19,21</td>
<td>19,22</td>
<td>23,25</td>
<td>24,26</td>
</tr>
<tr>
<td>D</td>
<td></td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19,20</td>
<td>19,21</td>
<td>19,22</td>
<td>23,25</td>
<td>24,26</td>
</tr>
<tr>
<td>S</td>
<td></td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19,20</td>
<td>19,21</td>
<td>19,22</td>
<td>23,25</td>
<td>24,26</td>
</tr>
<tr>
<td>N</td>
<td></td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19,20</td>
<td>19,21</td>
<td>19,22</td>
<td>23,25</td>
<td>24,26</td>
</tr>
<tr>
<td>Z</td>
<td></td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19,20</td>
<td>19,21</td>
<td>19,22</td>
<td>23,25</td>
<td>24,26</td>
</tr>
</tbody>
</table>
Table 2-4. Conversion Rules (continued)

NOTES: (→ implies conversion to)

1. Applies to Bm→Bn. Copy the leftmost \([\min(m, n)]\) bits from the source field to the destination field. If \(m > n\), the rightmost \((m-n)\) bits of the source field are ignored. If \(m < n\), the rightmost \((n-m)\) bits of the destination field are set to zero.

BmB0 may be used to skip \(m\) bits of the source record. B0Bn may be used to zero an \(n\)-bit field in the destination area.

2. Applies to Bm→Xn, An, Cn. Copy the source field to the destination field one bit at a time from the left. Convert each zero bit to the character 0, and each one bit to the character 1. If \(m > n\), the rightmost \((m-n)\) bits of the source field are ignored. If \(m < n\), the rightmost \((n-m)\) characters of the destination field are set to 0.

B0Xn (or B0An or B0Cn) may be used to set an \(n\)-character field in the destination area to all 0’s.

3. Applies to Bm→Numeric Type. The source field is treated as an \(m\)-bit positive binary integer.

A B0 source field is treated as zero.

4. Applies to →I. The source field is rounded (if necessary) to an integer, and the low order 59 bits are taken as the value. If the magnitude is \(>2^{59}\) (i.e., more than 59 bits required), an ERROR condition is flagged.

5. Applies to →U. The source field is rounded (if necessary) to an integer, and the high order 48 bits are taken as the value. The result is kept as a single precision floating point number. However, this number is de-normalized (COBOL COMP-1 definition), if necessary, to keep the biased exponent \(\geq 2006_8\).

6. Applies to →E. The source field is rounded to 48-bit precision and the result is kept as a single precision floating point number.

7. Applies to →D. The source field is rounded to 96-bit precision and the result is kept as a double-precision floating point number.

8. Applies to →Sn. The source field is rounded (if necessary) to an integer. If the magnitude is \(>10^n\), high order truncation occurs. The value is converted to a Display Code string of decimal digits, with leading zeros if necessary. The sign of the number is indicated by amending the low order (units) digit as follows:

Positive: \[0 \rightarrow \leq \]
(\(0 \rightarrow \leq\) card punch)

1-9 → A-J
(corresponds to 12-1 → 12-9 card punch)

Negative: \[0 \rightarrow \wedge \]
(\(0 \rightarrow \wedge\) card punch)

1-9 → J-R
(corresponds to 11-1 → 11-9 card punch)
9. Applies to →Nn. The source field is rounded (if necessary) to an integer. If the magnitude is \( \geq 10^n \), an ERROR is flagged. The value is converted to a Display Code string of decimal digits, with leading zeros if necessary. The sign of the field is lost (magnitude only saved).

10. Applies to →Zn. The source field is rounded (if necessary) to an integer. If the magnitude is \( \geq 10^n \), an ERROR is flagged. The value is converted to a Display Code string of decimal digits. Leading zeros are suppressed and replaced by blanks. If the number is negative, a - replaces the rightmost blank. If the number is negative and no blanks are in the field, an ERROR is flagged.

11. Applies to Xm→Bn. The source field is copied to the destination field one character at a time from the left. Each 0 is converted to a single zero bit, and each 1 is converted to a single one bit. If any character besides 0 or 1 is encountered, an ERROR is flagged. If \( m > n \), the rightmost \((m-n)\) characters of the source are ignored. If \( m < n \), the rightmost \((n-m)\) bits of the destination are set to zero.

12. Applies to (string)m→(string)n. The source field is copied to the destination field from the left, with conversion according to appendices A, B, and C if necessary. If the source field corresponds to card (Hollerith) input, and if the card punches for any character position are invalid, the eight-ones character (hexadecimal FF) is used for that position. If \( m > n \), the rightmost \((m-n)\) characters of the source string are ignored. If \( m < n \), the rightmost \((n-m)\) characters of the destination are set to blanks.

Using an \( n \) value of 0, \( m \) characters of the source may be skipped. Using an \( m \) value of 0 allows setting a destination field to all spaces.

13. Applies to (string)→Numeric Type. A source character string which is to be converted to numeric type must have the following general form:

\[
\begin{bmatrix}
\text{+} \\
- \\
\end{bmatrix}
digit-1[digit-2] \ldots \left[ \text{[digit-3]} \ldots \text{[E} \begin{bmatrix}
\text{+} \\
- \\
\end{bmatrix}
digit-4[digit-5] \ldots \right]
\]

where digit is 0 through 9. This format closely follows that used by FORTRAN, a decimal value optionally followed by a power-of-ten exponent. Numbers are kept to a precision of at least 96 bits. Spaces are ignored; they may be embedded anywhere within the field. If any other character appears in the field, or if the syntactic form is incorrect, an ERROR is flagged. If the source field width is zero, the value is taken to be zero. If \( E \) is present, a decimal point also must be present.

14. Applies to Numeric→Bn. If necessary, the source field is converted to binary and rounded to integer form. The rightmost \( n \) bits, with sign extended, are moved to the destination field. The binary representation is in the form appropriate to the destination: 2's complement for 360 format and 1's complement for CDC CYBER 70 format. If \( n \) bits are insufficient to contain the result, the rightmost \( n \) bits are placed in the destination field. If the source field width is zero, or the value is infinite or indefinite, the value is taken to be -0 (+0 if the destination is 360 format).

15. Applies to Numeric→(string)n. The conversion of numeric fields to alphanumeric (string) fields depends upon several factors, including the size of the destination field, magnitude and sign of the source field, and maximum precision of the source item. The sub-rules below constitute an algorithm by which the receiving field format may be determined: (Alphabetic designation implies: A=all, I=integer, F=float point F type, E=E type, R and X are special cases.)
A.1 If the destination field width is zero, no conversion takes place.

A.2 If the source item is indefinite or infinite (internal items only) go to step X.1.

A.3 Determine the maximum precision, p, of the source item from the following table:

<table>
<thead>
<tr>
<th>source item type</th>
<th>p (digits)</th>
<th>number of guaranteed accurate decimal digits</th>
</tr>
</thead>
<tbody>
<tr>
<td>(360)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>W</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>G</td>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td>F</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>L</td>
<td>17</td>
<td>16</td>
</tr>
<tr>
<td>E</td>
<td>34</td>
<td>33</td>
</tr>
<tr>
<td>Pm</td>
<td>2m-1</td>
<td>2m-1</td>
</tr>
<tr>
<td>Sm</td>
<td>m</td>
<td>m</td>
</tr>
<tr>
<td>(CYBER)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>18</td>
<td>17</td>
</tr>
<tr>
<td>U</td>
<td>15</td>
<td>14</td>
</tr>
<tr>
<td>E</td>
<td>15</td>
<td>14</td>
</tr>
<tr>
<td>D</td>
<td>29</td>
<td>28</td>
</tr>
<tr>
<td>Sm</td>
<td>m</td>
<td>m</td>
</tr>
<tr>
<td>Nm</td>
<td>m</td>
<td>m</td>
</tr>
<tr>
<td>Zm</td>
<td>m</td>
<td>m</td>
</tr>
</tbody>
</table>

A.4 Set a variable, d, to n. If the source value is negative, set d to n-1. d is the available destination field width.

A.5 If the source value is an integer (units bit represented and fractional part equal to zero) go to step I.1, otherwise, go to step F.1.

I.1 If the magnitude of the value is $\geq 10^d$, go to step E.1.

I.2 Convert the value to a decimal integer and place it in the destination field, right adjusted. Replace all leading zeros (except in the units position) by spaces. If the value is positive, procedure is DONE.

I.3 If the value is negative and n>1, place a - immediately to the left of the leftmost digit. Otherwise (must be -0 in a one character field) replace the 0 by a - DONE.

F.1 Determine r, the minimum number of digit positions required to use this representation as follows:

If $|\text{value}| \geq 1$, then $r=K$, where $10^{K-1} \leq |\text{value}| < 10^K$

If $|\text{value}| < 1$, then $r=K-1+\min(p,d-5)$, where $10^{-K} \leq |\text{value}| < 10^{-K+1}$
F.2 If \( r \leq (d-1) \), proceed to step F.3, otherwise go to step E.1.

F.3 Convert the value according to one of the following formats and place the result string, right adjusted, in the destination field. Round the value, if necessary, to the indicated number of places.

(a) If \([x \geq 1) \) and \([r = (d-1) \) or \((r \geq p)\):\]

\[
d1d2 \ldots \text{di.} \quad i = r
\]

(b) If \((x \geq 1) \) and \([r < (d-1) \) and \((r < p)\):\]

\[
d1d2 \ldots \text{di.d1+1} \ldots \text{dj} \quad i = r, \quad j = \min(p,d-1)
\]

(c) If \((x < 1) \) and \([d-1) > (k-1+p)\):\]

\[
0.d1d2 \ldots \text{dj} \quad j = K-1+p
\]

(d) If \((x < 1) \) and \([d-1) \leq (k-1+p)\):\]

\[
.\text{d1d2} \ldots \text{dj} \quad j = d-1
\]

F.4 If the value is negative, place a - immediately to the left of the leftmost non-blank character.

DONE.

E.1 If \( d < 6 \), or if \( d = 6 \) and \( |\text{value}| < 0.95 \times 10^{-99} \), go to step R.1, otherwise proceed to step E.2.

E.2 Convert the value according to the following format:

\[
d1.d2.\ldots\text{dj}E\text{eee} \quad j = \min(p,d-5)
\]

where eee is -99 to -01, +00 to +99, or 100 to 305.

If a negative exponent less than -99 is required, the following format is used:

\[
d1.d2\ldots\text{dj}E-n\text{nnn} \quad j = \min(p,d-6)
\]

Similar to FORTRAN scientific notation. The value is rounded to the indicated number of digits and placed, right adjusted, in the destination field.

E.3 If the value is negative, place a - immediately to the left of the leftmost non-blank character.

DONE.

R.1 This step is reached when the receiving field is not wide enough to represent the value, the destination field is filled with all asterisks. If the value is negative, the leftmost * is replaced by a -. DONE.
X.1 This step is reached when the source field is either infinite or indefinite (internal source items only). The destination field is filled as shown below:

<table>
<thead>
<tr>
<th>Condition</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4 or more (right adjusted in field)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+∞</td>
<td>F</td>
<td>NF</td>
<td>INF</td>
<td>INF</td>
</tr>
<tr>
<td>-∞</td>
<td>F</td>
<td>-F</td>
<td>-NF</td>
<td>-INF</td>
</tr>
<tr>
<td>+?</td>
<td>D</td>
<td>ND</td>
<td>IND</td>
<td>IND</td>
</tr>
<tr>
<td>-?</td>
<td>D</td>
<td>-D</td>
<td>-ND</td>
<td>-IND</td>
</tr>
</tbody>
</table>

16. Applies to $\rightarrow$H. The source value is rounded, if necessary, to a 2's complement integer, and the low-order 16 bits are taken as the value. If significance is lost, an ERROR is flagged.

17. Applies to $\rightarrow$W. The source value is rounded, if necessary, to a 2's complement integer, and the low-order 32 bits are taken as the value. If significance is lost, an ERROR is flagged.

18. Applies to $\rightarrow$G. The source value is rounded, if necessary, to a 2's complement integer, and the low-order 64 bits are taken as the value. If significance is lost, an ERROR is flagged.

19. Applies to $\rightarrow$F,L,E. The source value is converted to a 360 format floating point number, rounded to the indicated number of bits. If the source magnitude is too large ($> \sim 5 \times 10^{75}$), the largest possible number is supplied. If the source magnitude is too small ($< \sim 5 \times 10^{-75}$) but not zero, the smallest non-zero number is supplied.

20. Applies to $\rightarrow$F. The resultant value is a 4-byte field of 21-24 bit precision (360 short floating point).

21. Applies to $\rightarrow$L. The resultant value is an 8-byte field of 53-56 bit precision (360 long floating point).

22. Applies to $\rightarrow$E. The resultant value is a 16-byte field of 110-112 bit precision (360/370 extended precision floating point). The low-order 14-16 bits may not be accurate, since only 96-bit precision is guaranteed.

23. Applies to $\rightarrow$Pm. The source value is rounded, if necessary, to an integer and converted to packed decimal form. If $|\text{value}| \geq 10^{2m-1}$, overflow has occurred; and an ERROR is flagged.

24. Applies to $\rightarrow$Sm. The source value is rounded, if necessary, to an integer and converted to a decimal string. If $|\text{value}| \geq 10^m$, overflow has occurred and an ERROR is flagged.
25. For P fields, the sign of the field and low-order (units) numeric place are contained in the low-order (rightmost) byte as follows:

<table>
<thead>
<tr>
<th>Bits</th>
<th>0</th>
<th>3</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Units Digit</td>
<td>Sign</td>
<td></td>
</tr>
</tbody>
</table>

Valid sign codes are:

- bit pattern: 1010  sign: +
- 1011  -
- 1100  +  Preferred code used when a P field is generated
- 1101  -
- 1110  +
- 1111  +

26. For S fields in EBCDIC data, the sign of the field and low-order (units) numeric places are contained in the low-order (rightmost) byte as follows:

<table>
<thead>
<tr>
<th>Low Order Digit</th>
<th>Character Placed in that Position</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+ (sign) -</td>
</tr>
</tbody>
</table>

| 0    | (    ) |
| 1    | A     |
| 2    | B     |
| 3    | C     |
| 4    | D     |
| 5    | E     |
| 6    | F     |
| 7    | G     |
| 8    | H     |
| 9    | I     |

These replacements correspond to numeric + overpunch representation in card decks (digit+12 zone for plus, digit+11 zone for minus). In a source field, either the signed or unsigned form may be used in the sign position.

For S fields in ASCII data, the sign of the field and low-order (units) numeric digit are contained in the low-order byte as follows:

<table>
<thead>
<tr>
<th>Low Order Digit</th>
<th>Character Placed in that Position</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>+ (sign) -</td>
</tr>
</tbody>
</table>

| 0  | @   | P  |
| 1  | A   | Q  |
| 2  | B   | R  |
| 3  | C   | S  |
| 4  | D   | T  |
| 5  | E   | U  |
| 6  | F   | V  |
| 7  | G   | W  |
| 8  | H   | X  |
| 9  | I   | Y  |
The subroutines XFILE, XREAD, XREREAD, and XWRITE described in this section are used to read or write 9-track or 7-track IBM sequential tape files, to read or punch cards, or to print extended character-set data. All input and output goes through the Record Manager, but records are neither converted nor blocked/deblocked at that level. Accordingly, special conventions are necessary for Record Manager, KRONOS and SCOPE declarations; these are discussed in section 6, System Interface.

INPUT/OUTPUT PARAMETERS

For the calling sequences to the input/output subroutines, the following general information applies to the parameters.

File Name (file)

In a COBOL calling sequence, this parameter must be a file name. In a FORTRAN calling sequence, it may be either an integer file number or a literal constant written nL and containing the file name. In a COMPASS calling sequence, the parameter may be either an nL literal constant or the address of a word containing the file name in Display Code, left adjusted and zero-filled.

Integer Parameter (size)

In a COBOL calling sequence to an I/O subroutine, the size parameter must be an integer COMPUTATIONAL-1 item, containing no more than 14 decimal digits. In FORTRAN, the parameter may be an integer constant, variable, or expression. In COMPASS, the size parameter may be an integer constant or the address of a word containing an integer value.

Real Parameter (status)

The status parameter in a COBOL calling sequence must be a COMPUTATIONAL-2 item. In FORTRAN, it must be a real variable name. In COMPASS, it must be the location symbol of a word that contains a real (floating-point) value.

String Parameters (file-string, conversion-string)

String parameters are used to pass variable information and are elementary 01 level or SYNC LEFT display data items. In a COBOL calling sequence, these parameters are DISPLAY data items with VALUE set to the desired string elements, or they are alphanumeric literal strings. In FORTRAN, they can be variable or array names or constants written in nH or nL form, or set off by "' or # quotation symbols. In COMPASS, the string parameters may be written as literal constants or as location symbols of words which contain the string items written as character data.
FILE DEFINITION

Before the first reference to a file by XREAD or XWRITE, the file must be defined to the input/output procedures by a calling sequence to the XFILE routine. No operation is performed on the file by XFILE; however, workspace and other specific information is defined for the file.

The workspace provided by XFILE for the file may not be used simultaneously for any other purpose. It may be freed at the user's discretion for other use, but no more references may be made to the defined file unless another XFILE calling sequence is executed.

FORTRAN Calling Sequence:

CALL XFILE(file,workspace,file-string,size)

file
File identification; integer constant, simple integer variable, or a literal written nL and containing the file name. The file name must have been declared in the PROGRAM statement.

workspace
Name of working area to be used by input/output procedures. It is an array name that is dimensioned or equivalenced to satisfy workspace size requirements.

file-string
Parenthesized list that specifies required file information in keyword form. In FORTRAN either ≠ or " marks must delineate the file-string which is written as a variable or array name or as a string constant in H, L, or "..." format.

size
Size of workspace area to be used, expressed as a decimal integer constant or variable which describes a number of central memory words.

COBOL Calling Sequence:

ENTER XFILE USING file,workspace,file-string,size.

file
Identification of file (file name) to be associated with workspace area.

workspace
Name of data working area to be used by input/output procedures. It is a data item name and must begin on a word boundary (synchronized or 01 level item).

file-string
Parenthesized list of parameters specifying certain file information in keyword form. In COBOL, the parenthesized list must be delineated by ≠ marks and may be written as a data name or alphanumeric literal constant.

size
Size of workspace area in number of central memory words, expressed as a COMPUTATIONAL-1 item. This parameter is optional; the COBOL default value for size is the length of the workspace data item in words.
COMPASS Calling Sequence:

<table>
<thead>
<tr>
<th>Location</th>
<th>Operation</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA1</td>
<td></td>
<td>plist</td>
</tr>
<tr>
<td>RJ</td>
<td></td>
<td>XFILE</td>
</tr>
<tr>
<td></td>
<td>plist</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VFD</td>
<td>60/file</td>
</tr>
<tr>
<td></td>
<td>VFD</td>
<td>60/workspace</td>
</tr>
<tr>
<td></td>
<td>VFD</td>
<td>60/file-string</td>
</tr>
<tr>
<td></td>
<td>VFD</td>
<td>60/size</td>
</tr>
<tr>
<td></td>
<td>BSSZ</td>
<td>1</td>
</tr>
</tbody>
</table>

plist Symbolic location of parameter list which must be terminated with word of binary zeros.

file Symbolic location of word containing identification of file to be associated with workspace area.

workspace Symbolic location of working area to be used by input/output procedures. The size of the area must be sufficient to meet the requirements of the character strings to be processed.

file-string Symbolic location of a parenthesized literal written as a string in H, L or "..." format, specifying required file information in keyword form.

size Symbolic location of integer value specifying size of the workspace area in number of central memory words.

FILE-STRING PARAMETERS

File-strings are written in the XFILE calling sequence as a series of keywords and parameters, enclosed in parentheses. Three keywords with parameters are required for tape and printer files; only two are required for card files. The order in which the keywords and optional parameters are written is of no consequence.

For tape and print files, the file-string is written:

\[(\text{FT}=\text{ft}, \text{USE}=\text{use}, \text{RECFM}=\text{form}, \text{optional parameters})\]

In card files, it is written:

\[(\text{FT}=\text{ft}, \text{USE}=\text{use}, \text{optional parameters})\]

\(\text{ft}\) File type keyword

- T IBM 360/370 format tape
- P Print file with extended character set print train
- C Card file (read or punch)
use File usage keyword

R Read (input) file
W Write (output) file

For FT=P, file usage is USE=W only.

form Record format keyword (omitted for card files)

F
FB
FS
FSB
V
VB
VS
VSB
U
US
UB
USB

Required tape file parameter. Defines IBM record/block format used as explained under IBM File Record and Block Formats in appendix B.

These formats are extensions of IBM file record/block types. See appendix B.

Optional file-string parameters are:

Block Size: Required for tape files; gives maximum size of tape block in 8-bit characters.
BLKSIZE = nnnn

Record Size: Required for blocked tape files and VS format (FB, FSB, VS, VB, VSB, UB, USB); gives maximum size of logical record in 8-bit characters.
LRECL = nnnn

Data Code: Optional for tape files; defines external code as ASCII (A) or EBCDIC (C). If omitted, EBCDIC is assumed.
CODE = A
C

Print Format: Optional parameter for print files; 1, 2, and 3 indicate single, double, or triple space. If A, record character 1 is used as carriage control code. If parameter is omitted, default is FMT=A.
FMT = 1
2
3
A

RECFM, BLKSIZE, and LRECL have the same meaning as their 360 (Job Control Language) counterparts, and generally, they should be copied from the associated DD card. Appendix B includes a description of IBM 360/370 tape formats.
WORKSPACE AREA SIZE

Size of the workspace area in number of central memory words is determined by the file type parameter in the file-string, as follows:

FT=C 20 words are required
FT=P 32 words are required
FT=T  Workspace is determined by the record format parameter in the file-string, as follows:

RECFM = F,FS,FB,FSB,U,US,V and reading of VB,UB,USB

\[
\text{space} = 6 + \left\lfloor \frac{\text{BLKSIZE}}{7.5} \right\rfloor \text{ words}
\]

RECFM = VS,VSB and writing of VB,UB,USB

\[
\text{space} = 6 + \left\lfloor \frac{\text{LRECL}}{7.5} \right\rfloor + \left\lfloor \frac{\text{BLKSIZE}}{7.5} \right\rfloor \text{ words}
\]

where \( \left\lfloor x \right\rfloor \) means the smallest integer not less than \( x \).

FILE USAGE

Files written or read by the 8-bit routines may not be processed subsequently by normal read/write functions in the same FORTRAN or COBOL program. The 8-bit routines alter the Record Manager file information table (FIT), and the original information contained in the FIT for the files is not viable for normal FORTRAN or COBOL input/output.

XWRITE WRITE A RECORD

This routine takes data from a record area in memory, converts it, if specified in the calling sequence, and writes it in the file workspace area. When enough data is collected in the workspace area to form a record, XWRITE sends the record to Record Manager which outputs the record. When used with COPY8P (extended print codes) the trailing zero byte is inserted during conversion.

If the file is not opened when XWRITE is ready to send a record to the input/output manager, XWRITE will open it. A file can be opened by an OPEN request in COMPASS, or by a previous XWRITE calling sequence. XWRITE will specify open for I/O with no file positioning.

FORTRAN Calling Sequence:

\[
\text{[CALL] XWRITE} \left( \text{workspace,source \left\{ \begin{array}{l}
\text{}\text{,conversion-string} \\
\text{}\text{,conversion-string,status}
\end{array} \right\} \right)
\]

workspace Name of working area to be used by XWRITE, as defined for the file name given in the XFILE call. The XWRITE call is linked with a specific file only by the workspace name.
source

Name of array containing source record.

conversion-string

Conversion string to be used. This parameter is parenthesized, and is described in section 2, Data Conversion Strings. It may be written as a variable, an array name, or a string constant. If parameter is omitted, record size is not converted before it is written to the workspace. Record size default values are 80 characters for card files, 137 characters for print files (136 if a format is specified), and LRECL for tape files.

status

Optional real variable name to which a status value is returned as follows:

0.0 No error, no abnormal condition
1.0 Error during conversion
2.0 Non-recoverable error on output file

If XWRITE routine is called as a FORTRAN function, the status value is returned as the value of the function. For example:

\[
\text{KEY}=\text{XWRITE}(A,B,\neq(X10X10)\neq)
\]

In this case, status will be returned as a real value for the value of the function, which is then converted to an integer to replace the value of KEY.

If conversion-string parameter is omitted from the calling sequence, then the status parameter must be omitted also.

COBOL Calling Sequence:

```
ENTER XWRITE USING workspace,source
\{ \{ ,conversion-string
\{ ,conversion-string,status \} \} \}.
```

workspace

Name of working storage area to be used by XWRITE, as defined for the file name given in the XFILE calling sequence. The XWRITE call is linked with a specific file only by the workspace name.

source

Data item name containing source record.

conversion-string

Conversion specification to be used. This parameter is parenthesized, as described in section 2, Data Conversion Strings. It may be a data name or an alphanumeric literal string surrounded by quotation marks. If this parameter is omitted, the record is not converted before it is written into the workspace. Item size is passed in the parameter list; the shorter of item size or record size determines the record length.

status

Optional COMPUTATIONAL-2 (real) item to which a status value is returned.

0.0 No error or abnormal condition
1.0 Error during conversion
2.0 Non-recoverable error on output file

If the conversion-string parameter is omitted from the calling sequence, then the status parameter must be omitted also.
COMPASS Calling Sequence:

<table>
<thead>
<tr>
<th>Location</th>
<th>Operation</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA1</td>
<td>plist</td>
<td></td>
</tr>
<tr>
<td>RJ</td>
<td>XWRITE</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>plist</td>
<td>VFD</td>
<td>60/workspace, 60/source { {60/conversion-string }, {60/conversion-string, 60/status } } , 60/0</td>
</tr>
</tbody>
</table>

plist | Symbolic location of parameter list which must be terminated with a word of binary zeros.

workspace | Name of working area to be used by XWRITE as defined in the XFILE call. The XWRITE call is linked with a specific file only by the workspace name.

source | Name of area to contain the source record.

collection-string | Symbolic location of parenthesized literal specifying the conversion option. If parameter omitted, the record is not converted before it is written in the workspace area.

status | Optional location in which the status value is returned after the write operation, as follows:

0.0 No error; no abnormal condition
1.0 Error during conversion
2.0 Non-recoverable error on output file

If conversion-string parameter is omitted, the status parameter must be omitted also.

FORCING TERMINATION OF FILE OUTPUT

The XWRITE routine may be used to force termination of output to a file, and cause the writing of any partially filled workspace area. When blocked tape formats (FB, FSB, VB, VSB, UB, USB) are used, such a call to XWRITE should be issued after all other XWRITE calls, to ensure that all data is output. This call does NOT close the file.

FORTRAN Calling Sequence:

    CALL XWRITE(workspace)

COBOL Calling Sequence:

    ENTER XWRITE USING workspace.
COMPASS Calling Sequence:

<table>
<thead>
<tr>
<th>Location</th>
<th>Operation</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA1</td>
<td>plst</td>
<td></td>
</tr>
<tr>
<td>RJ</td>
<td></td>
<td>XWRITE</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>plst</td>
<td></td>
<td>42/0,18/workspace</td>
</tr>
<tr>
<td>VFD</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>BSSZ</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

XREAD/XREREAD READ A RECORD

These routines can be used to read a next record from the file workspace area, convert it, if specified in the calling sequence, and place it in a destination area in memory; to reread the current record, possibly with a different conversion specification; or to skip over records.

If the file is not open when XREAD is ready to process a record, XREAD will open it, specifying open for input/output with no rewind. A file cannot be opened by the XREREAD routine.

FORTRAN Calling Sequence:

To read a next record:

\[
\text{[CALL] \text{XREAD} (\text{workspace,destination} \ [\{,\text{conversion-string} \\
\quad \{,\text{conversion-string,status} \} \}])}
\]

To reread a current record:

\[
\text{[CALL] \text{XREREAD} (\text{workspace,destination} \ [\{,\text{conversion-string} \\
\quad \{,\text{conversion-string,status} \} \}])}
\]

workspace Name of working storage area for a file, as defined in the XFILE calling sequence. The XREAD/XREREAD call is linked to a specific file only by the workspace name.

destination Name of array to contain processed record.

conversion-string Conversion string to be used. If this parameter is omitted, the record is not converted as it is transferred to the destination area. This parameter is parenthesized, as described in section 2, Data Conversion Strings, and may be a variable, an array name, or a Hollerith constant.
status Optional real variable name to which a status value is returned:

-4.0 No data (XRREAD preceded any XREAD)
-3.0 End-of-information encountered
-2.0 End-of-partition encountered
-1.0 End-of-section encountered
0.0 No error or abnormal condition
1.0 Error during conversion
2.0 Non-recoverable error on input file

If either routine is used as a function, the status value is returned as the value of the function also.

COBOL Calling Sequence:

To read a next record:

ENTER XREAD USING workspace,destination [ { ,conversion-string } ] .

To reread a current record:

ENTER XREREAD USING workspace,destination [ { ,conversion-string, status } ] .

workspace Name of working storage area for a file, as defined in the XFILE calling sequence. The XREAD/XREREAD call is linked to a specific file only through the workspace name.

destination Data name of destination record area.

conversion-string Conversion string to be used. If this parameter is omitted, the record is transferred to the destination area without conversion. See section 2, Data Conversion Strings. This parameter may be a data name or a literal string surrounded with quotation marks.

status An optional COMPUTATIONAL-2 data name to which a status value is returned:

-4.0 No data (XRREAD preceded any XREAD)
-3.0 End-of-information encountered
-2.0 End-of-partition encountered
-1.0 End-of-section encountered
0.0 No error or abnormal condition
1.0 Error during conversion
2.0 Non-recoverable error on input file
COMPASS Calling Sequence:

To read a next record:

<table>
<thead>
<tr>
<th>Location</th>
<th>Operation</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA1</td>
<td>plist</td>
<td></td>
</tr>
<tr>
<td>RJ</td>
<td>XREAD</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>plist</td>
<td>VFD</td>
<td>60/workspace,60/destination</td>
</tr>
<tr>
<td></td>
<td>VFD</td>
<td>{ 60/conversion-string, status }</td>
</tr>
<tr>
<td></td>
<td>VFD</td>
<td>60/0</td>
</tr>
</tbody>
</table>

To reread a current record:

<table>
<thead>
<tr>
<th>Location</th>
<th>Operation</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA1</td>
<td>plist</td>
<td></td>
</tr>
<tr>
<td>RJ</td>
<td>XREREAD</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>plist</td>
<td>VFD</td>
<td>60/workspace,60/destination</td>
</tr>
<tr>
<td></td>
<td>VFD</td>
<td>{ 60/conversion-string, status }</td>
</tr>
<tr>
<td></td>
<td>VFD</td>
<td>60/0</td>
</tr>
</tbody>
</table>

plist  Symbolic location of parameter list which must be terminated with a word of binary zeros.

workspace Location of workspace area for a file. The read calls are linked to a specific file only by the workspace name.

destination Location of the area where the processed record is to be placed.

conversion-string Symbolic location of literal listing conversions to be made. If parameter is omitted, the record is transferred without conversion. The literal is parenthesized, as described in section 2, Data Conversion Strings.
status

Optional location of a status value indicating the result of the read as follows:

-4.0 No data (XREREAD preceded any XREAD)
-3.0 End of information encountered
-2.0 End of partition encountered
-1.0 End of section encountered
0.0 No error; no abnormal condition
1.0 Error during conversion
2.0 Non-recoverable error on input file

SKIPPING RECORDS

The XREAD sequence can be used to skip a record by causing the next record to be read from the workspace area but not converted or moved to a destination area. The subsequent record then is available as the current record and may be processed by XREREAD.

FORTRAN Calling Sequence:

CALL XREAD(workspace)

workspace Identifies the array area containing the file record to be skipped.

COBOL Calling Sequence:

ENTER XREAD USING workspace.

workspace Name of working storage area, as defined for a given file in the XFILE calling sequence.

COMPASS Calling Sequence:

<table>
<thead>
<tr>
<th>Location</th>
<th>Operation</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SA1</td>
<td>plist</td>
</tr>
<tr>
<td></td>
<td>RJ</td>
<td>XREAD</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>plist</td>
<td>VFD</td>
<td>42/0,18/workspace</td>
</tr>
<tr>
<td></td>
<td>BSSZ</td>
<td>1</td>
</tr>
</tbody>
</table>

plist Symbolic location of parameter list which must be terminated with a word of binary zeros.

workspace Symbolic location of file working storage area containing the record to be skipped.

Any calling sequence to XREREAD specifying only the workspace parameter and omitting all other parameters would be, effectively, a no-operation and no action is taken.
ERROR HANDLING

When errors are detected in processing input or output calling sequences, an error message (appendix G) is written to the job's OUTPUT file and dayfile. Error status codes are returned in the status location given in the XREAD, XREREAD, and XWRITE parameter list.

For a COBOL or FORTRAN Extended program, an attempt is made to provide traceback information.

No error is fatal to execution; however, all data conversion errors will terminate translation at the point of detection. For example, when a conversion error is found by XREAD, the remainder of the record will not be converted. In such a case, XREREAD may be used as an alternate procedure to reprocess the record with a different conversion string. When a conversion error is detected by XWRITE, the record will not be written. Input/output errors cause unpredictable results.

COBOL INPUT/OUTPUT EXAMPLES

For these examples, assume that a tape containing only 30-character EBCDIC coded records (no binary data) is to be read with a COBOL program. The characters must be converted to internal CDC format, as described in table 2-1. The conversion to be performed is specified by the conversion-string parameter in the XREAD/ XWRITE calling sequences.

The file type must be specified in the XFILE file-string parameter, along with usage, record format, and block size. These parameters should be copied from the associated DD card in 360 JCL. Data Division entries in the COBOL program might appear as:

    DATA DIVISION
    .
    .
    WORKING-STORAGE SECTION.
    77 SZ PIC 999 USAGE COMP-1 VALUE 46.
    77 STAT USAGE COMP-2 VALUE 0.
    01 CSTRING PIC X(60) VALUE X30C30.
    01 FSTRING PIC X(60) VALUE FT=R,RECFM=FB,BLKSZ=150,LRECL=30,CODE=C.
    01 TMP1 PIC X(30).
    01 WSL PIC X(460) VALUE SPACES.
    .

Any of the values in the working storage section can be altered at any time during execution; for example, the COMPUTATIONAL-2 item STAT is changed to reflect the status of read and write operations. In the FILE CONTROL paragraph of the Input-Output Section, the file name TT9 is used in the SELECT clause.
COBOL TAPE READING EXAMPLE

Before the file TT9 can be processed, the file must be opened and the XFILE routine called. These operations, together with the first read of the tape file might appear in the Procedure Division of the program as:

```
R1.
OPEN INPUT TT9.
ENTER XFILE USING TT9, WS1, FSTRING, SZ.
ENTER XREAD USING WS1, TMP1, CSTRING, STAT.
IF STAT NOT EQUAL 0 DISPLAY #FAIL R1#.
```

CONVERSION-STRING VARIATIONS FOR PRECEDING INPUT EXAMPLE

Conversions are performed when the format of the destination field is not the same as the format of the input field designation in the COBOL program. The X used to specify characters does not change, though the code designation of the tape may be changed. Thus, the same file on an ASCII format tape could be mounted, the tape code designation changed in the file-string parameters of the XFILE call, and the new tape processed with the same results.

To convert the information from an EBCDIC tape to ASCII or Display Code, specify T2 in the conversion item as A or X. When T2 is A, each 8-bit character from the tape is converted to an internal ASCII 12-bit character. The leftmost 4 bits within each 12-bit byte remain unused. When T2 is X, the 8-bit characters are converted to 6-bit Display Code characters according to the translation shown in appendix A.

- **(X10A10)** Converts ten 8-bit EBCDIC characters to ten 12-bit ASCII characters.
- **(X10X10)** Converts ten 8-bit EBCDIC characters to ten 6-bit Display Code characters where matching characters exist (see appendix A). Non-matching characters are converted to Display Code blanks.
- **(X30B30)** Converts EBCDIC characters to bits. This conversion is valid only for zero and one EBCDIC characters; otherwise, an error occurs.
- **(B240B240)** Converts bits to bits. Bits are moved to the destination field (TMP1) with no conversion. Source field bits that exceed the number of destination field bits as in B240B100, are skipped. In the reverse situation such as B240B300, remaining bits are set to zero.

Selector expressions may be used to execute conversion specifications conditionally based on a test made during execution. If the selector expression preceding the colon is true, the following conversion items are executed until a Q or semicolon is encountered. If the selector expression is not true, the next conversion specification following the semicolon is attempted.
(1X1 EQ *1*:9X2X1,X0X1;X1)

In the above example, if the first EBCDIC character (X1) is equal to the literal 1, the repeat count 9 is used to move 2 characters from the source field to a 1-character receiving field 9 times, effectively converting every other character to 6-bit Display Code, 9 times. X0X1 causes a Display Code blank to be appended. Thus, if the input record were:

12345678901234567890 in EBCDIC form (hexadecimal F1F2F3F4F5F6F7F8F9F0F1F2F3F4F5F6F7F8F9F0)
the resulting receiving field would contain 135791357 blank. Internally in Display Code, the octal equivalent of the characters would appear:

3436404244436404255

If the first character referenced on the tape were not a 1, only the conversion item X1 would be performed, and the first character would be converted to Display Code. The remainder of the destination field would remain untouched.

Bit manipulation differs from other formats in that a slash is required if other than the first bit within a byte is to be referenced.

(2/5B2 EQ $10S:X10,Q,X5)

Causes the fifth and sixth bits of the second character to be compared with the binary literal 10; if they match, 10 characters are converted to Display Code and the Q causes termination. (Were the Q not included, the semicolon would terminate the scope of the selector expression.)

If bits 5 and 6 are not equal to the binary literal 10, the selector expression is not true. The next conversion item is executed, and 5 characters are converted to Display Code and stored in the destination field. The conversion terminates.

The foregoing concepts may be extended to create the following conversion strings:

(3X3 EQ 6X3:10X2X1,Q;10X1X0,X1)

If the three characters starting with the third (characters 3, 4, 5) are equal to the three characters starting with the sixth (characters 6, 7, 8), convert every other 8-bit EBCDIC character to Display Code 10 times. (Where character translations do not exist, see appendix A.)

If the selector expression proves false, skip 10 characters and then convert one 8-bit EBCDIC character to Display Code.

(X3,(X3 EQ -3X3:X7))

Convert three 8-bit EBCDIC characters to Display Code and store in destination field, if the next three characters are the same as the preceding three (-3), convert the next seven 8-bit EBCDIC characters to Display Code also.

To resolve questions concerning defaults, refer to table 2-3. When uncertain as to the type of conversion or output, see table 2-4.
COBOL TAPE WRITING EXAMPLE

To write from an internal file to an EBCDIC-coded tape file, rules similar to those for reading apply. Internal data is in either 6-bit Display Code or 12-bit ASCII/EBCDIC format.

In this example, TT9 is the output file name. The record format of the output file is specified in the XFILE calling sequence. Before the first write may take place, the file must be opened and the XFILE routine called. This portion of the program might appear as:

```
  W1.
  OPEN OUTPUT TT9.
  ENTER XFILE USING TT9, WS1, FSTRING, SZ.
  ENTER XWRITE USING WS1, CS1, ≠(C30X30)≠, STAT.
  IF STAT NOT EQUAL 0 DISPLAY ≠FAIL W1≠.
```

CONVERSION STRING VARIATIONS FOR PRECEDING OUTPUT EXAMPLE

In the following examples, the source field descriptors will be A for 12-bit ASCII, C for 12-bit EBCDIC, X for 6-bit Display Code, or B for bits. X for the destination field descriptor represents 8-bit EBCDIC characters.

```
(C10X10) or (C10)
```

Would cause ten 12-bit EBCDIC characters to be written to tape as ten 8-bit EBCDIC characters.

To convert one character from internal 12-bit ASCII to 8-bit EBCDIC on tape:

```
(A1X1)
```

To convert 30 characters of internal 6-bit Display Code to 8-bit EBCDIC on tape:

```
(X30X30)
or (30X1X1)
or (30X1)
or (X30)
```

The last two examples assume the default given in table 2-3.
The following examples illustrate conversion strings for extended applications. In all examples, the source is an internal file and the destination is an EBCDIC tape file.

( X1 EQ $1S : 9X1X2 ; X1 EQ $2S : 9X1X3 ; X1X1 )

If the first character in the internal record equals a literal 1, move consecutive 1-character internal source fields to consecutive 2-character tape destination fields 9 times. In the destination, each of the nine EBCDIC characters converted from Display Code will be separated by spaces. If the first character is equal to a literal 2, move a 1-character source field to a 3-character destination field 9 times. As a result, two spaces will separate each of the nine converted characters. If the first character is neither 1 or 2, it is converted to an 8-bit EBCDIC character in the tape destination field.

(3C4 EQ $BCDE$ : C10X15 ; 11C2 EQ *F1* ; C6X6 ; C5)

Four internal 12-bit EBCDIC characters starting in byte 3 are tested first in this example. Bytes 3 and 4 in the word contain character 2. If characters 2 through 5 are equal to the literal BCDE, ten 12-bit EBCDIC characters are written to the destination field as ten 8-bit EBCDIC characters with 5 blank characters appended. If the 4 characters are not equal to BCDE, then characters 6 and 7 are tested. (Bytes 1 through 4 in the next word contain these two characters.) If they are equal to the literal F1, six internal 12-bit EBCDIC characters are written to the destination field as six 8-bit EBCDIC characters. If neither relationship is true, then five 12-bit EBCDIC characters are written to the destination field as five 8-bit EBCDIC characters.

(5X2 EQ *23* : X5 ; X1X0, (X1 NE $2S : X9X10 ; X4X0,X5X5))

Starting with the fifth character in a string, if the two characters (5 and 6) are equal to the literal 23, convert the first five characters in the string to 8-bit EBCDIC and terminate conversion of the remainder of the record.

If the fifth and sixth characters are not equal to 23, skip the first character in the source field with X1X0 and test the next single character. If not equal to 2, convert 9 characters (the second through the tenth) from internal Display Code to external EBCDIC and append one blank character.

If the two relationships are not true, skip the next 4 characters (the second through the fifth) and convert the next 5 characters from Display Code to 8-bit EBCDIC.

(X8X0,(4X3 EQ +6X3 : 10X1X2 ; 10X1X3))

Skip over the first 8 characters in the internal record. If three Display Code characters starting with the fifth in the record (characters 5-7) are equal to the three characters starting with the 15th (characters 15-17), convert ten single Display Code characters to 8-bit EBCDIC and insert a blank between each. If the relationship is not true, 10 characters are converted to EBCDIC with two blanks inserted between each in the destination record.
FORTRAN INPUT/OUTPUT EXAMPLES

It is expected that double-precision and floating-point usage will be of more interest to FORTRAN users; therefore, such information is given here rather than among the COBOL examples, although it is applicable to both.

A typical FORTRAN Extended job using the 8-bit subroutines might appear as follows:

```
JOBNAME,CM6000,T200.
†ACCOUNT CARD
FTN.
††REQUEST,TAPE1,NT,S.
   FILE(TAPE1,MRL=134,RT=U,BT=K,MBL=134,MNB=24,MNR=24)
   LDSET.FILES=TAPE1)
   LGO.
   7/8/9
```

To determine the proper record format and values for the FILE control card, refer to appendix B and section 6, System Interfaces.

Before any reference to the XREAD or XWRITE routines, a call to the XFILE routine must appear in the program. Such a call might appear as:

```
CALL XFILE (1,WSA,≠(FT=T,USE=W,RECFM=VSB,BLKSIZE=100,LRECL=150,CODE=C)≠,40)
```

To move ten 8-bit EBCDIC characters from a tape file, convert them to ten internal 12-bit EBCDIC characters, and move them to the destination array IRAY, the following statement would be written:

```
CALL XREAD(WSA,IRAY,≠(X10C10)≠,STAT)
```

To reverse the procedure and write the ten 12-bit EBCDIC internal characters from the source array IRAY, the following statement would be written:

```
CALL XWRITE(WSA,IRAY,≠(C10X10)≠,STAT)
```

The following is an example of a tape writing program, showing bit conversion strings in the calling sequence to XWRITE:

```
INTEGER NFIELDS(156), WSA(520)
    
    CALL XFILE(1,WSA,≠(FT=T,USE=W,RECFM=F,BLKSZ=3840)≠,520)
CALL XWRITE(WSA,NFIELDS,≠(512(B0B52,N3B8))≠)
```

††In KRONOS, the account card follows the job card in all decks.
†††The format of the KRONOS request card is: REQUEST,TAPE1,NT,F=S
The following FORTRAN Extended program would accomplish the same tape reading operation described under the COBOL tape reading example:

```
INTEGER SZ, CSTRING, IRAY(30), WSA(46)
DATA STAT/0.0/, CSTRING/*(X30C30)*/, SZ/46/

CALL XFILE(1, WSA, *(FT=T, USE=W, RECFM=FB, BLKSZ=150, LRECL=30, CODE=C)*/, SZ)
CALL XREAD(WSA, IRAY, CSTRING, STAT)
IF(STAT .NE. 0.0) PRINT 10
10 FORMAT(*FAILED IN READ*)
```

The COBOL example defines the file-string and conversion-string parameters in the working storage section. In this FORTRAN example, the DATA statement contains the optional definition of the conversion-string symbol CSTRING and that symbol is used in the XREAD calling sequence. Conversely, the file-string parameter might have been defined in the DATA statement and its symbolic name used as a parameter in the XFILEx calling sequence.

The following example accomplishes the same tape writing operation as that described under COBOL tape writing example:

```
INTEGER FSTRING(5), IRAY(30), WSA(46)
DATA FSTRING/*(FT=T, USE=W, RECFM=FB, BLKSZ=150, LRECL=30, CODE=C)*/,

STAT=0.0
CALL XFILE(1, WSA, FSTRING, 46)
CALL XWRITE(WSA, IRAY, *(X30C30)*/, STAT)
IF(STAT .NE. 0.0) PRINT 20
20 FORMAT(*FAILED IN WRITE*)
```

**COMPASS INPUT/OUTPUT EXAMPLES**

This program reads ten 8-bit EBCDIC characters from a tape file working storage area WS1, converts them to internal EBCDIC and places them in area IRAY. After conversion is complete, the EBCDIC characters are to be written from the area IRAY to workstorage area WS2; from there, the characters will be written to tape.
<table>
<thead>
<tr>
<th>Location</th>
<th>Operation</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA1</td>
<td>T1LIST</td>
<td></td>
</tr>
<tr>
<td>RJ</td>
<td>XFILE</td>
<td></td>
</tr>
<tr>
<td>SA1</td>
<td>T2LIST</td>
<td></td>
</tr>
<tr>
<td>RJ</td>
<td>XFILE</td>
<td></td>
</tr>
<tr>
<td>SA1</td>
<td>RDLST</td>
<td></td>
</tr>
<tr>
<td>RJ</td>
<td>XREAD</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SA1</td>
<td>WRLST</td>
<td></td>
</tr>
<tr>
<td>RJ</td>
<td>XWRITE</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T1LIST</td>
<td>VFD</td>
<td>60/FNAME1,60/WS1,60/FSTR1,60/SIZE1,60/0</td>
</tr>
<tr>
<td>T2LIST</td>
<td>VFD</td>
<td>60/FNAME2,60/WS2,60/FSTR2,60/SIZE2,60/0</td>
</tr>
<tr>
<td>RDLST</td>
<td>VFD</td>
<td>60/WS1,60/IRAY,60/CSTR1,60/STAT,60/0</td>
</tr>
<tr>
<td>WRLST</td>
<td>VFD</td>
<td>60/WS2,60/IRAY,60/CSTR2,60/STAT,60/0</td>
</tr>
<tr>
<td>FSTR1</td>
<td>DATA</td>
<td>51L(FT=T,USE=R,RECFM=VSB,BLKSZIE=100,LRECL=150,CODE=C)</td>
</tr>
<tr>
<td>FSTR2</td>
<td>DATA</td>
<td>51L(FT=T,USE=W,RECFM=VSB,BLKSZIE=100,LRECL=150,CODE=C)</td>
</tr>
<tr>
<td>CSTR1</td>
<td>DATA</td>
<td>8L(X10C10)</td>
</tr>
<tr>
<td>CSTR2</td>
<td>DATA</td>
<td>8L(C10X10)</td>
</tr>
<tr>
<td>WS1</td>
<td>BSS</td>
<td>40</td>
</tr>
<tr>
<td>WS2</td>
<td>BSS</td>
<td>40</td>
</tr>
<tr>
<td>IRAY</td>
<td>BSS</td>
<td>60</td>
</tr>
<tr>
<td>STAT</td>
<td>BSS</td>
<td>1</td>
</tr>
<tr>
<td>FNAME1</td>
<td>VFD</td>
<td>60/5LFILE1</td>
</tr>
<tr>
<td>FNAME2</td>
<td>VFD</td>
<td>60/5LFILE2</td>
</tr>
<tr>
<td>SIZE1</td>
<td>VFD</td>
<td>60/40</td>
</tr>
<tr>
<td>SIZE2</td>
<td>VFD</td>
<td>60/52</td>
</tr>
</tbody>
</table>
The four utility subroutines described in this chapter perform a variety of operations including comparison and moving of data strings in any of three internal forms—12-bit ASCII, 12-bit EBCDIC, and 6-bit Display Code.

UTILITY SUBROUTINE PARAMETERS

For the calling sequences to the utility subroutines, the following general information applies to the parameters.

Integer Parameters (length, position)

In a COBOL calling sequence to a utility subroutine, the length and position parameters must be an integer COMPUTATIONAL-1 item, containing no more than 14 decimal digits. In FORTRAN, the parameters may be integer constants, variables or expressions. In COMPASS, the parameters must be the location symbols of words containing integer values.

Real Parameter (status)

The status parameter in a COBOL calling sequence must be a COMPUTATIONAL-2 item. In FORTRAN, it must be a real variable name. In COMPASS, it must be the location symbol of a word that contains a real (floating point) value.

String Parameter (xy)

In a COBOL calling sequence, this parameter is an elementary 01 level alphanumeric literal or a SYNC LEFT DISPLAY data item described in the Data Division with VALUE set to the desired two or three character string. In FORTRAN, the parameter may be a variable name defined by nH or nL to contain the desired characters; or it may be a constant written in nH or nL form or set off by '" or # quotation symbols. In COMPASS, the parameter must be written as a location symbol of a word containing the character data.

XCOMP STRING COMPARISON

The XCOMP subroutine compares two character strings not necessarily of the same character code.

FORTRAN Calling Sequence:

\[ \text{[CALL] \ XCOMP} (xy, \text{source-1, source-2, status, length} \begin{cases} \text{.position-1} \\ \text{.position-1, position-2} \end{cases} \) \]
XCOMP may be called as a subroutine or used as a function.

**xy**

Two characters describing the character sets of the strings to be compared: 
x describes the source-1 character string code; 
y describes the source-2 character string code as follows:

A 12-bit ASCII code character string
C 12-bit EBCDIC code character string
X 6-bit Display Code character string

When xy is presented in the call as a literal, it must be written 2Hxy or 2Lxy. When a variable name is used in the call, the variable must be defined previously as containing the characters for xy left justified.

**source**

Location containing first character in a string to be compared. One source parameter may be an actual Hollerith string written in the calling sequence.

**status**

Real variable name in which the result of the comparison is returned:

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Status Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>source-1 .LT. source-2</td>
<td>-1.0</td>
</tr>
<tr>
<td>source-1 .EQ. source-2</td>
<td>0.0</td>
</tr>
<tr>
<td>source-1 .GT. source-2</td>
<td>+1.0</td>
</tr>
</tbody>
</table>

When XCOMP is used as a function, the status value will be returned also as the value of the function.

**length**

Required integer constant or previously defined integer variable denoting the number of characters to be compared.

**position**

Optional integer constant or variable that indexes the position, with the group, of the first character of a string to be compared. If omitted, the default value is 1, denoting the leftmost (first character) in the source group.
COBOL Calling Sequence:

```
ENTER XCOMP USING xy,source-1,source-2,status [ {length
  ,length,position-1
  ,length,position-1,position-2 } ]
```

xy An alphanumeric literal or a data name described in DATA DIVISION as alphabetic or alphanumerical, size 2. The data name contains two characters describing the character sets of each character string to be compared. The first character (x) describes source-1; the second (y) describes source-2 as follows:

A 12-bit ASCII character string
C 12-bit EBCDIC character string
X 6-bit Display Code character string

source Data item containing the first character of each character string to be compared.

status COMPUTATIONAL-2 item to contain a return value representing the result of the comparison:

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Status Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>source-1 &lt; source-2</td>
<td>-1.0</td>
</tr>
<tr>
<td>source-2 = source-2</td>
<td>0.0</td>
</tr>
<tr>
<td>source-1 &gt; source-2</td>
<td>+1.0</td>
</tr>
</tbody>
</table>

length Optional COMPUTATIONAL-1 item representing number of characters to be compared. If omitted, the character size of the longer source character field is used. When strings of unequal-length are compared, the shorter field is treated as blank-filled on the right to equal the longer field size. When given, the length specification overrides actual string lengths.

position Optional COMPUTATIONAL-1 item designating initial character position to be used for comparison in the source string. When omitted, the default value is 1, designating the first character in the group. If position is designated, length also must be specified.
COMPASS Calling Sequence:

<table>
<thead>
<tr>
<th>Location</th>
<th>Operation</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA1, RJ</td>
<td>plist</td>
<td>XCOMP</td>
</tr>
<tr>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>VFD</td>
<td>60/xy, 60/source-1, 60/source-2, 60/status, 60/length</td>
<td></td>
</tr>
</tbody>
</table>
| VFD      | \[
|          | \left\{ 60/\text{position-1} \\
|          | \{} 60/\text{position-1}, 60/\text{position-2} \right\} .60/0 |

plist

Symbolic location of parameter list which must end with word of binary zeros.

xy

Symbolic location of literal containing two characters, left adjusted, describing the code form of each character string. The literal may be described as either 2Hxy or 2Lxy, the fill characters of blank or zero are of no consequence. x describes the character set of source-1; y describes the character set of source-2, as follows:

A 12-bit ASCII code character string

C 12-bit EBCDIC code character string

X 6-bit Display Code character string

source

Symbolic location of word containing the first character in a string.

status

Location symbol of a word in which a floating-point value is returned; the value reflects the result of the string comparison, as follows:

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Status Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>source-1 &gt; source-2</td>
<td>-1.0</td>
</tr>
<tr>
<td>source-1 = source-2</td>
<td>0.0</td>
</tr>
<tr>
<td>source-1 &lt; source-2</td>
<td>+1.0</td>
</tr>
</tbody>
</table>

length

Symbolic location of a word containing an integer value representing the number of sequential characters to be compared.

position-s,

position-d

Optional symbolic location of an integer value indicating the position of the first character in each string. When comparison begins with the first character in the source group, the position is 1. The default value is 1, when the parameter is omitted.
COLLATING SEQUENCE

The type of source-I string determines the collating sequence used:

AC specifies the ASCII collating sequence (numbers low)

CA specifies the EBCDIC collating sequence (numbers high)

XA or XC specify the 8-bit string is to be case-folded logically (according to table on page A-7 of appendix A) during comparison, so that upper and lower case letters are equal.

AX or CX specify upper and lower case letters are to retain their identity and collate separately.

In the 8-bit ASCII and EBCDIC collating sequences, ascending binary code values correspond to ascending collating values. 6-bit Display Code, however, can provide a variation through installation option. Appendix A gives the 6-bit Display Code collating sequences.

COMPARISON EXAMPLES

The examples for each utility routine given in this chapter use strings from the character groups S1DISPC and S2ASCII. Each group is stored in central memory; S1DISPC is stored in Display Code in 6-bit bytes as follows:

<table>
<thead>
<tr>
<th></th>
<th>Word 1</th>
<th>Word 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>C</td>
<td>D</td>
</tr>
<tr>
<td>E</td>
<td>F</td>
<td>G</td>
</tr>
<tr>
<td>H</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>.</td>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>Z</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

S2ASCII is stored in ASCII code in 12-bit bytes as follows:

<table>
<thead>
<tr>
<th></th>
<th>Word 1</th>
<th>Word 2</th>
<th>Word 3</th>
<th>Word 4</th>
<th>Word 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
<td>Y</td>
<td>X</td>
<td>A</td>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>F</td>
<td>G</td>
</tr>
<tr>
<td>H</td>
<td>2</td>
<td>3</td>
<td>.</td>
<td>X</td>
<td>Y</td>
</tr>
<tr>
<td>Z</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

A five-word work storage area WS1 also will be used.
Example of FORTRAN comparison:

Compare 20 characters, starting with character position 1 of S1DISPC and character position 4 of S2ASCII. Return the status information in the variable XSTAT.

INTEGER S2ASCII(5), S1DISPC(2)
.
.
CALL XCOMP(2LXA, S1DISPC, S2ASCII, XSTAT, 20, 1, 4)
.
.
The comparison may be performed in a replacement statement, using XCOMP as a function, as follows:

XSTAT = XCOMP(2LXA, S1DISPC, S2ASCII, XSTAT, 20, 1, 4)
IF(XSTAT) 10, 20, 30
.
.
The IF statement is used to direct the execution of appropriate parts of the program, depending upon the results of the comparison.

Example of COBOL comparison:

Compare 20 characters, starting with character position 1 of S1DISPC and character position 4 of S2ASCII. Return the status information in item XSTAT.

DATA DIVISION.
01 EXWHY PICTURE(XX) VALUE ≠XA≠.
01 S1DISPC PICTURE X(20).
01 S2ASCII PICTURE X(50).
01 XSTAT USAGE IS COMPUTATIONAL-2.
01 LENGTH USAGE IS COMPUTATIONAL-1 VALUE IS 20 PIC 99.
01 POSONE USAGE IS COMPUTATIONAL-1 VALUE IS 1 PIC 99.
01 POSTWO USAGE IS COMPUTATIONAL-1 VALUE IS 4 PIC 99.
.
.
PROCEDURE DIVISION.
.
.
ENTER XCOMP USING EXWHY, S1DISPC, S2ASCII, XSTAT, LENGTH, POSONE, POSTWO.
.
.
Example of COMPASS comparison:

Compare 20 characters, starting with character position 1 of S1DISPC and character 4 of S2ASCII. Return the status information in location XSTAT.

<table>
<thead>
<tr>
<th>Location</th>
<th>Operation</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SA1</td>
<td>PLIST</td>
</tr>
<tr>
<td></td>
<td>RJ</td>
<td>XCOMP</td>
</tr>
<tr>
<td></td>
<td>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>PLIST</td>
<td>VFD</td>
<td>42/0,18/XY,42/0,18/S1DISPC,42/0,18/S2ASCII,42/0,18/XSTAT</td>
</tr>
<tr>
<td></td>
<td>VFD</td>
<td>42/0,18/LGTPOS,42/0,18/P1,42/0,18/P2,60/0</td>
</tr>
<tr>
<td></td>
<td>.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>XY</td>
<td>VFD</td>
<td>18/2LXA,42/0</td>
</tr>
<tr>
<td>S1DISPC</td>
<td>BSS</td>
<td>2</td>
</tr>
<tr>
<td>S2ASCII</td>
<td>BSS</td>
<td>5</td>
</tr>
<tr>
<td>XSTAT</td>
<td>BSSZ</td>
<td>1</td>
</tr>
<tr>
<td>LGTPOS</td>
<td>DATA</td>
<td>20</td>
</tr>
<tr>
<td>P1</td>
<td>DATA</td>
<td>1</td>
</tr>
<tr>
<td>P2</td>
<td>DATA</td>
<td>4</td>
</tr>
</tbody>
</table>

**XMOVE STRING MOVE**

The XMOVE routine moves a designated character string from a source location to a destination location, translating from one character code to another during the move.

When no character conversion is involved, XMOVE transfers character strings in word groups. If character conversion is specified, the move is done character by character. The XMOVE subroutine operates on character strings in a left-to-right sequence. The programmer must be aware that characters may not be moved as expected when the source and destination fields overlap.

**FORTRAN Calling Sequence**

```fortran
CALL XMOVE (xy,source,destination,length \[ \{\text{position-s} \} \{\text{position-s,position-d}\} )
```

*xy*  
Descriptors for source and destination string character sets: *x* describes the source, *y* describes the destination. The descriptors may be two or three characters, as follows:

- **A**  
  String of 12-bit characters in ASCII

- **C**  
  String of 12-bit characters in EBCDIC

- **X**  
  String of 6-bit characters in Display Code
Three-character parameters begin with X:

XAL    Moves 6-bit to 12-bit ASCII, with case reversed
XCL    Moves 6-bit to 12-bit EBCDIC, with case reversed

When XAL and XCL codes are used, the characters appear in the destination string in lower case rather than in the usual upper case. Special symbolic characters are also case-reversed; as a result, printable characters may become non-printable. A table of upper and lower case characters appears on page A-7.

The xy parameter may be written as a literal constant or as a variable previously defined to contain the appropriate characters. In either case, the variable or constant should be defined as 2Hxy or 2Lxy, or as 3Hxy or 3Lxy, if case reversal is used.

source    Starting location of source character string to be moved, given as a variable name, an array name, a subscripted array name, or optionally as a Hollerith or literal constant when the source string code type is X.

destination    Starting location of destination given as a variable name, an array name, or a subscripted array name.

length    Integer or integer variable designating the number of characters in the string to be moved.

definitions:
position-s    Optional integer or integer variable that indexes the first character in the source string to be moved. If not given, the default value is 1, denoting the leftmost character is the first character in the string.

position-d    Optional integer or integer variable that indexes the first receiving character in the destination string. If not given, the default value is 1, denoting the leftmost character is the first character in the string.

COBOL Calling Sequence:

ENTER XMOVE USING xy,source,destination [ (length (length-position-s) ) ]

xy    Data name or alphabetic literal descriptors for source and destination string character codes: x describes the source, y describes the destination. The descriptors may be two or three characters, as follows:

A    String of 12-bit character in ASCII
C    String of 12-bit character in EBCDIC
X    String of 6-bit character is Display Code.
Three-character parameters begin with X:

XAL    Moves 6-bit to 12-bit ASCII, with case reversed
XCL    Moves 6-bit to 12-bit EBCDIC, with case reversed

When XAL and XCL codes are used, the characters appear in the destination string in lower case rather than in the usual upper case. Special symbolic characters are also case-reversed; as a result, printable characters may become non-printable. A table of upper and lower case characters appears on page A-7.

The size of xy field should be two- or three-character alphabetic or alphanumerical data names or literals.

<table>
<thead>
<tr>
<th>Location</th>
<th>Operation</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA1</td>
<td>plist</td>
<td></td>
</tr>
<tr>
<td>RJ</td>
<td>XMOVE</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>plist</td>
<td>VFD</td>
<td>60/xy,60/source,60/destination,60/length</td>
</tr>
<tr>
<td></td>
<td>VFD</td>
<td>{60/position-s,60/position-d},60/0</td>
</tr>
</tbody>
</table>

COMPASS Calling Sequence:
Symbolic location of parameter list which must end with word of binary zeros.

Symbolic location of descriptors for source and destination string character codes: x describes the source; y describes the destination. The descriptors may be two or three characters, as follows:

A String of 12-bit characters in ASCII
C String of 12-bit characters in EBCDIC
X String of 6-bit characters in Display Code

Three-character parameters begin with X:

XAL Moves 6-bit to 12-bit ASCII, with case reversed
XCL Moves 6-bit to 12-bit EBCDIC, with case reversed

When XAL and XCL codes are used, the characters appear in the destination string in lower case rather than in the usual upper case. Special symbolic characters are also case-reversed; as a result, printable characters may become non-printable. A table of upper and lower case characters appears on page A-7.

The xy parameter may be written as a literal or a variable previously defined to contain the appropriate characters. In either case, the variable or literal should be defined as 2Hxy or 2Lxy, or when case reversal is used as 3Hxy or 3Lxy.

source Starting location of area containing the character string to be moved.

destination Starting location of destination area to receive the character string.

length Symbolic location of word containing an integer value, denoting length of character string to be moved.

position-s Symbolic location of integer value indicating the position of the first character in the source string (position-s) and the first receiving character in the destination string (position-d). If not given, the default value is 1, denoting the left-most character position.
MOVE EXAMPLES

The character strings described for the COMPARISON EXAMPLES on page 4-5 are to be used for these examples.

Example of FORTRAN move:

Move a 20-character ASCII code string, starting with character 4 of the group stored in S2ASCII, convert the string to 12-bit EBCDIC and place it in array WS1, starting with character 1.

INTEGER S2ASCII(5),WS1(5)
.
.
CALL XMOVE (2HAC,S2ASCII,WS1,20,4)

Omission of a value for parameter position-d implies character position 1 by default.

Example of COBOL move:

01 XY PIC XX VALUE ≠AC≠.
01 S2ASCII PIC X(50).
01 WS1 PIC X(50).
01 LENGTH USAGE IS COMPUTATIONAL-1 VALUE IS 20 PIC 99.
01 POSIT USAGE IS COMPUTATIONAL-1 VALUE is 4 PIC 99.
01 WS2 PIC X(60).
.
ENTER XMOVE USING XY,S2ASCII,WS1,LENGTH,POSIT.

This example moves a 20-character 12-bit ASCII code string starting with character 4 of the group stored in S2ASCII, converts the string to 12-bit EBCDIC, and places it in WS1, starting with character 1.

ENTER XMOVE USING ≠AC≠,S2ASCII,WS2.

This example moves a 25-character, 12-bit ASCII code string from S2ASCII, converts the string to 12-bit EBCDIC, appends five 12-bit EBCDIC spaces on the right, then places the entire string in WS2.
Example of COMPASS move:

Move a 20-character ASCII code string starting in character position 4 of S2ASCII, convert it to 12-bit EBCDIC and place it starting in character position 1 of WS1.

<table>
<thead>
<tr>
<th>Location</th>
<th>Operation</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA1</td>
<td>PLIST</td>
<td></td>
</tr>
<tr>
<td>RJ</td>
<td>XMOVE</td>
<td></td>
</tr>
<tr>
<td>PLIST</td>
<td>VFD</td>
<td>42/0,18/XY</td>
</tr>
<tr>
<td></td>
<td>VFD</td>
<td>42/0,18/S2ASCII</td>
</tr>
<tr>
<td></td>
<td>VFD</td>
<td>42/0,18/WS1</td>
</tr>
<tr>
<td></td>
<td>VFD</td>
<td>42/0,18/LEN</td>
</tr>
<tr>
<td></td>
<td>VFD</td>
<td>42/0,18/LEN+1</td>
</tr>
<tr>
<td>BSSZ</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XY</td>
<td>VFD</td>
<td>18/2LAC,42/0</td>
</tr>
<tr>
<td>S2ASCII</td>
<td>BSS</td>
<td>5</td>
</tr>
<tr>
<td>WS1</td>
<td>BSS</td>
<td>5</td>
</tr>
<tr>
<td>LEN</td>
<td>DATA</td>
<td>20,4</td>
</tr>
</tbody>
</table>

**XPACK STRING COMPRESSION**

This subroutine will compress 8-bit character data from a 5-character per word internal format containing 12-bit ASCII or EBCDIC characters. When packing is performed, seven 8-bit character codes are right adjusted in a word; and the leftmost 4 bits, as well as any unused character positions are set to binary zero. The packed word format is:

```
<table>
<thead>
<tr>
<th>CM</th>
<th>CH</th>
<th>CH</th>
<th>CH</th>
<th>CH</th>
<th>CH</th>
<th>CH</th>
<th>CH</th>
<th>CH</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>
```

No conversion of character data takes place during string compression.

Several characters are moved at once; when source and destination fields overlap, the characters may not be moved as expected.
FORTRAN Calling Sequence:

CALL XPACK(string-u,string-p,length[,position])

string-u    Symbolic location of 12-bit (unpacked) character source string.
string-p    Symbolic location of 8-bit (packed) character destination string.
length    Number of characters to be moved and packed; may be an integer constant or variable.
position    Optional integer constant or variable that indexes the source character group for the first character in the string to be packed. If omitted, the default value is 1, indicating the first character in the source group.

COBOL Calling Sequence:

ENTER XPACK USING string-u,string-p [{length} {length,position}]

string-u    Location of 12-bit (unpacked) character source string
string-p    Location of 8-bit character (packed) destination string. The string must correspond to a word boundary. Every 01 item in COBOL begins on a word boundary.
length    Optional COMPUTATIONAL-1 item giving the number of characters to be moved and packed. If omitted, the default is the length in number of characters of the shorter string. Binary fill completes the receiving field as needed.
position    Optional COMPUTATIONAL-1 item which indexes the first character in string-u to be packed in string-p. The default value is 1.

COMPASS Calling Sequence:

<table>
<thead>
<tr>
<th>Location</th>
<th>Operation</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SA1</td>
<td>plist</td>
</tr>
<tr>
<td></td>
<td>RJ</td>
<td>XPACK</td>
</tr>
<tr>
<td></td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td></td>
<td>plist</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VFD</td>
<td>60/string-u</td>
</tr>
<tr>
<td></td>
<td>VFD</td>
<td>60/string-p</td>
</tr>
<tr>
<td></td>
<td>VFD</td>
<td>60/length</td>
</tr>
<tr>
<td></td>
<td>VFD</td>
<td>60/position</td>
</tr>
<tr>
<td></td>
<td>VFD</td>
<td>60/0</td>
</tr>
</tbody>
</table>

plist    Symbolic location of parameter list which must end with word of binary zeros.
string-u    Storage location of 12-bit (unpacked) character string source.
string-p Storage location of 8-bit (packed) character string destination.

length Symbolic location of an integer value giving number of characters in the string to be packed.

position Optional symbolic location of an integer value indicating the character position within the group of the first character of the string to be packed. Default value is 1.

PACK EXAMPLES

Example of FORTRAN packing:

Pack into array WS1 a 20-character string starting with character 4 of the ASCII code group stored in array S2ASCII.

INTEGER WS1(3),S2ASCII(5)
  .
  .
CALL XPACK(S2ASCII,WS1,20,4)

In the destination array WS1, the first two words will be filled with 7 characters each; the third word will contain 6 characters and an 8-bit byte filled with binary zeros.

Example of COBOL packing:

Pack a 20-character string into WS1 starting with character 4 of the ASCII code group stored in array S2ASCII.

01 S2ASCII PICTURE X(50).
01 WS1 PICTURE X(30).
01 LENGTH USAGE IS COMPUTATIONAL-1 VALUE IS 20 PIC 99.
01 POSIT USAGE IS COMPUTATIONAL-1 VALUE IS 4 PIC 99.
  .
ENTER XPACK USING S2ASCII,WS1,LENGTH,POSIT.
  .
Example of COMPASS packing:

<table>
<thead>
<tr>
<th>Location</th>
<th>Operation</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA1</td>
<td></td>
<td>PLIST</td>
</tr>
<tr>
<td>RJ</td>
<td></td>
<td>XPACK</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLIST</td>
<td>VFD</td>
<td>60/S2ASCII,60/WSI,60/LENGTH,60/LENGTH+1,60/0</td>
</tr>
<tr>
<td>S2ASCII</td>
<td>BSS</td>
<td>5</td>
</tr>
<tr>
<td>WS1</td>
<td>BSS</td>
<td>3</td>
</tr>
<tr>
<td>LENGTH</td>
<td>DATA</td>
<td>20,4</td>
</tr>
</tbody>
</table>

This example will pack a 20-character string starting at character 4 of the group stored starting in S2ASCII into the storage area WS1.

XPAND STRING EXPANSION

After character strings are packed and written to file storage, the file information density is increased and storage is used more economically. If the character strings are required for use in central memory again, they must be unpacked into 12-bit bytes for proper handling. The XPAND routine reverses the process performed by XCOMP and unpacks the 8-bit compressed string into words containing five 12-bit character bytes.

FORTRAN Calling Sequence:

\[
\text{CALL} \text{XPAND}(\text{string-u, string-p, length[, position]})
\]

- \text{string-u} Location of the 8-bit unpacked characters in the string source area.
- \text{string-p} Location of the 12-bit packed characters in the string destination area.
- \text{length} Integer constant or variable indicating the number of characters to be unpacked.
- \text{position} Integer constant or variable which indexes the first character of the string to be unpacked. If omitted, the default value is 1.
COBOL Calling Sequence:

\[
\text{ENTER XPAND USING string-u,string-p } \{\text{length,position} \}
\]

- **string-u**: Location where the unpacked string is to be placed — the destination area of the string.
- **string-p**: Location of the packed string in the string source area; this location must correspond with a word boundary; in COBOL, it must be a 01 item.
- **length**: Optional COMPUTATIONAL-1 item giving number of characters to be unpacked. If omitted, the default is the character length of the shorter string; binary zero fill completes the destination area if needed.
- **position**: Optional COMPUTATIONAL-1 item indexing the position of the first character in the string-u (destination) area. If omitted, the default value is 1, designating the leftmost character position in the destination area.

COMPASS Calling Sequence:

<table>
<thead>
<tr>
<th>Location</th>
<th>Operation</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA1</td>
<td>plist</td>
<td></td>
</tr>
<tr>
<td>RJ</td>
<td>XPAND</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>plist</td>
<td>VFD</td>
<td>60/string-u,60/string-p {60/length,60/position},60/0</td>
</tr>
</tbody>
</table>

- **plist**: Symbolic location of parameter list which must be terminated by a word of binary zeros.
- **string-u**: Storage location of unpacked 12-bit character in the destination area.
- **string-p**: Storage location of packed 8-bit character string in the source area.
- **length**: Symbolic location of integer value giving number of characters in the string to be unpacked.
- **position**: Optional symbolic location of integer value indicating the position of the first character in the string-u (destination) area. If omitted, the default value is 1, designating the leftmost character in the destination area.
EXPAND EXAMPLES

Example of FORTRAN expand:

Reversing the example given for XPAR, expand a 20-character string that has been read from a file into central memory storage array WS1. Restore the expanded string to array S2ASCII, starting at character position 4.

```
INTEGER WS1(3),S2ASCII(5)
  .
  .
CALL XPAND(S2ASCII,WS1,20,4)
```

Example of COBOL expand:

Reversing the example given for XPAR, expand a 20-character string that has been read from a file into memory storage area WS1. Restore the expanded string to area S2ASCII, starting at character position 4. The calling sequence would be:

```
ENTER XPAND USING S2ASCII,WS1,LENGTH,POSIT.
```

Example of COMPASS expand:

Reversing the example given for XPAR, expand a 20-character string that has been read from a file into memory storage area WS1. Restore the expanded string to area S2ASCII, starting in character position 4. The COMPASS calling sequence would be:

<table>
<thead>
<tr>
<th>Location</th>
<th>Operation</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SAI</td>
<td>PLIST</td>
</tr>
<tr>
<td></td>
<td>RJ</td>
<td>XPAND</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLIST</td>
<td>VFD</td>
<td>42/0,18/S2ASCII</td>
</tr>
<tr>
<td></td>
<td>VFD</td>
<td>42/0,18/WS1</td>
</tr>
<tr>
<td></td>
<td>VFD</td>
<td>42/0,18/LENPOS</td>
</tr>
<tr>
<td></td>
<td>VFD</td>
<td>42/0,18/LENPOS+1</td>
</tr>
<tr>
<td></td>
<td>BSSZ</td>
<td>1</td>
</tr>
<tr>
<td>WS1</td>
<td>BSS</td>
<td>3</td>
</tr>
<tr>
<td>S2ASCII</td>
<td>BSS</td>
<td>5</td>
</tr>
<tr>
<td>LENPOS</td>
<td>DATA</td>
<td>20,4</td>
</tr>
</tbody>
</table>
COPY8P PROGRAM

COPY8P is a separate utility program used to copy IBM 360/370 print files to CDC CYBER 70 compatible print files. Upper and lower case character capability is maintained, as loss of 8-bit significance is avoided. IBM options with respect to record type, block type and print format selection are available. COPY8P does not use Record Manager in copying the print files; therefore, no file cards are needed in conjunction with this program.

CALLING COPY8P

COPY8P is called by a control card in the job deck. Certain mandatory information, such as input and output local file names, is specified by control card parameters. The control card is written:

COPY8P,in-lfn,out-lfn,options.

in-lfn Local file name of the input file containing 8-bit ASCII or EBCDIC data in IBM 360/370 format. Under normal conditions, the file is on magnetic tape made available to the job by a REQUEST card. Files copied from tape to disk by a utility routine are acceptable as input as long as the same 8-bit characters and control information are still available. Such a copy of a file from tape to disk, should provide one SCOPE logical record per block.

out-lfn Local file name of output tape or disk file to contain data in a format suitable for the printer. If full upper and lowercase information is to be printed, the file output must be directed to a printer with an extended character set print train by use of the DISPOSE control card. See Operating System Interface, section 6.

COPY8P OPTIONS

The following parameters may be specified as required on the COPY8P control card. If a parameter is not specified, the indicated default value is used. Parameters must be separated by commas.

.RECFM=rr (Default: .RECFM=U)

Describes the record format of the input file. Values for rr are: F, V, U, FB, and VB. They have the same meaning as the equivalent 360/370 JCL (Job Control Language) specification.

.BLKSIZE=nnnn (Default: .BLKSIZE=137)

Defines the maximum block length in 8-bit characters. The parameter nnnn is a decimal count. This parameter has the same meaning as the equivalent 360/370 JCL specification.
,LRECL=nnnn  (Default: omitted)
Defines the maximum logical record size in 8-bit characters. The maximum value which can be specified for LRECL is 5120, the parameter nnnn is a decimal count. This parameter has the same meaning as the equivalent 360/370 JCL specification. If omitted, LRECL is assumed to be the same as BLKSIZE.

,CODE=(A/C)  (Default: ,CODE=C)
Defines the character set code present on the input file. A means ASCII; C means EBCDIC. This parameter must match the code parameter used on the input file REQUEST control card.

,FOLD  (Default: omitted)
Causes output to be folded to a 64-character set, 6-bit character representation, for printing. Special characters not having a 64-character set representation will not be printed. If omitted, upper and lowercase information is preserved for printing.

Note: When a print file is folded, a system restriction prohibits the character pair :: from occurring at certain points in a print line corresponding to ends of central memory words. This possibility is monitored by COPY8P. If found, the second : is replaced by a space, so that inadvertent line termination will not occur.

An output file produced by COPY8P to be printed with full upper and lower case characters must be directed to the CDC 595-6 printer through the DISPOSE control card written in one of two formats:

DISPOSE,Ifn,*PE.

DISPOSE,Ifn,PE.

Ifn is the logical file name given as the out-Ifn parameter in the COPY8P control card, and is the file to be printed. The *PE parameter indicates the file is to be disposed at end of job rather than when the control card is encountered.

In KRONOS, an extended character set file can be printed through TELEX at a local ASCII terminal.

If the FOLD option is used on the COPY8P control card, the out-Ifn contains only upper case characters; out-Ifn may be copied directly to the job OUTPUT file; or it may be disposed, using the DISPOSE control card in one of the following formats:

DISPOSE,Ifn,PR.

DISPOSE,Ifn,*PR.

The logical file name (Ifn) is the same as the out-Ifn parameter given on the COPY8P control card.
,FMT=(1|2|3|A|M)  (Default: ,FMT=A)

Defines the print spacing convention to be used. 1, 2, and 3 yield single, double and triple space. A indicates the first character of each line image is assumed to contain a format control character. The following format control characters are recognized when A is specified:

+ No space before printing
blank Single space before printing
0 Double space before printing

M assumes the first character of each line image is one of the following IBM 1403 printer hardware control characters:

<table>
<thead>
<tr>
<th>Hexadecimal Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Write, no space after print</td>
</tr>
<tr>
<td>09</td>
<td>Write, space 1 after print</td>
</tr>
<tr>
<td>11</td>
<td>Write, space 2 after print</td>
</tr>
<tr>
<td>19</td>
<td>Write, space 3 after print</td>
</tr>
<tr>
<td>89</td>
<td>Write, page eject after print</td>
</tr>
<tr>
<td>0B</td>
<td>Space 1 line</td>
</tr>
<tr>
<td>13</td>
<td>Space 2 lines</td>
</tr>
<tr>
<td>1B</td>
<td>Space 3 lines</td>
</tr>
<tr>
<td>8B</td>
<td>Page eject</td>
</tr>
<tr>
<td>Other</td>
<td>Write, space 1 after print</td>
</tr>
</tbody>
</table>

CONTROL COMMANDS, the remainder of the line is not printed

CHARACTER SET RESTRICTIONS

The extended printer character set used is the 95-character ASCII character set available on a 595-6 print train. Certain EBCDIC characters input to COPY8P may be converted to other graphics according to the table which follows. If an ASCII or EBCDIC character is input that does not correspond to an available character, a blank is printed. The print conversion for COPY8P is not identical to that given in appendix A for the other routines.
### Table 5-1. COPY8P

#### Print Conversion Tables

<table>
<thead>
<tr>
<th>EBCDIC</th>
<th>prints as</th>
<th>ASCII</th>
<th>folds to</th>
<th>64 Char. Set</th>
<th>Subset</th>
<th>ASCII Subset</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP</td>
<td>a</td>
<td>SP</td>
<td>a</td>
<td>≤</td>
<td></td>
<td>SP a</td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td>!</td>
<td>A</td>
<td>v</td>
<td>A</td>
<td>! A</td>
</tr>
<tr>
<td>&quot;</td>
<td>B</td>
<td>&quot;</td>
<td>B</td>
<td>#</td>
<td>B</td>
<td>&quot; B</td>
</tr>
<tr>
<td>#</td>
<td>C</td>
<td>#</td>
<td>C</td>
<td>₵</td>
<td>C</td>
<td># C</td>
</tr>
<tr>
<td>$</td>
<td>D</td>
<td>$</td>
<td>D</td>
<td>₵</td>
<td>D</td>
<td>$ D</td>
</tr>
<tr>
<td>%</td>
<td>E</td>
<td>%</td>
<td>E</td>
<td>₵</td>
<td>E</td>
<td>% E</td>
</tr>
<tr>
<td>&amp;</td>
<td>F</td>
<td>&amp;</td>
<td>F</td>
<td>G</td>
<td>F</td>
<td>&amp; F</td>
</tr>
<tr>
<td>(</td>
<td>G</td>
<td>(</td>
<td>G</td>
<td>(</td>
<td>G</td>
<td>( G</td>
</tr>
<tr>
<td>)</td>
<td>H</td>
<td>)</td>
<td>H</td>
<td>)</td>
<td>H</td>
<td>) H</td>
</tr>
<tr>
<td>*</td>
<td>I</td>
<td>*</td>
<td>I</td>
<td>*</td>
<td>I</td>
<td>* I</td>
</tr>
<tr>
<td>+</td>
<td>J</td>
<td>+</td>
<td>J</td>
<td>+</td>
<td>J</td>
<td>+ J</td>
</tr>
<tr>
<td>,</td>
<td>K</td>
<td>,</td>
<td>K</td>
<td>,</td>
<td>K</td>
<td>, K</td>
</tr>
<tr>
<td>-</td>
<td>L</td>
<td>-</td>
<td>L</td>
<td>-</td>
<td>L</td>
<td>- L</td>
</tr>
<tr>
<td>/</td>
<td>M</td>
<td>/</td>
<td>M</td>
<td>/</td>
<td>M</td>
<td>/ M</td>
</tr>
<tr>
<td>0</td>
<td>N</td>
<td>0</td>
<td>N</td>
<td>0</td>
<td>N</td>
<td>0 N</td>
</tr>
<tr>
<td>1</td>
<td>O</td>
<td>1</td>
<td>O</td>
<td>1</td>
<td>O</td>
<td>1 O</td>
</tr>
<tr>
<td>2</td>
<td>P</td>
<td>2</td>
<td>P</td>
<td>2</td>
<td>P</td>
<td>2 P</td>
</tr>
<tr>
<td>3</td>
<td>Q</td>
<td>3</td>
<td>Q</td>
<td>3</td>
<td>Q</td>
<td>3 Q</td>
</tr>
<tr>
<td>4</td>
<td>R</td>
<td>4</td>
<td>R</td>
<td>4</td>
<td>R</td>
<td>4 R</td>
</tr>
<tr>
<td>5</td>
<td>S</td>
<td>5</td>
<td>S</td>
<td>5</td>
<td>S</td>
<td>5 S</td>
</tr>
<tr>
<td>6</td>
<td>T</td>
<td>6</td>
<td>T</td>
<td>6</td>
<td>T</td>
<td>6 T</td>
</tr>
<tr>
<td>7</td>
<td>U</td>
<td>7</td>
<td>U</td>
<td>7</td>
<td>U</td>
<td>7 U</td>
</tr>
<tr>
<td>8</td>
<td>V</td>
<td>8</td>
<td>V</td>
<td>8</td>
<td>V</td>
<td>8 V</td>
</tr>
<tr>
<td>9</td>
<td>W</td>
<td>9</td>
<td>W</td>
<td>9</td>
<td>W</td>
<td>9 W</td>
</tr>
<tr>
<td>;</td>
<td>X</td>
<td>;</td>
<td>X</td>
<td>;</td>
<td>X</td>
<td>; X</td>
</tr>
<tr>
<td>:</td>
<td>Y</td>
<td>:</td>
<td>Y</td>
<td>:</td>
<td>Y</td>
<td>: Y</td>
</tr>
<tr>
<td>$</td>
<td>Z</td>
<td>$</td>
<td>Z</td>
<td>$</td>
<td>Z</td>
<td>$ Z</td>
</tr>
<tr>
<td>&lt;</td>
<td>[</td>
<td>&lt;</td>
<td>[</td>
<td>[</td>
<td>[</td>
<td>[ [</td>
</tr>
<tr>
<td>?</td>
<td>\</td>
<td>?</td>
<td>\</td>
<td>?</td>
<td>\</td>
<td>? \</td>
</tr>
<tr>
<td>\</td>
<td>_</td>
<td>\</td>
<td>_</td>
<td>\</td>
<td>_</td>
<td>\ _</td>
</tr>
<tr>
<td>other</td>
<td>none</td>
<td>other</td>
<td>none</td>
<td></td>
<td>space</td>
<td>space</td>
</tr>
</tbody>
</table>

**Notes:**

1. This table is designed to give maximum print graphic matching.

2. In folded representations, column 2 stands for both columns 2 and 3 of the full tables.
The preceding table was produced from card input by using the following code to govern column spacing and line skipping. This table is a valid example of card input and data code conversion:

CDFIL,CM65000,T120.
COPYBR, INPUT, TAPE01.
RFWIND(TAPE01)
FILE(TAPE01, BT=C, RT=F, MRL=160, SDS=YES)
FILE(TAPE02, BT=C, RT=U, MRL=280, SDS=YES)
REWIND(TAPE01)
FTN.
LDSET(FILES=TAPE01/TAPE02)
LGO.
DISPOSE(TAPE02, *PE)
7/8/9
flag card

(card input (copied to tape file TAPE01 by COPYBR control card))

flag card

PROGRAM CDPNT (TAPE02=66, TAPE01=66, TAPE2=TAPE02, TAPE1=TAPE01)
DIMENSION WS1(20), WS2(32), ARRAY(16)
CALL XFILE(1, WS1, *(FT=C, USE=R), 20)
CALL XFILE(2, WS2, *(FT=P, USE=W, FMT=A), 32)
100 CALL XREAD(WS1, ARRAY, *(XROA80), STAT)
IF (STAT .LT. 0) CALL EXIT
CALL XWRITE(WS2, ARRAY, *(A1, X0X10, A79), STAT)
GO TO 100
END

7/8/9
6/7/8/9

See section 6, System Interface, for discussion of FILE, LDSET and DISPOSE control cards as used in this example.

Note that in a KRONOS 2.1 job, an account card will follow the jobcard.
This section describes a variety of system interfaces that occur with Record Manager, and CYBER LOADER when extended character set (ASCII and EBCDIC) files are processed by 8-bit subroutines.

OPERATING SYSTEM INTERFACE

The central memory requirements for jobs involving 8-bit subroutines are specified on the job control card. Usage of all 8-bit subroutines will require an additional 14K (octal) words if all modules are referenced. For SCOPE 3.4 nine-track tape requirements also should be given in the nT parameter on the job card; for KRONOS, requirements are given on a RESOURC card.

TAPE FILES

ASCII or EBCDIC on either seven or nine-track IBM sequential tapes can be processed by the 8-bit subroutines. Seven-track tapes are requested for the job in the manner described in the KRONOS 2.1 and SCOPE 3.4 Reference Manuals.

Nine-track tape files to be processed by 8-bit subroutines or by the COPY8P program should be made available to the job by the REQUEST card.

The programmer should be fully familiar with the REQUEST control card; a complete description of the nine-track tape parameters can be found in the KRONOS 2.1 and SCOPE 3.4 Reference Manuals.

All code conversion parameters should be omitted from nine-track requests for processing by 8-bit subroutines, since conversion to or from ASCII or EBCDIC is done by the 8-bit subroutines. If either US or EB is given, it will be overridden by the CM=NO parameter on the Record Manager FILE card.

Input tape files can be copied to disk or other devices that can be substituted for tapes, provided the data format is identical to that of IBM tapes when they are read by Record Manager.
CARD FILES

All card files input to the 8-bit subroutines must be in free-form binary format. This format is fully described in appendix E.

The deck setup of such a job would be:

Card output files will be punched in free-form binary format. At the end of a run that produces card output, the disposition code P8 must be specified on the DISPOSE control card. The card is written in one of the following forms:

SCOPE 3.4  
DISPOSE,lfn,P8.

KRONOS 2.1  
DISPOSE,lfn=P8.

DISPOSE,lfn,*P8.

The logical file name (lfn) is the name of the file containing the card images to be punched. In SCOPE 3.4, the asterisk in the *P8 parameter indicates the file is to be disposed at end-of-job rather than when the control card is encountered in the job.
PRINT FILES

When an extended character set output file produced by COPY8P or the 8-bit subroutines is to be printed with full upper and lower case characters, the file must be directed to the CDC 595-6 printer through use of the DISPOSE control card written in one of the following forms:

SCOPE 3.4
DISPOSE,lnn,PE.
DISPOSE,lnn,*PE.

The logical file name (lnn) is the name of the file to be printed. The asterisk in the *PE parameter indicates the file is to be disposed at end-of-job, rather than when the control card is encountered.

KRONOS 2.1 does not support the CDC 595-6 printer.

RECORD MANAGER INTERFACE

Since Record Manager cannot block or deblock records in IBM format, it is necessary to include a FILE card specifying a transparent mode of operation of Record Manager for each IBM 8-bit sequential tape file to be processed by the 8-bit subroutines (excluding the COPY8P program). This card is written:

FILE(lnn,parameter string)

Parameters on the FILE card provide record and block information which is used to update the file information table (FIT) when a given ASCII or EBCDIC character file is opened the first time in the job. Refer to the Record Manager Reference Manual for complete information on this card.

Under SCOPE 3.4, the loader control card LDSET should be included to list each file for which a FILE card appears in the job deck. See Loader Interface in this section.

TAPE FILES

The parameters required on the FILE card for tape files are:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blocking Type</td>
<td>BT=K</td>
</tr>
<tr>
<td>Record Type</td>
<td>RT=U</td>
</tr>
<tr>
<td>Records/block</td>
<td>RB=I</td>
</tr>
<tr>
<td>Maximum Block Size</td>
<td>MBL=nnn</td>
</tr>
<tr>
<td>Maximum Record Length</td>
<td>MRL=nnn</td>
</tr>
<tr>
<td>Conversion Mode</td>
<td>CM=NO</td>
</tr>
</tbody>
</table>

\[
nnn = \left\lceil \frac{\text{BLKSIZE} \times 8}{6} \right\rceil \quad \text{†}, \quad \text{since BLKSIZE refers to 8-bit characters, and the Record Manager expects sizes in terms of 6-bit characters}
\]

† The notation \( \left\lceil m \right\rceil \) indicates that the expression, \( m \), is evaluated as the smallest integer greater than the real value of \( m \). A fraction is always rounded up to the next higher integer.
For 9-track tape input/output two additional parameters are necessary for noise record skipping:

Minimum Record Length \( \text{MNR} = 24 \)

Minimum Block Length \( \text{MNB} = 24 \)

For a COBOL calling program, the clause BLOCK CONTAINS nnnn CHARACTERS should be included with each 8-bit file description; otherwise, Record Manager may diagnose an inadequate BFS specification.

**CARD FILES**

The user must specify the following information to Record Manager as FILE card parameters to permit reading or writing of card image files:

\[
\begin{align*}
\text{BT} &= C \\
\text{RT} &= F \\
\text{MRL} &= 160
\end{align*}
\]

**PRINT FILES**

For Record Manager handling of files formatted for the printer, the following FILE card parameters must be given:

\[
\begin{align*}
\text{BT} &= C \\
\text{RT} &= U \\
\text{MRL} &= 280
\end{align*}
\]

**LOADER INTERFACE**

The loader directive LDSET is used to control the load process under a variety of conditions. When Record Manager is to process tape, card and print files, and the FILE control card is present in the job deck, the LDSET card ensures that library programs are loaded for processing specified files. In this case, the LDSET card is written:

\[
\text{LDSET} (\text{FILES} = l\text{fn}_1, l\text{fn}_2, \ldots)
\]

The file names on the card refer to the tape, print, or card files used in a given run of the job to which a FILE card pertains.

**OMITTING UNNEEDED 8-BIT MODULES**

If not all features of the 8-bit subroutines package are used, some 8-bit modules may be omitted from the program being loaded to conserve space in the field length. The OMIT option on the LDSET card is written:

\[
\text{LDSET} (\text{OMIT} = \text{name/name} \ldots)
\]
The names given are entry points of the modules to be omitted from the load. Because the names given below contain the special character, period, they must be surrounded by dollar-sign characters in the name list as follows:

```
LDSET(OMIT=$T8.TSTC$/T8.TSTT$)
```

<table>
<thead>
<tr>
<th>Entry Name</th>
<th>Module Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>T8.TST6</td>
<td>Selector Expressions on Internal Data</td>
</tr>
<tr>
<td>T8.TSTT</td>
<td>Selector Expressions on Tape Data</td>
</tr>
<tr>
<td>T8.TSTC</td>
<td>Selector Expressions on Card Data</td>
</tr>
<tr>
<td>T8.CN6T</td>
<td>Conversion, Internal to Tape</td>
</tr>
<tr>
<td>T8.CN6C</td>
<td>Conversion, Internal to Card</td>
</tr>
<tr>
<td>T8.CN6P</td>
<td>Conversion, Internal to Print</td>
</tr>
<tr>
<td>T8.CNT6</td>
<td>Conversion, Tape to Internal</td>
</tr>
<tr>
<td>T8.CNC6</td>
<td>Conversion, Card to Internal</td>
</tr>
</tbody>
</table>

**LOADER CONSIDERATIONS**

The XREAD and XWRITE routines do not reference any of the above entry point names; rather, they depend on XFILE to place an appropriate address in the working storage area. The listed entry names are referenced only by XFILE.

In an overlay situation, it may be desirable to place XFILE in a different overlay from XREAD or XWRITE; but it is important to place the working code into a module common to all, as they share subroutines and some common areas. The LOADER Reference Manual contains instructions for this procedure as well as for overlay and segment loading.
It is assumed that the user of the 8-bit subroutines is familiar with COBOL and has access to the IBM 360/370 COBOL Language and COBOL Programmer's Guide publications. This section is intended to clarify the usage of 8-bit subroutines in connection with IBM 360/370 compatible tape files created by or to be read by an IBM 360/370 COBOL program. The examples assume that the record description for the file used in the IBM COBOL and the CDC COBOL program is established. The writing of a conversion string for the reading and converting of an IBM COBOL tape file to internal CDC format will be demonstrated.

A tape file created by an IBM COBOL program may contain any combination of the following data types: (The letters in parentheses are the T values given in table 2-1.)

- 8-bit characters (X)
- Half-word integers, 16 bits (H)
- Whole word integers, 32 bits (W)
- Double word integers, 64 bits (G)
- 32-bit floating point (F)
- 64-bit floating point (L)
- Packed decimal (internal decimal) (P)
- Decimal signed numeric (external decimal) (S)

An IBM COBOL program cannot create 128-bit extended precision floating point data. Any of the above data types also can be considered to be bit string data (B). To select the appropriate m value for each of the data types, refer to table 2-1. A complete description of IBM data type formats is given in appendix C.
The following table shows the relationship between IBM COBOL USAGE clauses, picture clauses IBM byte storage allocation, and the corresponding 8-bit Tm values to be used in converting such data types.

<table>
<thead>
<tr>
<th>IBM COBOL USAGE Category</th>
<th>Picture Format</th>
<th>IBM Boundary Alignment</th>
<th>T</th>
<th>IBM 8-Bit Bytes Used (m derived from this)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISPLAY</td>
<td>1. alphabetic</td>
<td>none</td>
<td>X</td>
<td>1 per character or digit (except for V in external floating point) 18 digit limit</td>
</tr>
<tr>
<td></td>
<td>2. alphanumeric</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3. report format</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>4. external floating point form</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DISPLAY</td>
<td>numeric unsigned</td>
<td>none</td>
<td>N or X</td>
<td>1 per digit, limit 18</td>
</tr>
<tr>
<td>DISPLAY</td>
<td>numeric signed</td>
<td>none</td>
<td>S</td>
<td>1 per digit, limit 18</td>
</tr>
<tr>
<td>COMP (binary)</td>
<td>1 to 4 digits</td>
<td>halfword</td>
<td>H</td>
<td>2</td>
</tr>
<tr>
<td>COMP (binary)</td>
<td>5 to 9 digits</td>
<td>fullword</td>
<td>W</td>
<td>4</td>
</tr>
<tr>
<td>COMP (binary)</td>
<td>10 to 18 digits</td>
<td>fullword</td>
<td>G</td>
<td>8</td>
</tr>
<tr>
<td>COMP-1 (internal floating point)</td>
<td>fullword</td>
<td>F</td>
<td>4 (short precision)</td>
<td></td>
</tr>
<tr>
<td>COMP-2 (internal floating point)</td>
<td>double word</td>
<td>L</td>
<td>8 (long precision)</td>
<td></td>
</tr>
<tr>
<td>COMP-3 (packed decimal)</td>
<td>numeric unsigned or signed</td>
<td>none</td>
<td>P</td>
<td>1 byte per 2 digits plus 1 byte for low order digit and sign</td>
</tr>
</tbody>
</table>

If IBM computational items are mixed with other elementary items in the data record description, slack bytes may be present on the IBM file to assure the proper byte alignment for each COMPUTATIONAL, COMPUTATIONAL-1, and COMPUTATIONAL-2 item. For instance, the byte address of the first byte of a half-word binary item must be divisible by 2; the byte address of a full or double-word binary item must be divisible by 4; the byte address of a COMPUTATIONAL-1 item must be divisible by 4; the address of a COMPUTATIONAL-2 item must be divisible by 8.

The following examples in figure 7-1 show the relationship between some IBM COBOL elementary data description entries and the corresponding 8-bit Tm values. Table 7-1 was used to derive the Tm value. If no usage clause is specified, usage is assumed to be DISPLAY.
<table>
<thead>
<tr>
<th>Case</th>
<th>Elementary Data Entry</th>
<th>USAGE Category</th>
<th>Tm</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01 EG1 PIC A(20).</td>
<td>DISPLAY-alphabetic</td>
<td>X20</td>
</tr>
<tr>
<td>2</td>
<td>01 EG2 PIC X(53).</td>
<td>DISPLAY-alphanumeric</td>
<td>X53</td>
</tr>
<tr>
<td>3</td>
<td>01 EG3 PIC $999,999.99-</td>
<td>DISPLAY-report form</td>
<td>X12</td>
</tr>
<tr>
<td>4</td>
<td>01 EG4 PIC +.9(8)E+99.</td>
<td>DISPLAY-external floating</td>
<td>X14</td>
</tr>
<tr>
<td></td>
<td></td>
<td>point form</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>01 EG5 PIC 9(8)V99.</td>
<td>DISPLAY-numeric unsigned</td>
<td>N10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(or X10)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>01 EG6 PIC S9(8)V99.</td>
<td>DISPLAY-numeric signed</td>
<td>S10</td>
</tr>
<tr>
<td>7</td>
<td>01 EG7 PIC S9(3)V9 COMPUTATIONAL.</td>
<td>COMP, 1-4 digits</td>
<td>H</td>
</tr>
<tr>
<td>8</td>
<td>01 EG8 PIC 9(6) COMPUTATIONAL.</td>
<td>COMP, 5-9 digits</td>
<td>W</td>
</tr>
<tr>
<td>9</td>
<td>01 EG9 PIC 9(8)V9(9) COMPUTATIONAL.</td>
<td>COMP, 10-18 digits</td>
<td>G</td>
</tr>
<tr>
<td>10</td>
<td>01 EG10 COMPUTATIONAL-1.</td>
<td>COMP-1</td>
<td>F</td>
</tr>
<tr>
<td>11</td>
<td>01 EG11 COMPUTATIONAL-2.</td>
<td>COMP-2</td>
<td>L</td>
</tr>
<tr>
<td>12</td>
<td>01 EG12 PIC S9(5)V9(6) COMP-3.</td>
<td>COMP-3</td>
<td>P6</td>
</tr>
<tr>
<td>13</td>
<td>01 EG13 PIC 9(4)V9(4) COMP-3.</td>
<td>COMP3</td>
<td>P5</td>
</tr>
</tbody>
</table>

Figure 7-1. IBM Examples

In each case, m is the size of the data item in 8-bit bytes. The m must be omitted for COMP, COMP-1 and COMP-2 data items. Also, calculation of m for COMP-3 data is not as straight-forward as for other data types, since COMP-3 data is packed 2 digits per 8-bit byte. The sign and the low order digit appear together in the rightmost byte. To calculate the m value, use the following algorithm:

Set D as the number of digits specified in the PICTURE clause. (For EG12, set D as 11; for EG13, set D as 8.) The examples given are lines 12 and 13 from figure 7-1.

Obtain E by adding 1 to D to allow for the sign half-byte which always appears in the right half of the rightmost byte along with the low order digit. (For EG12, E is 12; for EG13, E is 9.)

If E is odd, set F to E + 1. (For EG13, F is 10.) If E is even, set F to E. (For EG12, F is 12.)

Divide F by 2 to obtain the proper value for m; therefore, for EG12, m is 6; for EG13, m is 5. m is the number of 8-bit bytes occupied by the packed decimal data field on the IBM tape.
INTERNAL CDC COBOL DATA FORMATS

An internal CDC data record created by a CDC COBOL program in conjunction with the 8-bit subroutines may contain any combination of the following data types:

6-bit characters (X)
12-bit ASCII characters (A)
12-bit EBCDIC characters (C)
unnormalized floating point (60 bit) (U)
single precision floating point (60 bit) (E)
double precision floating point (120 bit) (D)
numeric display signed overpunch (S)
numeric display, unsigned (N)
numeric display, leading zeros suppressed (Z)

The letters in parentheses are the corresponding T values as described in table 2-1 under Internal Media. The A and C values for T are unique to 8-bit subroutine processing. A CDC 6000 COBOL program cannot create or process 60-bit integer data. Any of the above data types can be considered as bit string data (B). To choose the corresponding m value for the conversion items, refer to the right half of table 2-1. A more complete description of the CDC internal format types appears in appendix D.

Table 7-2 shows the relationship between CDC COBOL USAGE clauses, PICTURE clauses, CDC storage allocation, and the corresponding 8-bit Tm values used in processing such data types.
Table 7-2. CDC COBOL — Tm Values

<table>
<thead>
<tr>
<th>CDC COBOL Usage</th>
<th>Picture Format</th>
<th>Boundary Alignment</th>
<th>T</th>
<th>CDC 6-Bit Bytes Used in Storage Allocation (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISPLAY</td>
<td>1. alphabetic</td>
<td>none</td>
<td>X</td>
<td>1 per digit or character (except for V)</td>
</tr>
<tr>
<td></td>
<td>2. alphanumeric</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3. edited report form</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>except leading zeros suppressed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMPUTATIONAL or DISPLAY</td>
<td>numeric unsigned</td>
<td>none</td>
<td>N or X</td>
<td>1 per digit — limit 18</td>
</tr>
<tr>
<td>COMPUTATIONAL or DISPLAY</td>
<td>signed numeric</td>
<td>none</td>
<td>S</td>
<td>1 per digit — limit 18</td>
</tr>
<tr>
<td>DISPLAY</td>
<td>alphanumeric</td>
<td>none</td>
<td>A</td>
<td>2 per digit†</td>
</tr>
<tr>
<td></td>
<td>12-bit ASCII</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DISPLAY</td>
<td>alphanumeric</td>
<td>none</td>
<td>C</td>
<td>2 per digit†</td>
</tr>
<tr>
<td></td>
<td>12-bit EBCDIC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DISPLAY</td>
<td>edited, leading zeros suppressed</td>
<td>none</td>
<td>Z</td>
<td>1 per digit or space 18 digit limit</td>
</tr>
<tr>
<td>COMP-1</td>
<td>1 to 14 digits</td>
<td>word</td>
<td>U</td>
<td>10</td>
</tr>
<tr>
<td>COMP-1</td>
<td>15 to 18 digits</td>
<td>word</td>
<td>D</td>
<td>20</td>
</tr>
<tr>
<td>COMP-2</td>
<td>1 to 14 digits</td>
<td>word</td>
<td>E</td>
<td>10</td>
</tr>
</tbody>
</table>

†For A and C types, m = half the number of 6-bit bytes used.

If CDC COMP-1 or COMP-2 data items are mixed with other data types in the data record description, slack bytes may be present in the CDC storage of the record to assure word alignment for each COMP-1 or COMP-2 item.
The examples in figure 7-2 show the relationship between some CDC COBOL elementary data description entries and the corresponding 8-bit \( T_m \) values derived by using table 7-2. USAGE is DISPLAY unless specified otherwise.

<table>
<thead>
<tr>
<th>Case</th>
<th>Elementary Data Entry</th>
<th>USAGE Category</th>
<th>( T_m )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01 AG1 PIC A(20).</td>
<td>DISPLAY-alphabetic</td>
<td>X20</td>
</tr>
<tr>
<td>2</td>
<td>01 AG2 PIC X(53).</td>
<td>DISPLAY-alphanumeric</td>
<td>X53</td>
</tr>
<tr>
<td>3</td>
<td>01 AG3 PIC $999,999.99-</td>
<td>DISPLAY-report form</td>
<td>X12</td>
</tr>
<tr>
<td>4</td>
<td>01 AG4 PIC 9(8)V99.</td>
<td>DISPLAY-numeric</td>
<td>N10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>signed</td>
<td>(or X10)</td>
</tr>
<tr>
<td>5</td>
<td>01 AG5 PIC S9(8)V99.</td>
<td>DISPLAY-numeric</td>
<td>S10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>signed</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>01 AG6 PIC X(100).</td>
<td>DISPLAY-12-bit ASCII</td>
<td>A50</td>
</tr>
<tr>
<td>7</td>
<td>01 AG7 PIC X(100).</td>
<td>DISPLAY-12-bit EBCDIC</td>
<td>C50</td>
</tr>
<tr>
<td>8</td>
<td>01 AG8 PIC Z(9).</td>
<td>DISPLAY-leading zeros</td>
<td>Z9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>suppressed</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>01 AG9 PIC S9(10) USAGE COMP-1.</td>
<td>COMP-1, 1-14 digits</td>
<td>U</td>
</tr>
<tr>
<td>10</td>
<td>01 AG10 PIC S9(16) USAGE COMP-1.</td>
<td>COMP-1, 15-18 digits</td>
<td>D</td>
</tr>
<tr>
<td>11</td>
<td>01 AG11 USAGE COMPUTATIONAL-2.</td>
<td>COMP-2</td>
<td>E</td>
</tr>
</tbody>
</table>

Figure 7-2. CDC Examples

Except for cases 6 and 7, \( m \) is the size of the data item in 6-bit bytes. For 12-bit ASCII or EBCDIC internal CDC data items, \( m \) is the size of the data item in 12-bit bytes; \( m \) must be omitted for COMP-1 and COMP-2 data items.

EXAMPLE CONVERSIONS — IBM COBOL FILES TO INTERNAL CDC FILES

The following examples show IBM COBOL data record descriptions with corresponding CDC COBOL data record descriptions. A set of conversion strings appears after each set of record descriptions. The XREAD conversion strings can be used to read and convert the IBM file to internal CDC format. The XWRITE conversion strings can be used to write and convert a CDC file to IBM format.
Case 1. Simple Display Item Conversions:

IBM data record and CDC data record:

```
01 RECORD.
  02 RA1 PIC X{5}.
  02 RA2 PIC X{6}.
  02 RA3 PIC 9{8}.
  02 RA3A PIC X{8} REDEFINES RA3.
  02 RA4 PIC S9{5}V99.
  02 RA5 PIC 9{4}V9{3}.
```

XREAD Conversion string (IBM tape file converted to CDC internal format):

```
{X5X5, X6X6, N8N8, S7S7, N7N7}
```

XREAD equivalent conversion strings:

```
{X5, X6, N8, S7, N7} (omitting Tm2's)
{X5, X6, X8, S7, X7}
{X19, S7, X7}
```

XWRITE conversion strings:

Same as read conversion strings for this case.

REDEFINES items present a problem if they require a different type of conversion from the redefined area. See case 4 for an illustration of how to use selector expressions in the conversion string to process REDEFINES areas.
Case 2. Display, Binary and Packed Decimal Data with Occurrences:

IBM data record:

    01 RECORD2.
    02 RB1 OCCURS 5 TIMES PIC S9{8}.
    02 RB2 OCCURS 10 TIMES.
    03 RB3 PIC S9{5} COMPUTATIONAL.
    03 RB4 PIC XX.
    02 RB5 PIC S9{13}V99 COMPUTATIONAL-3.

CDC data record:

    01 RECORD2.
    02 RB1 OCCURS 5 TIMES PIC S9{8}.
    02 RB2 OCCURS 10 TIMES.
    03 RB3 PIC S9{5}.
    03 RB4 PIC XX.
    02 RB5 PIC S9{13}V99.

XREAD conversion string (IBM tape file converted to CDC internal format):

    {S8S8, 10{SS5, X2X2, X2X01, P8S15} }

The IBM COBOL compiler inserts two slack bytes after RB4 for storage allocation to ensure each occurrence of RB3 begins on a full-word boundary. The X2X0 item in the conversion string causes these slack bytes to be skipped.

XREAD equivalent conversion strings:

    {S8S8, 10{SS5, X2X2, X2X01, P8S15} }  
    or
    {S8S8, 10{SS5, X4X21, P8S15} }

XWRITE conversion strings (CDC internal format converted to IBM tape file):

    {SS8S8, 10{SS5W, X2X2, X0X21, S15P8} }  
    or
    {SS8, 10{SS5W, X2X41, S15P8} }

In XWRITE conversion strings, the Tm1 values are interchanged with the Tm2 values in each single conversion item.
Case 3.

IBM data record description:

01 RECORD3.

  02 DIN OCCURS 3 TIMES.
    03 DID PIC 9 USAGE COMP-3.
    03 DID1 PIC 9{5} USAGE COMP-3.
    02 IRP PIC S9{9}V99 USAGE COMP-3.
    02 IRP2 PIC S9V99 COMPUTATIONAL.
    02 IRP3 PIC 9{15} COMPUTATIONAL.
    02 IRP4 USAGE COMPUTATIONAL-1.
    02 IRP5 USAGE COMPUTATIONAL-2.

CDC COBOL record description:

01 RECORD3.

  02 DIN OCCURS 3 TIMES.
    03 DID PIC 9.
    03 DID1 PIC 9{5}.
    02 IRP PIC S9{9}V99.
    02 IRP2 PIC S9V99.
    02 IRP3 PIC 9{15}.
    02 IRP4 PIC S9{14} COMP-1.
    02 IRP5 PIC S9{18} COMP-1.

XREAD conversion string (IBM tape file converted to internal CDC):

{3PLN1, P3N5}, P6S11, HS3, GN15, FE, LD}

XWRITE conversion string (internal CDC converted to IBM tape file):

{3N1P1, N5P3}, S11P6, S3H, N15G, EF, DL}
Case 4:

IBM data record description:

01 RECORDA.
   02 FIXED-FIELD PIC X{40}.
   02 KEY-FIELD PIC 9.
   02 ALT1.
      03 A11 PIC S9{5}V99 COMP-3 OCCURS 5 TIMES.
      03 A12 TOT1 PIC S9{10}.
   02 ALT2 REDEFINES ALT1.
      03 A21 PIC X{5} OCCURS 6 TIMES.
   02 ALT3 REDEFINES ALT2.
      03 A31 PIC 999 COMPUTATIONAL OCCURS 10 TIMES.
      03 A32 PIC S9{10}.

CDC data record description:

01 RECORD A.
   02 FIXED-FIELD PIC X{40}.
   02 KEY-FIELD PIC 9.
   02 ALT1.
      03 A11 PIC S9{5}V99 OCCURS 5 TIMES.
      03 A12 TOT1 PIC S9{10}.
   02 ALT2 REDEFINES ALT1.
      03 A21 PIC X{5} OCCURS 6.
   02 ALT3 REDEFINES ALT2.
      03 A31 PIC 999 OCCURS 10 TIMES.
      03 A32 PIC S9{10}.
The REDEFINES clauses in this example require different treatment in the formation of the conversion string. Assume, for the above example, that KEY-FIELD determines the format of the remaining bytes of the record. If KEY-FIELD is 1, the remainder of the record conforms to ALT1 formatting. If it is 2, the remainder of the record conforms with ALT2. If it is 3, the remainder of the record conforms with ALT3. The selector expression in the following conversion string determines which of the three possible formats to use in converting the IBM input data.

XREAD conversion string (IBM file converted to CDC internal):

```
{X40X40, N1 EQ 1: N1N1, 5P4S7, S10S10;}
N1 EQ 2: N1N1, 6X5X5;
N1 EQ 3: N1N1, 10HN3, S10S10}
```

An equivalent conversion string is:

```
{X40, N1 EQ 1: N1, 5P4S7, S10;}
N1 EQ 2: X31;
N1 EQ 3: N1, 10HN3, S10}
```

XWRITE conversion string (CDC internal converted to IBM file):

```
{X40X40, N1 EQ 1: N1N1, 5S7P4, S10S10;}
N1 EQ 2: N1N1, 6X5X5;
N1 EQ 3: N1N1, 10N3H, S10S10}
```

Case 5:

IBM data record description:

```
01 TRANS-HOLD-IN.

05 DEALER-INVOICE-NO.

10 DEALER-ID PICTURE 9 VALUE 0 USAGE COMP-3.

10 DEALER-ID1 PICTURE 9{5} VALUE 0 USAGE COMP-3.

05 CRV-NO PICTURE 9{6} VALUE 0 USAGE COMP-3.

05 PURCHASE-DATE.

10 PURCHASE-YEAR PICTURE 9 VALUE ZEROS.

10 PURCHASE-DAY PICTURE 999 VALUE ZEROS.
```
05 MATURITY-DATE.
   10 MATURITY-YEAR PICTURE 9 VALUE ZEROS.
   10 MATURITY-DAY PICTURE 999 VALUE ZEROS.
05 DAYS-TO-MATURITY PICTURE 99V9(4) VALUE ZEROS USAGE COMP-3.
05 INTEREST-RATE PICTURE 99V9(4) VALUE ZEROS USAGE COMP-3.
05 FACE-AMOUNT PICTURE 9{9}V99 VALUE ZEROS USAGE COMP-3.
05 DISCOUNT-AMOUNT PICTURE 9{8}V99 VALUE ZEROS USAGE COMP-3.
05 NET-PROCEEDS PICTURE 9{9}V99 VALUE ZEROS USAGE COMP-3.
05 COMMISSION-AMOUNT PICTURE 9{5}V99 VALUE ZEROS USAGE COMP-3.
05 WEIGHTED-GROSS-SALES PICTURE 9{12}V99 VALUE ZEROS USAGE COMP-3.
05 WEIGHTED-NET-PROCEEDS PICTURE 9{12}V99 VALUE ZERO USAGE COMP-3.
05 DAILY-DISCOUNT-AMOUNT PICTURE 9{8}V9{5} VALUE ZERO USAGE COMP-3.
05 ADVANCE-PURCHASE-TRAN PICTURE X VALUE SPACE.
05 FILLER PICTURE X{8} VALUE SPACES.
05 FILLER PICTURE X{4} VALUE SPACES.
CDC data record description:

01 TRANS-HOLD-IN.
   05 DEALER-INVOICE-No.
      10 DEALER-ID PIC 9.
      10 DEALER-ID1 PIC 9{5}.
   05 CRV-No PIC 9{6}.
   05 PURCHASE-DATE.
      10 PURCHASE-YEAR PIC 9.
      10 PURCHASE-DAY PIC 999.
   05 MATURITY-DATE.
      10 MATURITY-YEAR PIC 9.
      10 MATURITY-DAY PIC 999.
   05 DAYS-TO-MATURITY PIC 999.
   05 INTEREST-RATE PIC 99V9999.
   05 FACE-AMOUNT PIC 9{9}V99.
   05 DISCOUNT-AMOUNT PIC 9{8}V99.
   05 NET-PROCEEDS PIC 9{9}V99.
   05 COMMISSION-AMOUNT PIC 9{5}V99.
   05 WEIGHTED-GROSS-SALES PIC 9{12}V99.
   05 WEIGHTED-NET-PROCEEDS PIC 9{12}V99.
   05 DAILY-DISCOUNT-AMOUNT PIC 9{8}V9{5}.
   05 ADVANCE-PURCHASE-TRAN PIC X.
   05 FILLER PIC X{8}.
   05 FILLER PIC X{4}.
XREAD conversion string (IBM tape file converted to CDC internal):

\{P1N1, P3N5, P4N6, N1, N3, N1, N3, N3, P2N3, P4N6, P6N11, P6N10, P6N11, P4N7, P8N14, P8N14, P7N13, X1, X8, X4\}

or

\{P1N1, P3N5, P4N6, X8, P2N3, P4N6, P6N11, P6N10, P6N11, P4N7, P8N14, P8N14, P7N13, X13\}

XWRITE conversion string (CDC internal converted to IBM tape file):

\{N1P1, N5P3, N6P4, N1, N3, N1, N3, N3, N3P2, N6P4, N11P6, N10P6, N11P6, N7P4, N14P8, N13P7, X1, X8, X4\}

or

\{N1P1, N5P3, N6P4, X8, N3P2, N6P4, N11P6, N10P6, N11P6, N7P4, N14P8, N14P8, N13P7, X13\}

Case 6. Conversion String Maintaining 8-Bit Significance for Non-Numeric Characters

Assume the IBM tape file is written in ASCII format.

IBM data record description:

01 RECB.

02 NAME PIC X{20}.

02 ID PIC X{10}.

02 ITMS OCCURS 5 TIMES.

03 QUANTITY PIC 9999.

03 ITEM PIC X{5}.

03 PRICE PIC S9{4}.99.

02 TOT PIC S9{9}.99.
CDC data record description:

01 RECL.
   02 NAME PIC X(40).
   02 ID PIC X(20).
   02 ITMS OCCURS 5 TIMES.
      03 QUANTITY PIC 9999.
      03 ITEM PIC X(10).
      03 PRICE PIC $9(4).99.
   02 TOT PIC $9(9).99.

Data-names NAME, ID, and ITEM will be converted with 8-bit significance maintained. For these items, the CDC COBOL record description allows two 6-bit characters for each 8-bit character in storage allocation; each 8-bit character will occupy 12 bits.

XREAD conversion string (IBM tape file converted to internal CDC):

{X20A20, X10A10, 5{N4N4, X5A5, X8X8}, X13X13}

or

{X20A20, X10A10, 5{N4, X5A5, X8}, X13}

XWRITE conversion string (internal CDC converted to IBM tape file):

{A20X20, A10X10, 5{N4, A5X5, X8, X13}

SAMPLE CDC COBOL JOB

The following sample CDC 6000 COBOL program processes an IBM EBCDIC tape containing F-type records that conform to the case 5 format in the previous examples. The program will read and display each record on the tape. Code could be added to the COBOL program to save each converted record on a CDC tape file.

job card
COBOL.
VSN(TAPE02=135) NINE TRACK INPUT TAPE
REQUEST,TAPE02,NT,S.
FILE(TAPE02,BT=K,RT=U,NBL=107,MLR=107,CM=NO,IMN=24,IMN=24)
LDSET.FILES=TAPE02
LGO.
7/8/9
IDENTIFICATION DIVISION.
PROGRAM-ID. RC9T.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. 6600.
OBJECT-COMPUTER. 6600.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT TRANS-HOL ASSIGN TO TAPE02.
DATA DIVISION.
FILE SECTION.
FD TRANS-HOL
   RECORDING MODE IS DECIMAL
   LABEL RECORDS ARE OMITTED
   BLOCK CONTAINS 107 CHARACTERS
   DATA RECORD IS TRANS-HOLD-IN.
01 TRANS-HOLD-IN.
   05 DEALER-INVOICE-NO.
      10 DEALER-ID PIC 9.
      10 DEALER-ID1 PIC 9(5).
   05 CRV-NO PIC 9(6).
   05 PURCHASE-DATE.
      10 PURCHASE-YEAR PIC 9.
      10 PURCHASE-DAY PIC 999.
   05 MATURITY-DATE.
      10 MATURITY-YEAR PIC 9.
      10 MATURITY-DAY PIC 999.
   05 DAYS-TO-MATURITY PIC 999.
   05 FACE-AMOUNT PIC 9(9)V99.
   05 INTEREST-RATE PIC 99V9999.
   05 DISCOUNT-AMOUNT PIC 9(8)V99.
   05 NET-PROCEEDS PIC 9(9)V99.
   05 COMMISSION-AMOUNT PIC 9(5)V99.
   05 WEIGHTED-GROSS-SALES PIC 9(12)V99.
   05 WEIGHTED-NET-PROCEEDS PIC 9(12)V99.
   05 DAILY-DISCOUNT-AMOUNT PIC 9(8)V9(5).
   05 ADVANCE-PURCHASE-TRAN PIC X.
   05 FILLER PIC X(12).
WORKING-STORAGE SECTION.
   77 SZ PIC 99 USAGE COMP-1 VALUE 17.
   77 STAT USAGE COMP-2.
   01 WS1 PIC X(170).
   01 CSTRING PIC X(100) VALUE
      ≠(P1N1,P3N5,P4N6,X8,P2N3,P4N6,P6N11,P6N10,P6N11,  
      ≠P4N7,P8N14,P8N14,P7N13,X13)≠.
   01 FSTRING PIC X(50) VALUE
      ≠(FT=T,USE=R,RECP1=F,BLKSIZE=80,CODE=C)≠.
PROCEDURE DIVISION.
PARA1.
    OPEN INPUT TRANS-HOL WITH NO REWIND.
    ENTER XFILE USING TRANS-HOL, WSL, FSTRING, SZ.
RLOOP.
    ENTER XREAD USING WSL, TRANS-HOLD-IN, CSTRING, STAT.
        IF STAT = 1 DISPLAY 'BAD INPUT DATA'
        GO TO RLOOP.
    IF STAT IS LESS THAN 0 GO TO PROCESS-EOF.
    IF STAT NOT EQUAL 0 DISPLAY 'TAPE READ ERROR'.
    DISPLAY TRANS-HOLD-IN.
    GO TO RLOOP.
PROCESS-EOF.
    DISPLAY 'EOF ENCONTERED ON TAPE READ'.
    CLOSE TRANS-HOL.
    STOP RUN.
6/7/8/9

The following sample COBOL job reads 200 logical records from the second file of an IBM 360 EBCDIC tape. Logical blocks are 135 characters long and are blocked into 3105 character blocks containing 23 records per block. The tape file contains both upper and lower case character data. Each record is converted to display code with lower case data folded to its upper case equivalent, then displayed upon output.
IDENTIFICATION DIVISION.
PROGRAM-ID. LIST.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER 6600.
OBJECT-COMPUTER 6600.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
SELECT FILE1 ASSIGN TO TAPE01.
DATA DIVISION.
FILE SECTION.
FD FILE1.
LABEL RECORDS ARE OMITTED.
BLOCK CONTAINS 140 CHARACTERS.
DATA RECORDS ARE REC1.
WORKING-STORAGE SECTION.
01 WKSP PIC X(4200).
01 STAT COMP=2.
01 CNT PIC 999 VALUE 0.

PROCEDURE DIVISION.
PROCESS=DATA.
ENTFR FILE USING FILE1, WKSP,
  *(FMT=USE#R,RECFM=FB,HLKSIZE=3105,LRECL=135).
READ-PRINT-LOOP.
  ADD 1 TO CNT.
  IF CNT IS GREATER THAN 200 STOP RUN.
  ENTFR READ USING WKSP, REC1, *(A(135)) STAT.  
  IF STAT NOT EQUAL 0.
    DISPLAY #BAD RECORD, STAT = *, STAT, REC1.
    GO TO READ-PRINT-LOOP.
  DISPLAY REC1.
  GO TO READ-PRINT-LOOP.
COMPASS/7IMS A ANY ANY *COMPASS IMS

02/22/7100 UU

E 21 PII # CUC(11A ROSEMARIE) A

COMPASS/7 IMS A 9 5 6 12 93 F 9 10 0 THE ASSEMBLY PROCESS 9 THE ASSEMBLER ASSEMBLES ALL OPERATE ON ESSENTIALLY THE SAME PRINCIPLE. THE FIRST PASS READS THE SOURCE FILE AND PERFORMS THE FOLLOWING MAIN FUNCTIONS:

1) EXAMINES EACH INSTRUCTION TO DETERMINE HOW MUCH STORAGE IS REQUIRED 11 IN THE OBJECT CODE BY THE INSTRUCTIONS. 9 910 A P (DEFFINE SYMBOLS. 9 911 WHEN SYMBOL DEFINITIONS OPERATIONS ARE REQUESTED SUCH AS BY THE APPEARANCE OF A SYMBOL IN THE LOCATION FIELD OF A MACHINE INSTRUCTION OR BY ITS USE IN SOME PSEUDO-OPERATION, THE SYMBOL IS GIVEN A DEFINITION. 9 912 THE COLLECTION OF THESE SYMBOLS AND THEIR VALUES IS KNOWN AS THE SYMOL TABLE. 9

2) EXPANDS THE VARIOUS HIGHER LEVEL OPERATIONS, SUCH AS MACROS AND DUPLICATIONS. 9 914 (ACUMULATLIS LATERALS. 9 915 AT THE CONCLU DIATIONS OF EACH INSTRUCTION IN THE 91 ASSEMBLY, 9 912, THE VALUE RELATIVE TO SOME BASE BLOCK OF EACH SYMBOL, 9 914 THE QUANTITY VALUE AND NAME OF ALL LITERAL VALUES USUALLY NEEDED IN THIS INFORMATION, A SECOND PASS OVER THE SOURCE STATEMENTS CAN FILL IN ALL LITERAL SYMBOL VALUES. LOCATE LITERAL VALUES, AND ASSIGN BLOCK VALUES. 9 916 THIS TWO-PASS PHILOSOPHY IS FOLLOWED IN COMPASS. 9

3) THE MAIN CONTROL OF PASS 1 CONSISTS OF THE FOLLOWING OPERATIONS: 9 912, READING A STATEMENT FROM THE INPUT FILE AND PREPARING THE TRANSLATE FOR LISTING OUT, 9 912, EDITING THE STATEMENT TO REMOVE MICRO AND CONCATENATION 914 IMRS, 9 9 THE SOURCE FILE AND PERFORMS THE FOLLOWING MAIN FUNCTIONS:

- LOOKING UP THE OPERATION CODE AND JUMPING TO THE APPROPRIATE PROCESSOR, 9 914, READING THE STATEMENT FROM THE INTERMEDIATE FILE ALONG WITH OTHER PERTINENT 91 INFORMATION TO BE PASSED ALONG FOR PASS 2 PROCESSORS, 9

4) THE LENGTH OF EACH INSTRUCTION IN THE 91 ASSEMBLY, 9 912, THE VALUE RELATIVE TO SOME BASE BLOCK OF EACH SYMBOL, 9 914 THE QUANTITY VALUE AND NAME OF ALL LITERAL VALUES USUALLY NEEDED IN THIS INFORMATION, A SECOND PASS OVER THE SOURCE STATEMENTS CAN FILL IN ALL LITERAL SYMBOL VALUES, LOCATE LITERAL VALUES, AND ASSIGN BLOCK VALUES. 9 916 THIS TWO-PASS PHILOSOPHY IS FOLLOWED IN COMPASS. 9

5) THE MAIN CONTROL OF PASS 1 CONSISTS OF THE FOLLOWING OPERATIONS: 9 912, READING A STATEMENT FROM THE INPUT FILE AND PREPARING THE TRANSLATE FOR LISTING OUT, 9 912, EDITING THE STATEMENT TO REMOVE MICRO AND CONCATENATION 914 IMRS, 9 9 THE SOURCE FILE AND PERFORMS THE FOLLOWING MAIN FUNCTIONS:

- LOOKING UP THE OPERATION CODE AND JUMPING TO THE APPROPRIATE PROCESSOR FOR ASSEMBLY, 9 914, WHICH 9 914, EVALUATE THE STATEMENT AS A SYMBOL AND PERFORM THE APPROPRIATE ACTIONS, 9 914, JUMPING TO THE APPROPRIATE PROCESSOR FOR ASSEMBLY, 9 914, WHICH 9 914, EVALUATE THE STATEMENT AS A SYMBOL AND PERFORM THE APPROPRIATE ACTIONS, 9 914, JUMPING TO THE APPROPRIATE PROCESSOR FOR ASSEMBLY, 9 914, WHICH 9 914, EVALUATE THE STATEMENT AS A SYMBOL AND PERFORM THE APPROPRIATE ACTIONS, 9 914, JUMPING TO THE APPROPRIATE PROCESSOR FOR ASSEMBLY, 9 914, WHICH 9 914, EVALUATE THE STATEMENT AS A SYMBOL AND PERFORM THE APPROPRIATE ACTIONS, 9 914, JUMPING TO THE APPROPRIATE PROCESSOR FOR ASSEMBLY, 9 914, WHICH 9 914, EVALUATE THE STATEMENT AS A SYMBOL AND PERFORM THE APPROPRIATE ACTIONS, 9 914, JUMPING TO THE APPROPRIATE PROCESSOR FOR ASSEMBLY, 9 914, WHICH 9 914, EVALUATE THE STATEMENT AS A SYMBOL AND PERFORM THE APPROPRIATE ACTIONS, 9 914, JUMPING TO THE APPROPRIATE PROCESSOR FOR ASSEMBLY, 9 914, WHICH 9 914, EVALUATE THE STATEMENT AS A SYMBOL AND PERFORM THE APPROPRIATE ACTIONS, 9 914, JUMPING TO THE APPROPRIATE PROCESSOR FOR ASSEMBLY, 9 914, WHICH 9 914, EVALUATE THE STATEMENT AS A SYMBOL AND PERFORM THE APPROPRIATE ACTIONS, 9 914, JUMPING TO THE APPROPRIATE PROCESSOR FOR ASSEMBLY, 9 914, WHICH 9 914, EVALUATE THE STATEMENT AS A SYMBOL AND PERFORM THE APPROPRIATE ACTIONS, 9 914, JUMPING TO THE APPROPRIATE PROCESSOR FOR ASSEMBLY, 9 914, WHICH 9 914, EVALUATE THE STATEMENT AS A SYMBOL AND PERFORM THE APPROPRIATE ACTIONS, 9 914, JUMPING TO THE APPROPRIATE PROCESSOR FOR ASSEMBLY, 9 914, WHICH 9 914, EVALUATE THE STATEMENT AS A SYMBOL AND PERFORM THE APPROPRIATE ACTIONS, 9 914, JUMPING TO THE APPROPRIATE PROCESSOR FOR ASSEMBLY, 9 914, WHICH 9 914, EVALUATE THE STATEMENT AS A SYMBL AND PERFORM THE APPROPRIATE ACTIONS, 9

6) PASS 3 IS PERFORMED AT THE CONCLUSION OF EACH ASSEMBLY AND CONSISTS OF UPDATING THE SYMBOL TABLE, 9 912, JUMPING TO THE APPROPRIATE PROCESSOR FOR ASSEMBLY, 9 912, WHICH 9 912, EVALUATE THE STATEMENT AS A SYMBL AND PERFORM THE APPROPRIATE ACTIONS, 9

E AND CAUSE-REFERENCES IF REQUESTED. 9 912, PASS 3 IS PERFORMED AT THE CONCLUSION OF EACH ASSEMBLY AND CONSISTS OF UPDATING THE SYMBOL TABLE, 9 912, JUMPING TO THE APPROPRIATE PROCESSOR FOR ASSEMBLY, 9 912, WHICH 9 912, EVALUATE THE STATEMENT AS A SYMBL AND PERFORM THE APPROPRIATE ACTIONS, 9

912 INITIATED FOR THE PRESENT ARE THE INTER-PASS FUNCTIONS WHICH INITIALIZE EACH PASS AND TERMINATE EACH ASSEMBLY.

9) THEY CAN BE CLARIFIED ONLY WHEN THE OPERATIONS ARE KNOWN. 9 971 PASS 1 FUNCTIONS, 9 912, 911, 91, READING A STATEMENT, 9 911, IN COMPASS, THE SUBROUTINES INPUT, UCARD, AND HAS ARE ALL INVOLVED IN 911 THE READING PROCE.

9 911, 912 READING A STATEMENT, 9 911, IN COMPASS, THE SUBROUTINES INPUT, UCARD, AND HAS ARE ALL INVOLVED IN 911 THE READING PROCE.

9 911, 912 READING A STATEMENT, 9 911, IN COMPASS, THE SUBROUTINES INPUT, UCARD, AND HAS ARE ALL INVOLVED IN 911 THE READING PROCE.

9 911, 912 READING A STATEMENT, 9 911, IN COMPASS, THE SUBROUTINES INPUT, UCARD, AND HAS ARE ALL INVOLVED IN 911 THE READING PROCE.

9 911, 912 READING A STATEMENT, 9 911, IN COMPASS, THE SUBROUTINES INPUT, UCARD, AND HAS ARE ALL INVOLVED IN 911 THE READING PROCE.
This chapter discusses the usage of the 8-bit subroutines in connection with the reading and writing of IBM 360/370 compatible tape files used with or produced by an IBM 360/370 FORTRAN program. It is assumed that the user is familiar with FORTRAN and its input/output concepts. It would be beneficial for the user to have access to a CDC FORTRAN Extended Reference Manual.

**IBM FORTRAN DATA FORMATS**

IBM FORTRAN programs can utilize six types of constant data and four types of variable data. The constant type determines the number of 8-bit byte storage locations needed to represent the data. A default and optional length specification for each variable type determines the number of bytes reserved for that type. Table 8-1 gives the associated length specifications for constants and variables.

<table>
<thead>
<tr>
<th>Constant Type</th>
<th>8-Bit Bytes Allocated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>4</td>
</tr>
<tr>
<td>Real</td>
<td>4 (single precision)</td>
</tr>
<tr>
<td></td>
<td>8 (double precision)</td>
</tr>
<tr>
<td>Complex</td>
<td>8 (two single precision)</td>
</tr>
<tr>
<td></td>
<td>16 (two double precision)</td>
</tr>
<tr>
<td>Logical</td>
<td>4</td>
</tr>
<tr>
<td>Literal</td>
<td>(1 \leq n \leq 255) (n is string length)</td>
</tr>
<tr>
<td>Hexadecimal</td>
<td>1 byte for each 2 digits</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable Type</th>
<th>Default Length (bytes)</th>
<th>Optional Length (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Real</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>Complex</td>
<td>8</td>
<td>16</td>
</tr>
<tr>
<td>Logical</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>
An IBM 360/370 tape file created by an IBM FORTRAN program may contain combinations of the following data types:

- 8-bit characters (X)
- half-word integers, 16 bits (H)
- whole-word integers, 32 bits (W)
- 32-bit floating point (F)
- 64-bit floating point (L)

The letters in parentheses are the corresponding T values as described in table 2-1 in chapter 2. Note that an IBM FORTRAN Program cannot create 64-bit double word integers or 128-bit extended-precision floating point data. All of the above data could be processed also as bit string data (B).

Table 2-1 should also be consulted to determine the appropriate m value for each T selected. A more complete description of IBM data format types appears in appendix C.

The following table 8-2 shows the relationship between IBM FORTRAN data type declarations, byte storage allocation, and the corresponding Tm values for conversion use.

<table>
<thead>
<tr>
<th>IBM FORTRAN Type Declaration</th>
<th>IBM Boundary Alignment</th>
<th>T</th>
<th>8-Bit Bytes Used†</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer*2</td>
<td>halfword</td>
<td>H</td>
<td>2</td>
</tr>
<tr>
<td>Integer</td>
<td>fullword</td>
<td>W</td>
<td>4</td>
</tr>
<tr>
<td>Integer*4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Real*4</td>
<td>fullword</td>
<td>F</td>
<td>4 (single precision)</td>
</tr>
<tr>
<td>Real*8</td>
<td>doubleword</td>
<td>L</td>
<td>8 (double precision)</td>
</tr>
<tr>
<td>Double Precision</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logical</td>
<td>fullword</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>Logical*1</td>
<td>none</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

†The value used for m is determined from number of bytes used.
Table 8-2 contains only examples of explicit type declarations. COMPLEX declarations do not appear since the complex variable is composed of two real data items and follows the conventions for REAL data types. It is for the user to determine how LOGICAL data types are to be converted.

**CDC FORTRAN EXTENDED DATA FORMATS**

Programs written in CDC FORTRAN Extended can utilize seven types of constant data and five types of variable data. The type of constant or variable determines the number of central memory words needed to represent it. Table 8-3 shows the constant and variable types and the associated 6-bit byte length specifications.

<table>
<thead>
<tr>
<th>Constant Type</th>
<th>6-Bit Bytes Allocated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>10</td>
</tr>
<tr>
<td>Real</td>
<td>10</td>
</tr>
<tr>
<td>Double precision</td>
<td>20</td>
</tr>
<tr>
<td>Complex</td>
<td>20 (two real constants)</td>
</tr>
<tr>
<td>Octal</td>
<td>10</td>
</tr>
<tr>
<td>Hollerith</td>
<td>$1 \leq n \leq 150$ (n is string length)</td>
</tr>
<tr>
<td>Logical</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable Type</th>
<th>6-Bit Bytes Allocated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>10</td>
</tr>
<tr>
<td>Real</td>
<td>10</td>
</tr>
<tr>
<td>Double precision</td>
<td>20</td>
</tr>
<tr>
<td>Complex</td>
<td>20</td>
</tr>
<tr>
<td>Logical</td>
<td>10</td>
</tr>
</tbody>
</table>
Internal records in a CDC FORTRAN program may contain combinations of the following data types. The letter in parentheses is the corresponding T value as determined from table 2-1.

- 6-bit characters (X)
- whole-word integer, 60 bits (I)
- single-word floating point, 60 bits (E)
- double-word floating point, 120 bits (D)

All of the above data types could be processed as type B bit string data. Table 2-1 should be consulted to determine the appropriate m value. While a whole-word integer is a data type, integers used in multiplication and division operations are truncated to 48 bits. More complete information on this subject appears in appendix C and in the CDC FORTRAN Extended Reference Manual.

Table 8-4 shows the relationship between CDC FORTRAN data type declarations, storage allocation, and corresponding Tm values for conversion use.

<table>
<thead>
<tr>
<th>CDC FORTRAN Type Declaration</th>
<th>CDC Boundary Alignment</th>
<th>T</th>
<th>CDC 6-Bit Bytes Used†</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>fullword</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>Real</td>
<td>fullword</td>
<td>E</td>
<td>10</td>
</tr>
<tr>
<td>Double precision</td>
<td>doubleword</td>
<td>D</td>
<td>20</td>
</tr>
<tr>
<td>Logical</td>
<td>fullword</td>
<td>usage defined</td>
<td>20</td>
</tr>
</tbody>
</table>

† The value used for m is determined from number of bytes used.

COMPLEX type declarations do not appear in the table as the complex variable is composed of two real data items and follow the conventions for REAL data types. It is left to the user to determine the manner in which LOGICAL data items are to be converted.

SAMPLE FORTRAN JOB

The following sample CDC FORTRAN Extended (FTN) job reads 200 logical records from the second file of an IBM 360 EBCDIC tape. The logical records are 135 characters long and are blocked into 3105 character blocks containing 23 records per block. The tape contains both upper and lower case data. In the program, each record is converted to display code with lower case data folded to upper case equivalent before being displayed upon the file OUTPUT. A sample of the job output is included.
PROGRAM LISTA(TAPE1, OUTPUT)
INTEGER REC1(14), WKS(420), CNT
DIMENSION LH1(L)
STAT=0, 0
CNT=0
CALL XFILE (1*WKS, 420/1*REC1, CNT)
X 420
1
CNT=COUNT 1
IF (CNT < GT, 200) CALL EXIT
CALL ARFAD (WKS, REC1, (X135)*1*STAT)
PRINT 104, REC1
IF (STAT < NE, 00, 0) PRINT 105, STAT
GO TO 1
100 FORMAT (1X, *LABEL RECORD 1 = *8A10)
101 FORMAT (1X, *ERROR IN LABEL RECORD 1, STAT = *, F5.1)
102 FORMAT (1X, *LABEL RECORD 2 = *8A10)
103 FORMAT (1X, *ERROR IN LABEL RECORD 2, STAT = *, F5.1)
104 FORMAT (1X, *13A10, A5)
105 FORMAT (1X, *ERROR IN DATA RECORD, STAT = *, F5.1)
END

COMPASS/TIMS 6 ANY ANY *COMPASS IM

02/22/7100 00
E 2) 211 # CDC 11 A ROSEMARIE 4

CON ASS6/TIMS 6 951216 95129 91 0 THE ASSEMBLY PROCESS 9
ES IN A RATHER STRAIGHTFORWARD MANNER. THIS CHAPTER DESCRIBES THE PRINCIPLE FUNCTIONS OF THE PROCESS IN 9
9 THE DETAILS ARE EXAMINED LATER SECTIONS. 9412 91, 1 TWO-PASS 9
9 ERRORS ARE RATE ON ESSENTIALLY THE SAME PRINCIPLE: 9 THE FIRST PASS READS THE SOURCE FILE, AND PERFORMS 9
9 A(F)EXAMINES EACH INSTRUCTION TO DETERMINE HOW MUCH STORAGE IS REQUIRED.) IN 9
9)THE LEARNING OF A SYMBOL IN THE LOCATION FIELD OF A MACHINE INSTRUCTION 9 OR BY ITS USE IN SOME 9
9)EXPANDS THE VARIOUS HIGHER LEVEL OPERATIONS, SUCH AS MACROS AND 9)DUPLICATIONS. 9)ACUMULATES LI 9
9)THE LENGTH OF EACH BLOCK IN THE 9)ASSEMBLY: 9)2.) THE VALUE, RELATIVE TO SOME USE BLOCK, OF 9)IN 9
9)VALUE AND ORDER OF ALL LITERALS 9)USED. 9
9) THIS INFORMATION, A SECOND PASS OVER THE SOURCE STATEMENTS 9)CAN FILL IN ALL VALUM SYMBOL VALUES, LOC 9
9) CONSISTS OF THE FOLLOWING OPERATIONS: 9)1) READING A STATEMENT, 9)2) EDITING THE STATEMENT TO REMO 9
9)3) LOOK
The character set selected when the system is installed should be compatible with the printers.

SCOPE

With an installation parameter, the installation keypunch format standard can be selected as 026 or 029; the installation parameter can also allow a user to override the standard; a user may select a keypunch mode for his input deck by punching 26 or 29 in columns 79 and 80 of his JOB card or any 7/8/9 end-of-record card. The mode remains set for the remainder of the job or until it is reset by a different mode selection on another 7/8/9 card.

KRONOS

A user can select a keypunch mode for an input deck by use of a card containing 5/7/9 punches in column 1 and the desired conversion mode indicated in column 2. The conversions are indicated by the following punches:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>FORTRAN 029</td>
</tr>
<tr>
<td>8</td>
<td>COBOL 029</td>
</tr>
<tr>
<td>8/9</td>
<td>SNOBOL 029</td>
</tr>
</tbody>
</table>

The conversion mode will remain in effect for the remainder of the job or until reset by a different mode selection on another 5/7/9 card.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>:1</td>
<td>:</td>
<td>001</td>
<td>8.2</td>
<td>00</td>
<td>8.2</td>
<td>3A</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>36</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>01</td>
<td>12-1</td>
<td>61</td>
<td>12-1</td>
<td>41</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>37</td>
</tr>
<tr>
<td>B</td>
<td>B</td>
<td>02</td>
<td>12-2</td>
<td>62</td>
<td>12-2</td>
<td>42</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>38</td>
</tr>
<tr>
<td>C</td>
<td>C</td>
<td>03</td>
<td>12-3</td>
<td>63</td>
<td>12-3</td>
<td>43</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>39</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
<td>04</td>
<td>12-4</td>
<td>64</td>
<td>12-4</td>
<td>44</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>40</td>
</tr>
<tr>
<td>E</td>
<td>E</td>
<td>05</td>
<td>12-5</td>
<td>65</td>
<td>12-5</td>
<td>45</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>41</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>06</td>
<td>12-6</td>
<td>66</td>
<td>12-6</td>
<td>46</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>42</td>
</tr>
<tr>
<td>G</td>
<td>G</td>
<td>07</td>
<td>12-7</td>
<td>67</td>
<td>12-7</td>
<td>47</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>43</td>
</tr>
<tr>
<td>H</td>
<td>H</td>
<td>10</td>
<td>12-8</td>
<td>70</td>
<td>12-8</td>
<td>48</td>
<td>(</td>
<td>(</td>
<td>)</td>
<td>)</td>
<td>44</td>
</tr>
<tr>
<td>I</td>
<td>I</td>
<td>11</td>
<td>12-9</td>
<td>71</td>
<td>12-9</td>
<td>49</td>
<td>)</td>
<td>)</td>
<td>)</td>
<td>)</td>
<td>45</td>
</tr>
<tr>
<td>J</td>
<td>J</td>
<td>12</td>
<td>11-1</td>
<td>41</td>
<td>11-1</td>
<td>4A</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>$</td>
<td>46</td>
</tr>
<tr>
<td>K</td>
<td>K</td>
<td>13</td>
<td>11-2</td>
<td>42</td>
<td>11-2</td>
<td>4B</td>
<td>=</td>
<td>=</td>
<td>=</td>
<td>=</td>
<td>47</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>14</td>
<td>11-3</td>
<td>43</td>
<td>11-3</td>
<td>4C</td>
<td>blank</td>
<td>blank</td>
<td>blank</td>
<td>blank</td>
<td>48</td>
</tr>
<tr>
<td>M</td>
<td>M</td>
<td>15</td>
<td>11-4</td>
<td>44</td>
<td>11-4</td>
<td>4D</td>
<td>, (comma)</td>
<td>, (comma)</td>
<td>, (comma)</td>
<td>, (comma)</td>
<td>49</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>16</td>
<td>11-5</td>
<td>45</td>
<td>11-5</td>
<td>4E</td>
<td>, (period)</td>
<td>, (period)</td>
<td>, (period)</td>
<td>, (period)</td>
<td>50</td>
</tr>
<tr>
<td>O</td>
<td>O</td>
<td>17</td>
<td>11-6</td>
<td>46</td>
<td>11-6</td>
<td>4F</td>
<td>¥</td>
<td>¥</td>
<td>¥</td>
<td>¥</td>
<td>51</td>
</tr>
<tr>
<td>P</td>
<td>P</td>
<td>20</td>
<td>11-7</td>
<td>47</td>
<td>11-7</td>
<td>50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>Q</td>
<td>21</td>
<td>11-8</td>
<td>48</td>
<td>11-8</td>
<td>51</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>R</td>
<td>22</td>
<td>11-9</td>
<td>49</td>
<td>11-9</td>
<td>52</td>
<td>%**</td>
<td>%</td>
<td>%**</td>
<td>%**</td>
<td>54</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
<td>23</td>
<td>0-2</td>
<td>22</td>
<td>0-2</td>
<td>53</td>
<td>।</td>
<td>।</td>
<td>।</td>
<td>।</td>
<td>55</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>24</td>
<td>0-3</td>
<td>23</td>
<td>0-3</td>
<td>54</td>
<td>→</td>
<td>→</td>
<td>→</td>
<td>→</td>
<td>56</td>
</tr>
<tr>
<td>U</td>
<td>U</td>
<td>25</td>
<td>0-4</td>
<td>24</td>
<td>0-4</td>
<td>55</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>57</td>
</tr>
<tr>
<td>V</td>
<td>V</td>
<td>26</td>
<td>0-5</td>
<td>25</td>
<td>0-5</td>
<td>56</td>
<td>।</td>
<td>।</td>
<td>।</td>
<td>।</td>
<td>58</td>
</tr>
<tr>
<td>W</td>
<td>W</td>
<td>27</td>
<td>0-6</td>
<td>26</td>
<td>0-6</td>
<td>57</td>
<td>∧</td>
<td>&amp;</td>
<td>∧</td>
<td>&amp;</td>
<td>59</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>30</td>
<td>0-7</td>
<td>27</td>
<td>0-7</td>
<td>58</td>
<td>′</td>
<td>′</td>
<td>′</td>
<td>′</td>
<td>60</td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>31</td>
<td>0-8</td>
<td>28</td>
<td>0-8</td>
<td>59</td>
<td>′</td>
<td>′</td>
<td>′</td>
<td>′</td>
<td>61</td>
</tr>
<tr>
<td>Z</td>
<td>Z</td>
<td>32</td>
<td>0-9</td>
<td>29</td>
<td>0-9</td>
<td>5A</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&lt;</td>
<td>62</td>
</tr>
<tr>
<td>]</td>
<td>]</td>
<td>33</td>
<td>1-0</td>
<td>0</td>
<td>1-0</td>
<td>31</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
<td>&gt;</td>
<td>63</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>36</td>
<td>3-0</td>
<td>0</td>
<td>3-0</td>
<td>32</td>
<td>≤</td>
<td>@</td>
<td>≤</td>
<td>@</td>
<td>64</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>37</td>
<td>4-0</td>
<td>0</td>
<td>4-0</td>
<td>33</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>\</td>
<td>65</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>40</td>
<td>5-0</td>
<td>0</td>
<td>5-0</td>
<td>34</td>
<td>; (semicolon)</td>
<td>; (semicolon)</td>
<td>; (semicolon)</td>
<td>; (semicolon)</td>
<td>66</td>
</tr>
</tbody>
</table>

† Twelve or more zero bits at the end of a 60-bit word are interpreted as end-of-line mark rather than two colons. End-of-line mark is converted to external BCD 1632.

†† In installations using the CDC 63-graphic set, display code 00 has no associated graphic or Hollerith code; display code 63 is the colon (8-2 punch).

††† The alternate Hollerith (026) and ASCII (029) punches are accepted for input only.
<table>
<thead>
<tr>
<th>Collating Sequence Decimal/Octal</th>
<th>CDC Graphic</th>
<th>Display Code</th>
<th>External BCD</th>
<th>Collating Sequence Decimal/Octal</th>
<th>CDC Graphic</th>
<th>Display Code</th>
<th>External BCD</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 00</td>
<td>blank</td>
<td>55</td>
<td>20</td>
<td>32 40</td>
<td>H</td>
<td>10</td>
<td>70</td>
</tr>
<tr>
<td>01 01</td>
<td>≤</td>
<td>74</td>
<td>15</td>
<td>33 41</td>
<td>I</td>
<td>11</td>
<td>71</td>
</tr>
<tr>
<td>02 02</td>
<td>%</td>
<td>63 †</td>
<td>16 †</td>
<td>34 42</td>
<td>V</td>
<td>66</td>
<td>52</td>
</tr>
<tr>
<td>03 03</td>
<td>[</td>
<td>61</td>
<td>17</td>
<td>35 43</td>
<td>J</td>
<td>12</td>
<td>41</td>
</tr>
<tr>
<td>04 04</td>
<td>→</td>
<td>65</td>
<td>35</td>
<td>36 44</td>
<td>K</td>
<td>13</td>
<td>42</td>
</tr>
<tr>
<td>05 05</td>
<td>≡</td>
<td>60</td>
<td>36</td>
<td>37 45</td>
<td>L</td>
<td>14</td>
<td>43</td>
</tr>
<tr>
<td>06 06</td>
<td>^</td>
<td>67</td>
<td>37</td>
<td>38 46</td>
<td>M</td>
<td>15</td>
<td>44</td>
</tr>
<tr>
<td>07 07</td>
<td>‹</td>
<td>70</td>
<td>55</td>
<td>39 47</td>
<td>N</td>
<td>16</td>
<td>45</td>
</tr>
<tr>
<td>08 10</td>
<td>‡</td>
<td>71</td>
<td>56</td>
<td>40 50</td>
<td>O</td>
<td>17</td>
<td>46</td>
</tr>
<tr>
<td>09 11</td>
<td>&lt;</td>
<td>73</td>
<td>57</td>
<td>41 51</td>
<td>P</td>
<td>20</td>
<td>47</td>
</tr>
<tr>
<td>10 12</td>
<td>≥</td>
<td>75</td>
<td>75</td>
<td>42 52</td>
<td>Q</td>
<td>21</td>
<td>50</td>
</tr>
<tr>
<td>11 13</td>
<td>=</td>
<td>76</td>
<td>76</td>
<td>43 53</td>
<td>R</td>
<td>22</td>
<td>51</td>
</tr>
<tr>
<td>12 14</td>
<td>.</td>
<td>57</td>
<td>73</td>
<td>44 54</td>
<td>J</td>
<td>62</td>
<td>32</td>
</tr>
<tr>
<td>13 15</td>
<td>)</td>
<td>52</td>
<td>74</td>
<td>45 55</td>
<td>S</td>
<td>23</td>
<td>22</td>
</tr>
<tr>
<td>14 16</td>
<td>;</td>
<td>77</td>
<td>77</td>
<td>46 56</td>
<td>T</td>
<td>24</td>
<td>23</td>
</tr>
<tr>
<td>15 17</td>
<td>+</td>
<td>45</td>
<td>60</td>
<td>47 57</td>
<td>U</td>
<td>25</td>
<td>24</td>
</tr>
<tr>
<td>16 20</td>
<td>$</td>
<td>53</td>
<td>53</td>
<td>48 60</td>
<td>V</td>
<td>26</td>
<td>25</td>
</tr>
<tr>
<td>17 21</td>
<td>*</td>
<td>47</td>
<td>54</td>
<td>49 61</td>
<td>W</td>
<td>27</td>
<td>26</td>
</tr>
<tr>
<td>18 22</td>
<td>-</td>
<td>46</td>
<td>40</td>
<td>50 62</td>
<td>X</td>
<td>30</td>
<td>27</td>
</tr>
<tr>
<td>19 23</td>
<td>/</td>
<td>50</td>
<td>21</td>
<td>51 63</td>
<td>Y</td>
<td>31</td>
<td>30</td>
</tr>
<tr>
<td>20 24</td>
<td>,</td>
<td>56</td>
<td>33</td>
<td>52 64</td>
<td>Z</td>
<td>32</td>
<td>31</td>
</tr>
<tr>
<td>21 25</td>
<td>(</td>
<td>51</td>
<td>34</td>
<td>53 65</td>
<td>:</td>
<td>00 †</td>
<td>none†</td>
</tr>
<tr>
<td>22 26</td>
<td>)</td>
<td>54</td>
<td>13</td>
<td>54 66</td>
<td>0</td>
<td>33</td>
<td>12</td>
</tr>
<tr>
<td>23 27</td>
<td>≠</td>
<td>64</td>
<td>14</td>
<td>55 67</td>
<td>1</td>
<td>34</td>
<td>01</td>
</tr>
<tr>
<td>24 30</td>
<td>&lt;</td>
<td>72</td>
<td>72</td>
<td>56 70</td>
<td>2</td>
<td>35</td>
<td>02</td>
</tr>
<tr>
<td>25 31</td>
<td>A</td>
<td>01</td>
<td>61</td>
<td>57 71</td>
<td>3</td>
<td>36</td>
<td>03</td>
</tr>
<tr>
<td>26 32</td>
<td>B</td>
<td>02</td>
<td>62</td>
<td>58 72</td>
<td>4</td>
<td>37</td>
<td>04</td>
</tr>
<tr>
<td>27 33</td>
<td>C</td>
<td>03</td>
<td>63</td>
<td>59 73</td>
<td>5</td>
<td>40</td>
<td>05</td>
</tr>
<tr>
<td>28 34</td>
<td>D</td>
<td>04</td>
<td>64</td>
<td>60 74</td>
<td>6</td>
<td>41</td>
<td>06</td>
</tr>
<tr>
<td>29 35</td>
<td>E</td>
<td>05</td>
<td>65</td>
<td>61 75</td>
<td>7</td>
<td>42</td>
<td>07</td>
</tr>
<tr>
<td>30 36</td>
<td>F</td>
<td>06</td>
<td>66</td>
<td>62 76</td>
<td>8</td>
<td>43</td>
<td>10</td>
</tr>
<tr>
<td>31 37</td>
<td>G</td>
<td>07</td>
<td>67</td>
<td>63 77</td>
<td>9</td>
<td>44</td>
<td>11</td>
</tr>
</tbody>
</table>

† In installations using the 63-graphic set, the % graphic does not exist. The : graphic is display code 63, External BCD code 16.
<table>
<thead>
<tr>
<th>Collating Sequence Decimal/Octal</th>
<th>ASCII Graphic Subset</th>
<th>Display Code</th>
<th>ASCII Code</th>
<th>Collating Sequence Decimal/Octal</th>
<th>ASCII Graphic Subset</th>
<th>Display Code</th>
<th>ASCII Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>blank</td>
<td>55</td>
<td>20</td>
<td>32</td>
<td>@</td>
<td>74</td>
<td>40</td>
</tr>
<tr>
<td>01</td>
<td>!</td>
<td>66</td>
<td>21</td>
<td>33</td>
<td>A</td>
<td>01</td>
<td>41</td>
</tr>
<tr>
<td>02</td>
<td>&quot;</td>
<td>64</td>
<td>22</td>
<td>34</td>
<td>B</td>
<td>02</td>
<td>42</td>
</tr>
<tr>
<td>03</td>
<td>#</td>
<td>60</td>
<td>23</td>
<td>35</td>
<td>C</td>
<td>03</td>
<td>43</td>
</tr>
<tr>
<td>04</td>
<td>$</td>
<td>53</td>
<td>24</td>
<td>36</td>
<td>D</td>
<td>04</td>
<td>44</td>
</tr>
<tr>
<td>05</td>
<td>%</td>
<td>63</td>
<td>25</td>
<td>37</td>
<td>E</td>
<td>05</td>
<td>45</td>
</tr>
<tr>
<td>06</td>
<td>&amp;</td>
<td>67</td>
<td>26</td>
<td>38</td>
<td>F</td>
<td>06</td>
<td>46</td>
</tr>
<tr>
<td>07</td>
<td>'</td>
<td>70</td>
<td>27</td>
<td>39</td>
<td>G</td>
<td>07</td>
<td>47</td>
</tr>
<tr>
<td>08</td>
<td>(</td>
<td>51</td>
<td>28</td>
<td>40</td>
<td>H</td>
<td>10</td>
<td>48</td>
</tr>
<tr>
<td>09</td>
<td>)</td>
<td>52</td>
<td>29</td>
<td>41</td>
<td>I</td>
<td>11</td>
<td>49</td>
</tr>
<tr>
<td>10</td>
<td>*</td>
<td>47</td>
<td>2A</td>
<td>42</td>
<td>J</td>
<td>12</td>
<td>4A</td>
</tr>
<tr>
<td>11</td>
<td>+</td>
<td>45</td>
<td>2B</td>
<td>43</td>
<td>K</td>
<td>13</td>
<td>4B</td>
</tr>
<tr>
<td>12</td>
<td>.</td>
<td>56</td>
<td>2C</td>
<td>44</td>
<td>L</td>
<td>14</td>
<td>4C</td>
</tr>
<tr>
<td>13</td>
<td>-</td>
<td>46</td>
<td>2D</td>
<td>45</td>
<td>M</td>
<td>15</td>
<td>4D</td>
</tr>
<tr>
<td>14</td>
<td>:</td>
<td>57</td>
<td>2E</td>
<td>46</td>
<td>N</td>
<td>16</td>
<td>4E</td>
</tr>
<tr>
<td>15</td>
<td>/</td>
<td>50</td>
<td>2F</td>
<td>47</td>
<td>O</td>
<td>17</td>
<td>4F</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>33</td>
<td>30</td>
<td>48</td>
<td>P</td>
<td>20</td>
<td>50</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>34</td>
<td>31</td>
<td>49</td>
<td>Q</td>
<td>21</td>
<td>51</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
<td>35</td>
<td>32</td>
<td>50</td>
<td>R</td>
<td>22</td>
<td>52</td>
</tr>
<tr>
<td>19</td>
<td>3</td>
<td>36</td>
<td>33</td>
<td>51</td>
<td>S</td>
<td>23</td>
<td>53</td>
</tr>
<tr>
<td>20</td>
<td>4</td>
<td>37</td>
<td>34</td>
<td>52</td>
<td>T</td>
<td>24</td>
<td>54</td>
</tr>
<tr>
<td>21</td>
<td>5</td>
<td>40</td>
<td>35</td>
<td>53</td>
<td>U</td>
<td>25</td>
<td>55</td>
</tr>
<tr>
<td>22</td>
<td>6</td>
<td>41</td>
<td>36</td>
<td>54</td>
<td>V</td>
<td>26</td>
<td>56</td>
</tr>
<tr>
<td>23</td>
<td>7</td>
<td>42</td>
<td>37</td>
<td>55</td>
<td>W</td>
<td>27</td>
<td>57</td>
</tr>
<tr>
<td>24</td>
<td>8</td>
<td>43</td>
<td>38</td>
<td>56</td>
<td>X</td>
<td>30</td>
<td>58</td>
</tr>
<tr>
<td>25</td>
<td>9</td>
<td>44</td>
<td>39</td>
<td>57</td>
<td>Y</td>
<td>31</td>
<td>59</td>
</tr>
<tr>
<td>26</td>
<td>:</td>
<td>00</td>
<td>3A</td>
<td>58</td>
<td>Z</td>
<td>32</td>
<td>5A</td>
</tr>
<tr>
<td>27</td>
<td>;</td>
<td>77</td>
<td>3B</td>
<td>59</td>
<td>[</td>
<td>61</td>
<td>5B</td>
</tr>
<tr>
<td>28</td>
<td>&lt;</td>
<td>72</td>
<td>3C</td>
<td>60</td>
<td>\</td>
<td>75</td>
<td>5C</td>
</tr>
<tr>
<td>29</td>
<td>=</td>
<td>54</td>
<td>3D</td>
<td>61</td>
<td>]</td>
<td>62</td>
<td>5D</td>
</tr>
<tr>
<td>30</td>
<td>&gt;</td>
<td>73</td>
<td>3E</td>
<td>62</td>
<td>^</td>
<td>76</td>
<td>5E</td>
</tr>
<tr>
<td>31</td>
<td>?</td>
<td>71</td>
<td>3F</td>
<td>63</td>
<td>_</td>
<td>65</td>
<td>5F</td>
</tr>
</tbody>
</table>

† In installations using a 63-graphic set, the % graphic does not exist. The : graphic is display code 63.
<table>
<thead>
<tr>
<th>ASCII Character</th>
<th>Card Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>11-8-2</td>
<td>5A</td>
</tr>
</tbody>
</table>

**LEGEND**

<table>
<thead>
<tr>
<th>ASCII Character</th>
<th>Card Code</th>
<th>EBCDIC Character</th>
<th>EBCDIC Code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(Hexadecimal)</td>
<td></td>
</tr>
</tbody>
</table>
### Extended Binary Coded Decimal Interchange Code (EBCDIC) with Punched Card Codes and ASCII Translation

<table>
<thead>
<tr>
<th>Bits</th>
<th>EBCDIC</th>
<th>Card Code</th>
<th>ASCII Character</th>
<th>ASCII Code (Hexadecimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
<td>NULL</td>
<td>0</td>
<td>NULL</td>
<td>0</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>0</td>
<td>32</td>
<td>DC1</td>
<td>11</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>1</td>
<td>64</td>
<td>DC2</td>
<td>12</td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>2</td>
<td>96</td>
<td>DC3</td>
<td>13</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>3</td>
<td>128</td>
<td>ETX</td>
<td>14</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>4</td>
<td>160</td>
<td>ETA</td>
<td>15</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>5</td>
<td>192</td>
<td>ETB</td>
<td>16</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>6</td>
<td>224</td>
<td>ETM</td>
<td>17</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>7</td>
<td>256</td>
<td>GEN</td>
<td>18</td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>8</td>
<td>288</td>
<td>EM</td>
<td>19</td>
</tr>
<tr>
<td>1 0 1 0</td>
<td>9</td>
<td>320</td>
<td>CR</td>
<td>20</td>
</tr>
<tr>
<td>1 0 1 1</td>
<td>A</td>
<td>352</td>
<td>VT</td>
<td>21</td>
</tr>
<tr>
<td>1 1 0 0</td>
<td>B</td>
<td>384</td>
<td>FF</td>
<td>22</td>
</tr>
<tr>
<td>1 1 0 1</td>
<td>C</td>
<td>416</td>
<td>CR</td>
<td>23</td>
</tr>
<tr>
<td>1 1 1 0</td>
<td>D</td>
<td>448</td>
<td>FF</td>
<td>24</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>E</td>
<td>480</td>
<td>SI</td>
<td>25</td>
</tr>
</tbody>
</table>

### Legend
- **EBCDIC Character**: A set of binary codes used in the EBCDIC code for representing characters.
- **Card Code**: A set of binary codes used in punched card systems.
- **ASCII Character**: A set of binary codes used in the American Standard Code for Information Interchange (ASCII).
- **ASCII Code (Hexadecimal)**: The hexadecimal representation of the ASCII code.
### CONTROL DATA CHARACTER SETS
SHOWING TRANSLATIONS BETWEEN DISPLAY CODE AND ASCII/EBCDIC

<table>
<thead>
<tr>
<th>DISPLAY CODE</th>
<th>ASCII</th>
<th>EBCDIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCTAL</td>
<td>CH</td>
<td>HEX</td>
</tr>
<tr>
<td>00</td>
<td>3A</td>
<td>SUB</td>
</tr>
<tr>
<td>01</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>02</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>03</td>
<td>C</td>
<td>C</td>
</tr>
<tr>
<td>04</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>05</td>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>06</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>07</td>
<td>G</td>
<td>G</td>
</tr>
<tr>
<td>08</td>
<td>H</td>
<td>H</td>
</tr>
<tr>
<td>09</td>
<td>I</td>
<td>I</td>
</tr>
<tr>
<td>10</td>
<td>J</td>
<td>J</td>
</tr>
<tr>
<td>11</td>
<td>K</td>
<td>K</td>
</tr>
<tr>
<td>12</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>13</td>
<td>M</td>
<td>M</td>
</tr>
<tr>
<td>14</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>15</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>16</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>17</td>
<td>Q</td>
<td>Q</td>
</tr>
<tr>
<td>18</td>
<td>R</td>
<td>R</td>
</tr>
<tr>
<td>19</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>20</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>21</td>
<td>U</td>
<td>U</td>
</tr>
<tr>
<td>22</td>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td>23</td>
<td>W</td>
<td>W</td>
</tr>
<tr>
<td>24</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>25</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>26</td>
<td>Z</td>
<td>Z</td>
</tr>
<tr>
<td>27</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>28</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>29</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>30</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>31</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>32</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>33</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>34</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>35</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>36</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>37</td>
<td>:</td>
<td>:</td>
</tr>
</tbody>
</table>

### NOTES:

1. The terms "upper case" and "lower case" apply only to the case conversions, and do not necessarily reflect any true "case".
2. When translating from Display Code to ASCII/EBCDIC, the "upper case" and "lower case" characters fold together to a single Display Code equivalent character.
3. When translating from ASCII/EBCDIC to Display Code, the "upper case" and "lower case" characters fold together to a single Display Code equivalent character.
4. All ASCII and EBCDIC codes not listed are translated to Display Code 56 (SP).

5. Where two Display Code graphics are shown for a single octal code, the leftmost graphic corresponds to the CDC 64-character set, and the rightmost graphic corresponds to the CDC 64-character ASCII subset.

6. In a 63-character set system, the display code for the right graphic is 63. The % character does not exist, and translations from ASCII/EBCDIC % to ENO yield blank (55).
### HEXADECIMAL–OCTAL CONVERSION TABLE

<table>
<thead>
<tr>
<th>Second Hexadecimal Digit</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>
<th>8</th>
<th>9</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>020</td>
<td>040</td>
<td>060</td>
<td>100</td>
<td>120</td>
<td>140</td>
<td>160</td>
<td>200</td>
<td>220</td>
<td>240</td>
<td>260</td>
<td>300</td>
<td>320</td>
<td>340</td>
<td>360</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>021</td>
<td>041</td>
<td>061</td>
<td>101</td>
<td>121</td>
<td>141</td>
<td>161</td>
<td>201</td>
<td>221</td>
<td>241</td>
<td>261</td>
<td>301</td>
<td>321</td>
<td>341</td>
<td>361</td>
</tr>
<tr>
<td>2</td>
<td>002</td>
<td>022</td>
<td>042</td>
<td>062</td>
<td>102</td>
<td>122</td>
<td>142</td>
<td>162</td>
<td>202</td>
<td>222</td>
<td>242</td>
<td>262</td>
<td>302</td>
<td>322</td>
<td>342</td>
<td>362</td>
</tr>
<tr>
<td>3</td>
<td>003</td>
<td>023</td>
<td>043</td>
<td>063</td>
<td>103</td>
<td>123</td>
<td>143</td>
<td>163</td>
<td>203</td>
<td>223</td>
<td>243</td>
<td>263</td>
<td>303</td>
<td>323</td>
<td>343</td>
<td>363</td>
</tr>
<tr>
<td>4</td>
<td>004</td>
<td>024</td>
<td>044</td>
<td>064</td>
<td>104</td>
<td>124</td>
<td>144</td>
<td>164</td>
<td>204</td>
<td>224</td>
<td>244</td>
<td>264</td>
<td>304</td>
<td>324</td>
<td>344</td>
<td>364</td>
</tr>
<tr>
<td>5</td>
<td>005</td>
<td>025</td>
<td>045</td>
<td>065</td>
<td>105</td>
<td>125</td>
<td>145</td>
<td>165</td>
<td>205</td>
<td>225</td>
<td>245</td>
<td>265</td>
<td>305</td>
<td>325</td>
<td>345</td>
<td>365</td>
</tr>
<tr>
<td>6</td>
<td>006</td>
<td>026</td>
<td>046</td>
<td>066</td>
<td>106</td>
<td>126</td>
<td>146</td>
<td>166</td>
<td>206</td>
<td>226</td>
<td>246</td>
<td>266</td>
<td>306</td>
<td>326</td>
<td>346</td>
<td>366</td>
</tr>
<tr>
<td>7</td>
<td>007</td>
<td>027</td>
<td>047</td>
<td>067</td>
<td>107</td>
<td>127</td>
<td>147</td>
<td>167</td>
<td>207</td>
<td>227</td>
<td>247</td>
<td>267</td>
<td>307</td>
<td>327</td>
<td>347</td>
<td>367</td>
</tr>
<tr>
<td>8</td>
<td>010</td>
<td>030</td>
<td>050</td>
<td>070</td>
<td>110</td>
<td>130</td>
<td>150</td>
<td>170</td>
<td>210</td>
<td>230</td>
<td>250</td>
<td>270</td>
<td>310</td>
<td>330</td>
<td>350</td>
<td>370</td>
</tr>
<tr>
<td>9</td>
<td>011</td>
<td>031</td>
<td>051</td>
<td>071</td>
<td>111</td>
<td>131</td>
<td>151</td>
<td>171</td>
<td>211</td>
<td>231</td>
<td>251</td>
<td>271</td>
<td>311</td>
<td>331</td>
<td>351</td>
<td>371</td>
</tr>
<tr>
<td>A</td>
<td>012</td>
<td>032</td>
<td>052</td>
<td>072</td>
<td>112</td>
<td>132</td>
<td>152</td>
<td>172</td>
<td>212</td>
<td>232</td>
<td>252</td>
<td>272</td>
<td>312</td>
<td>332</td>
<td>352</td>
<td>372</td>
</tr>
<tr>
<td>B</td>
<td>013</td>
<td>033</td>
<td>053</td>
<td>073</td>
<td>113</td>
<td>133</td>
<td>153</td>
<td>173</td>
<td>213</td>
<td>233</td>
<td>253</td>
<td>273</td>
<td>313</td>
<td>333</td>
<td>353</td>
<td>373</td>
</tr>
<tr>
<td>C</td>
<td>014</td>
<td>034</td>
<td>054</td>
<td>074</td>
<td>114</td>
<td>134</td>
<td>154</td>
<td>174</td>
<td>214</td>
<td>234</td>
<td>254</td>
<td>274</td>
<td>314</td>
<td>334</td>
<td>354</td>
<td>374</td>
</tr>
<tr>
<td>D</td>
<td>015</td>
<td>035</td>
<td>055</td>
<td>075</td>
<td>115</td>
<td>135</td>
<td>155</td>
<td>175</td>
<td>215</td>
<td>235</td>
<td>255</td>
<td>275</td>
<td>315</td>
<td>335</td>
<td>355</td>
<td>375</td>
</tr>
<tr>
<td>E</td>
<td>016</td>
<td>036</td>
<td>056</td>
<td>076</td>
<td>116</td>
<td>136</td>
<td>156</td>
<td>176</td>
<td>216</td>
<td>236</td>
<td>256</td>
<td>276</td>
<td>316</td>
<td>336</td>
<td>356</td>
<td>376</td>
</tr>
<tr>
<td>F</td>
<td>017</td>
<td>037</td>
<td>057</td>
<td>077</td>
<td>117</td>
<td>137</td>
<td>157</td>
<td>177</td>
<td>217</td>
<td>237</td>
<td>257</td>
<td>277</td>
<td>317</td>
<td>337</td>
<td>357</td>
<td>377</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Octal</th>
<th>000</th>
<th>020</th>
<th>040</th>
<th>060</th>
<th>100</th>
<th>120</th>
<th>140</th>
<th>160</th>
<th>200</th>
<th>220</th>
<th>240</th>
<th>260</th>
<th>300</th>
<th>320</th>
<th>340</th>
<th>360</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>037</td>
<td>077</td>
<td>137</td>
<td>177</td>
<td>237</td>
<td>277</td>
<td>337</td>
<td>377</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**COBOL O29 CHARACTER SET**

<table>
<thead>
<tr>
<th>O29 BCD</th>
<th>Char.</th>
<th>Display</th>
<th>Hollerith</th>
<th>O29 BCD</th>
<th>Char.</th>
<th>Display</th>
<th>Hollerith</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>33</td>
<td>0</td>
<td>40</td>
<td>-</td>
<td>46</td>
<td>11</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>34</td>
<td>1</td>
<td>41</td>
<td>J</td>
<td>12</td>
<td>11-1</td>
</tr>
<tr>
<td>02</td>
<td>2</td>
<td>35</td>
<td>2</td>
<td>42</td>
<td>K</td>
<td>13</td>
<td>11-2</td>
</tr>
<tr>
<td>03</td>
<td>3</td>
<td>36</td>
<td>3</td>
<td>43</td>
<td>L</td>
<td>14</td>
<td>11-3</td>
</tr>
<tr>
<td>04</td>
<td>4</td>
<td>37</td>
<td>4</td>
<td>44</td>
<td>M</td>
<td>15</td>
<td>11-4</td>
</tr>
<tr>
<td>05</td>
<td>5</td>
<td>40</td>
<td>5</td>
<td>45</td>
<td>N</td>
<td>16</td>
<td>11-5</td>
</tr>
<tr>
<td>06</td>
<td>6</td>
<td>41</td>
<td>6</td>
<td>46</td>
<td>O</td>
<td>17</td>
<td>11-6</td>
</tr>
<tr>
<td>07</td>
<td>7</td>
<td>42</td>
<td>7</td>
<td>47</td>
<td>P</td>
<td>20</td>
<td>11-7</td>
</tr>
<tr>
<td>08</td>
<td>8</td>
<td>43</td>
<td>8</td>
<td>50</td>
<td>Q</td>
<td>21</td>
<td>11-8</td>
</tr>
<tr>
<td>09</td>
<td>9</td>
<td>44</td>
<td>9</td>
<td>51</td>
<td>R</td>
<td>22</td>
<td>11-9</td>
</tr>
<tr>
<td>10</td>
<td>:</td>
<td>63</td>
<td>8-2</td>
<td>52</td>
<td>null</td>
<td>63</td>
<td>11-0</td>
</tr>
<tr>
<td>11</td>
<td>#</td>
<td>75</td>
<td>8-3</td>
<td>53</td>
<td>$</td>
<td>53</td>
<td>11-8-3</td>
</tr>
<tr>
<td>12</td>
<td>@</td>
<td>74</td>
<td>8-4</td>
<td>54</td>
<td>*</td>
<td>47</td>
<td>11-8-4</td>
</tr>
<tr>
<td>13</td>
<td>'</td>
<td>64</td>
<td>8-5</td>
<td>55</td>
<td>)</td>
<td>52</td>
<td>11-8-5</td>
</tr>
<tr>
<td>14</td>
<td>=</td>
<td>54</td>
<td>8-6</td>
<td>56</td>
<td>;</td>
<td>77</td>
<td>11-8-6</td>
</tr>
<tr>
<td>15</td>
<td>null</td>
<td>63</td>
<td>8-7</td>
<td>57</td>
<td>null</td>
<td>63</td>
<td>11-8-7</td>
</tr>
<tr>
<td>16</td>
<td>&amp;</td>
<td>71</td>
<td>12</td>
<td>60</td>
<td>space</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>A</td>
<td>01</td>
<td>12-1</td>
<td>61</td>
<td>/</td>
<td>50</td>
<td>0-1</td>
</tr>
<tr>
<td>18</td>
<td>B</td>
<td>02</td>
<td>12-2</td>
<td>62</td>
<td>S</td>
<td>23</td>
<td>0-2</td>
</tr>
<tr>
<td>19</td>
<td>C</td>
<td>03</td>
<td>12-3</td>
<td>63</td>
<td>T</td>
<td>24</td>
<td>0-3</td>
</tr>
<tr>
<td>20</td>
<td>D</td>
<td>04</td>
<td>12-4</td>
<td>64</td>
<td>U</td>
<td>25</td>
<td>0-4</td>
</tr>
<tr>
<td>21</td>
<td>E</td>
<td>05</td>
<td>12-5</td>
<td>65</td>
<td>V</td>
<td>26</td>
<td>0-5</td>
</tr>
<tr>
<td>22</td>
<td>F</td>
<td>06</td>
<td>12-6</td>
<td>66</td>
<td>W</td>
<td>27</td>
<td>0-6</td>
</tr>
<tr>
<td>23</td>
<td>G</td>
<td>07</td>
<td>12-7</td>
<td>67</td>
<td>X</td>
<td>30</td>
<td>0-7</td>
</tr>
<tr>
<td>24</td>
<td>H</td>
<td>10</td>
<td>12-8</td>
<td>70</td>
<td>Y</td>
<td>31</td>
<td>0-8</td>
</tr>
<tr>
<td>25</td>
<td>I</td>
<td>11</td>
<td>12-9</td>
<td>71</td>
<td>Z</td>
<td>32</td>
<td>0-9</td>
</tr>
<tr>
<td>26</td>
<td>ç</td>
<td>65</td>
<td>12-0</td>
<td>72</td>
<td>null</td>
<td>63</td>
<td>0-8-2</td>
</tr>
<tr>
<td>27</td>
<td>.</td>
<td>57</td>
<td>12-8-3</td>
<td>73</td>
<td>,</td>
<td>56</td>
<td>0-8-3</td>
</tr>
<tr>
<td>28</td>
<td>&lt;</td>
<td>72</td>
<td>12-8-4</td>
<td>74</td>
<td>%</td>
<td>70</td>
<td>0-8-4</td>
</tr>
<tr>
<td>29</td>
<td>(</td>
<td>51</td>
<td>12-8-5</td>
<td>75</td>
<td>null</td>
<td>63</td>
<td>0-8-5</td>
</tr>
<tr>
<td>30</td>
<td>+</td>
<td>45</td>
<td>12-8-6</td>
<td>76</td>
<td>&gt;</td>
<td>73</td>
<td>0-8-6</td>
</tr>
<tr>
<td>31</td>
<td>\</td>
<td>67</td>
<td>12-8-7</td>
<td>77</td>
<td>null</td>
<td>63</td>
<td>0-8-7</td>
</tr>
</tbody>
</table>

(Note: There are no 60, 61, 62, 66, or 76 display codes.)
## FORTRAN O29 CHARACTER SET

<table>
<thead>
<tr>
<th>O29 BCD</th>
<th>Char.</th>
<th>Display</th>
<th>Hollerith</th>
<th>O29 BCD</th>
<th>Char.</th>
<th>Display</th>
<th>Hollerith</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>33</td>
<td>0</td>
<td>40</td>
<td>-</td>
<td>46</td>
<td>11</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>34</td>
<td>1</td>
<td>41</td>
<td>J</td>
<td>12</td>
<td>11-1</td>
</tr>
<tr>
<td>02</td>
<td>2</td>
<td>35</td>
<td>2</td>
<td>42</td>
<td>K</td>
<td>13</td>
<td>11-2</td>
</tr>
<tr>
<td>03</td>
<td>3</td>
<td>36</td>
<td>3</td>
<td>43</td>
<td>L</td>
<td>14</td>
<td>11-3</td>
</tr>
<tr>
<td>04</td>
<td>4</td>
<td>37</td>
<td>4</td>
<td>44</td>
<td>M</td>
<td>15</td>
<td>11-4</td>
</tr>
<tr>
<td>05</td>
<td>5</td>
<td>40</td>
<td>5</td>
<td>45</td>
<td>N</td>
<td>16</td>
<td>11-5</td>
</tr>
<tr>
<td>06</td>
<td>6</td>
<td>41</td>
<td>6</td>
<td>46</td>
<td>O</td>
<td>17</td>
<td>11-6</td>
</tr>
<tr>
<td>07</td>
<td>7</td>
<td>42</td>
<td>7</td>
<td>47</td>
<td>P</td>
<td>20</td>
<td>11-7</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>43</td>
<td>8</td>
<td>50</td>
<td>Q</td>
<td>21</td>
<td>11-8</td>
</tr>
<tr>
<td>11</td>
<td>9</td>
<td>44</td>
<td>9</td>
<td>51</td>
<td>R</td>
<td>22</td>
<td>11-9</td>
</tr>
<tr>
<td>12</td>
<td>:</td>
<td>63</td>
<td>8-2</td>
<td>52</td>
<td>V</td>
<td>66</td>
<td>11-0</td>
</tr>
<tr>
<td>13</td>
<td>=</td>
<td>54</td>
<td>8-3</td>
<td>53</td>
<td>$</td>
<td>53</td>
<td>11-8-3</td>
</tr>
<tr>
<td>14</td>
<td>*</td>
<td>47</td>
<td>8-4</td>
<td>54</td>
<td>*</td>
<td>47</td>
<td>11-8-4</td>
</tr>
<tr>
<td>15</td>
<td>*</td>
<td>47</td>
<td>8-5</td>
<td>55</td>
<td>)</td>
<td>32</td>
<td>11-8-5</td>
</tr>
<tr>
<td>16</td>
<td>=</td>
<td>54</td>
<td>8-6</td>
<td>56</td>
<td>;</td>
<td>77</td>
<td>11-8-6</td>
</tr>
<tr>
<td>17</td>
<td>I</td>
<td>61</td>
<td>8-7</td>
<td>57</td>
<td></td>
<td></td>
<td>76</td>
</tr>
<tr>
<td>20</td>
<td>+</td>
<td>45</td>
<td>12</td>
<td>60</td>
<td>space</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>A</td>
<td>01</td>
<td>12-1</td>
<td>61</td>
<td>/</td>
<td>50</td>
<td>0-1</td>
</tr>
<tr>
<td>22</td>
<td>B</td>
<td>02</td>
<td>12-2</td>
<td>62</td>
<td>S</td>
<td>23</td>
<td>0-2</td>
</tr>
<tr>
<td>23</td>
<td>C</td>
<td>03</td>
<td>12-3</td>
<td>63</td>
<td>T</td>
<td>24</td>
<td>0-3</td>
</tr>
<tr>
<td>24</td>
<td>D</td>
<td>04</td>
<td>12-4</td>
<td>64</td>
<td>U</td>
<td>25</td>
<td>0-4</td>
</tr>
<tr>
<td>25</td>
<td>E</td>
<td>05</td>
<td>12-5</td>
<td>65</td>
<td>V</td>
<td>26</td>
<td>0-5</td>
</tr>
<tr>
<td>26</td>
<td>F</td>
<td>06</td>
<td>12-6</td>
<td>66</td>
<td>W</td>
<td>27</td>
<td>0-6</td>
</tr>
<tr>
<td>27</td>
<td>G</td>
<td>07</td>
<td>12-7</td>
<td>67</td>
<td>X</td>
<td>30</td>
<td>0-7</td>
</tr>
<tr>
<td>30</td>
<td>H</td>
<td>10</td>
<td>12-8</td>
<td>70</td>
<td>Y</td>
<td>31</td>
<td>0-8</td>
</tr>
<tr>
<td>31</td>
<td>I</td>
<td>11</td>
<td>12-9</td>
<td>71</td>
<td>Z</td>
<td>32</td>
<td>0-9</td>
</tr>
<tr>
<td>32</td>
<td>&lt;</td>
<td>72</td>
<td>12-0</td>
<td>72</td>
<td>space</td>
<td>55</td>
<td>0-8-2</td>
</tr>
<tr>
<td>33</td>
<td>.</td>
<td>57</td>
<td>12-8-3</td>
<td>73</td>
<td>,</td>
<td>56</td>
<td>0-8-3</td>
</tr>
<tr>
<td>34</td>
<td>)</td>
<td>52</td>
<td>12-8-4</td>
<td>74</td>
<td>(</td>
<td>51</td>
<td>0-8-4</td>
</tr>
<tr>
<td>35</td>
<td>(</td>
<td>51</td>
<td>12-8-5</td>
<td>75</td>
<td>)</td>
<td>65</td>
<td>0-8-5</td>
</tr>
<tr>
<td>36</td>
<td>+</td>
<td>45</td>
<td>12-8-6</td>
<td>76</td>
<td>&gt;</td>
<td>73</td>
<td>0-8-6</td>
</tr>
<tr>
<td>37</td>
<td>;</td>
<td>77</td>
<td>12-8-7</td>
<td>77</td>
<td>^</td>
<td>67</td>
<td>0-8-7</td>
</tr>
</tbody>
</table>

(Note: There are no 60, 62, 64, 70, 71, 74, or 75 display codes.)
<table>
<thead>
<tr>
<th>O29 BCD</th>
<th>Char.</th>
<th>Display</th>
<th>Hollerith</th>
<th>O29 BCD</th>
<th>Char.</th>
<th>Display</th>
<th>Hollerith</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>33</td>
<td>0</td>
<td>40</td>
<td>-</td>
<td>46</td>
<td>11</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>34</td>
<td>1</td>
<td>41</td>
<td>J</td>
<td>12</td>
<td>11-1</td>
</tr>
<tr>
<td>02</td>
<td>2</td>
<td>35</td>
<td>2</td>
<td>42</td>
<td>K</td>
<td>13</td>
<td>11-2</td>
</tr>
<tr>
<td>03</td>
<td>3</td>
<td>36</td>
<td>3</td>
<td>43</td>
<td>L</td>
<td>14</td>
<td>11-3</td>
</tr>
<tr>
<td>04</td>
<td>4</td>
<td>37</td>
<td>4</td>
<td>44</td>
<td>M</td>
<td>15</td>
<td>11-4</td>
</tr>
<tr>
<td>05</td>
<td>5</td>
<td>40</td>
<td>5</td>
<td>45</td>
<td>N</td>
<td>16</td>
<td>11-5</td>
</tr>
<tr>
<td>06</td>
<td>6</td>
<td>41</td>
<td>6</td>
<td>46</td>
<td>O</td>
<td>17</td>
<td>11-6</td>
</tr>
<tr>
<td>07</td>
<td>7</td>
<td>42</td>
<td>7</td>
<td>47</td>
<td>P</td>
<td>20</td>
<td>11-7</td>
</tr>
<tr>
<td>08</td>
<td>8</td>
<td>43</td>
<td>8</td>
<td>50</td>
<td>Q</td>
<td>21</td>
<td>11-8</td>
</tr>
<tr>
<td>09</td>
<td>9</td>
<td>44</td>
<td>9</td>
<td>51</td>
<td>R</td>
<td>22</td>
<td>11-9</td>
</tr>
<tr>
<td>10</td>
<td>:</td>
<td>63</td>
<td>8-2</td>
<td>52</td>
<td>null</td>
<td>60</td>
<td>11-0</td>
</tr>
<tr>
<td>11</td>
<td>#</td>
<td>74</td>
<td>8-3</td>
<td>53</td>
<td>$</td>
<td>53</td>
<td>11-8-3</td>
</tr>
<tr>
<td>12</td>
<td>@</td>
<td>71</td>
<td>8-4</td>
<td>54</td>
<td>*</td>
<td>47</td>
<td>11-8-4</td>
</tr>
<tr>
<td>13</td>
<td>'</td>
<td>70</td>
<td>8-5</td>
<td>55</td>
<td>)</td>
<td>52</td>
<td>11-8-5</td>
</tr>
<tr>
<td>14</td>
<td>=</td>
<td>54</td>
<td>8-6</td>
<td>56</td>
<td>;</td>
<td>77</td>
<td>11-8-6</td>
</tr>
<tr>
<td>15</td>
<td>&quot;</td>
<td>64</td>
<td>8-7</td>
<td>57</td>
<td></td>
<td>76</td>
<td>11-8-7</td>
</tr>
<tr>
<td>16</td>
<td>&amp;</td>
<td>67</td>
<td>12</td>
<td>60</td>
<td>space</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>A</td>
<td>01</td>
<td>12-1</td>
<td>61</td>
<td>/</td>
<td>50</td>
<td>0-1</td>
</tr>
<tr>
<td>18</td>
<td>B</td>
<td>02</td>
<td>12-2</td>
<td>62</td>
<td>S</td>
<td>23</td>
<td>0-2</td>
</tr>
<tr>
<td>19</td>
<td>C</td>
<td>03</td>
<td>12-3</td>
<td>63</td>
<td>T</td>
<td>24</td>
<td>0-3</td>
</tr>
<tr>
<td>20</td>
<td>D</td>
<td>04</td>
<td>12-4</td>
<td>64</td>
<td>U</td>
<td>25</td>
<td>0-4</td>
</tr>
<tr>
<td>21</td>
<td>E</td>
<td>05</td>
<td>12-5</td>
<td>65</td>
<td>V</td>
<td>26</td>
<td>0-5</td>
</tr>
<tr>
<td>22</td>
<td>F</td>
<td>06</td>
<td>12-6</td>
<td>66</td>
<td>W</td>
<td>27</td>
<td>0-6</td>
</tr>
<tr>
<td>23</td>
<td>G</td>
<td>07</td>
<td>12-7</td>
<td>67</td>
<td>X</td>
<td>30</td>
<td>0-7</td>
</tr>
<tr>
<td>24</td>
<td>H</td>
<td>10</td>
<td>12-8</td>
<td>70</td>
<td>Y</td>
<td>31</td>
<td>0-8</td>
</tr>
<tr>
<td>25</td>
<td>I</td>
<td>11</td>
<td>12-9</td>
<td>71</td>
<td>Z</td>
<td>32</td>
<td>0-9</td>
</tr>
<tr>
<td>26</td>
<td>null</td>
<td>60</td>
<td>12-0</td>
<td>72</td>
<td>null</td>
<td>60</td>
<td>0-8-2</td>
</tr>
<tr>
<td>27</td>
<td>.</td>
<td>57</td>
<td>12-8-3</td>
<td>73</td>
<td>,</td>
<td>56</td>
<td>0-8-3</td>
</tr>
<tr>
<td>28</td>
<td>&lt;</td>
<td>61</td>
<td>12-8-4</td>
<td>74</td>
<td>%</td>
<td>75</td>
<td>0-8-4</td>
</tr>
<tr>
<td>29</td>
<td>(</td>
<td>51</td>
<td>12-8-5</td>
<td>75</td>
<td>null</td>
<td>60</td>
<td>0-8-5</td>
</tr>
<tr>
<td>30</td>
<td>)</td>
<td>45</td>
<td>12-8-6</td>
<td>76</td>
<td>&gt;</td>
<td>62</td>
<td>0-8-6</td>
</tr>
<tr>
<td>31</td>
<td>/</td>
<td>66</td>
<td>12-8-7</td>
<td>77</td>
<td>?</td>
<td>65</td>
<td>0-8-7</td>
</tr>
</tbody>
</table>

(Note: There are no 72 or 73 display codes.)
IBM TAPE FILE RECORD AND BLOCK FORMATS

Files may be written and read in IBM 360/370 comparable format by the XREAD and XWRITE routines. Parameters are required for the XFILE routine to describe the record and blocking format to be used. These parameters, described below, correspond to the DCB (data control block) subparameters RECFM, BLKSIZE, and LRECL on a DD (data definition) statement in 360/370 JCL (job control language).

BLKSIZE

Specifies length, in number of 8-bit bytes, of the longest tape block expected, regardless of format. For variable (V) format records, the specification must include space for a 4-byte block header. The value for BLKSIZE must not exceed 32760 (decimal).
LRECL

Specifies length, in number of 8-bit bytes, of the longest logical record expected. For V format records, the specification must include space for a 4-byte record header. The value for LRECL must not exceed 32756 for V format and 32760 for fixed (F) and unspecified (U) formats. LRECL must not exceed BLKSIZE for all formats except VS and VBS.

RECFM

Describes the physical arrangement for records in blocks and in files.

Fixed Format

F

Every record is exactly LRECL bytes long. Each block contains exactly one record.
(or FS)

```
+------------------+
|                  |
|                  |
|                  |
|                  |
|                  |
+------------------+
        Block

+------------------+
|                  |
|  LRECL           |
+------------------+
        LRECL

+------------------+
|                  |
|  BLKSIZE         |
+------------------+
        BLKSIZE
```

Fixed Format

FB

Every record is exactly LRECL bytes long. Each block contains an integral number of records, total block length must not exceed (BLKSIZE).
(or FSB)

```
+------------------+
|                  |
|                  |
|                  |
|                  |
|                  |
+------------------+
        Block

+------------------+
|                  |
|  Logical Record  |
|                  |
|  Logical Record  |
|                  |
|  Logical Record  |
+------------------+
        LRECL

+------------------+
|                  |
|  BLKSIZE         |
+------------------+
        BLKSIZE
```
Variable Format

Maximum record is LRECL-4 bytes, and is prefixed by a 4-byte record descriptor word RDW as follows:

Each block contains exactly one record, prefixed by a block descriptor word BDW as follows:

VB

This format is like V, except several records may occupy the block, not to exceed the total length ≤ BLKSIZE.
S indicates spanned. This mode makes best use of blocks for variable length records and allows writing of logical records exceeding BLKSIZE (however, the record length still must be \( \leq \) LRECL \( \leq \) 32752).

A record larger than the remaining block size (\( > \) BLKSIZE-8, since both BDW and SDW are required) may be split into two or more blocks. Each record segment is prefixed by a segment descriptor word SDW of the following form:

![Segment diagram](image)

Segment code:

0 Complete logical record; for code 0, the SDW is exactly an RDW.
1 First segment of a multi-segment logical record.
2 Last segment of a multi-segment logical record.
3 Middle segment of a multi-segment logical record.

Each block contains exactly one record segment, as in format V.

VSB

This format is like VS, except several record segments may occupy a block (as in VB). At most, one segment of a record may occur in a block. An attempt is made to fill the block to BLKSIZE, it may not be successful if remaining space is 4 bytes or less (at least one data byte must be written):

![Block diagram](image)
Unspecified Format

U (or US) Undefined length records. The record may be any non-zero length, up to LRECL. Each record occupies exactly one block.

UB (or USB) Undefined length, blocked records (not an IBM record format) included here to allow non-standard record handling. Records may be any non-zero length up to LRECL, and LRECL may be less than BLKSIZE. Blocking and de-blocking is handled as follows:

**Writing:** The length of a converted record is examined for fit in the current block. If it fits, it is appended to the data already in the block. If not, the block is written out, and this record is used to start the next block.

**Reading:** If at least LRECL characters remain in the current block, they are delivered to the user. If more than zero but fewer than LRECL characters remain, they are delivered to the user.

Reading and writing are not parallel. In general, successive reads on a UB format file will not return the same records that were written.
IBM DATA FORMATS

CHARACTER DATA

EBCDIC character codes are stored in IBM systems in 8-bit bytes. Bit numbers are assigned from high to low to each bit position within a byte. By IBM standard, the numbers assigned are E0 through E7

| E0 | E1 | E2 | E3 | E4 | E5 | E6 | E7 |

ASCII character codes are stored in IBM systems in 8-bit bytes. Bit numbers 1 to 8 are assigned to each bit position within a byte from low to high order.

| A8 | A7 | A6 | A5 | A4 | A3 | A2 | A1 |

NUMERIC DATA

Eight-bit bytes are grouped in IBM systems to represent numeric data. A double byte (16 bits) is referred to as a half-word; 4 bytes comprise a whole-word (full-word); 8 bytes are a double-word.

IBM System/360 uses four forms of numeric data: fixed-point binary, floating-point hexadecimal, packed decimal, and decimal signed numeric.

FIXED-POINT BINARY

Fixed-point values can be written in half-word, full-word, or double-word format consisting of a single sign bit followed by the binary field. On occasion, these formats are referred to as signed integer format. Negative values are represented in two's complement form.

Half-word binary — 2 bytes (16 bits)

<table>
<thead>
<tr>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
</tbody>
</table>
Full-word binary – 4 bytes (32 bits)

Double-word binary – 8 bytes (64 bits)

FLOATING-POINT HEXADECIMAL

Floating-point data occupies short, long, and extended-precision formats. Each form uses the first bit as the sign of the fraction, the next 7 bits to represent a characteristic, and the remaining bits to represent the fraction expressed in hexadecimal digits. The value expressed is the product of the fraction and the number 16 raised to the power of the exponent.

Floating-point number – 4 bytes (32 bits)

Long Floating point – 8 bytes (64 bits)

Extended-precision (or double) floating-point – 16 bytes (128 bits)
Normalization

The greatest precision is achieved when a floating-point number is normalized. The fraction part of the floating-point number has a non-zero, high-order, hexadecimal digit produced by shifting the fraction left until the high-order, hexadecimal digit is non-zero and reducing the characteristic by the number of hexadecimal digits shifted. A zero fraction cannot be normalized.

Normalization applies to hexadecimal digits; thus the three high-order bits of a normalized number may be zero.

PACKED DECIMAL

Because decimal numbers may be expressed by four bits, it is possible to pack two 1-digit decimal values into one 8-bit byte. Variable length fields are used to contain packed decimal values; the rightmost 4 bits of the low-order byte of the field contain the sign of the value:

```
high-order byte

| Digit | Digit | Digit |

low-order byte

| Digit | Digit | Digit | Sign |
```

The EBCDIC sign code generated is 1101 for minus and 1100 for plus.

Packed decimal data is not the same as 8-bit character data and cannot be treated as such.

DECIMAL SIGNED NUMERIC

Also known as Zoned Decimal Number Format, this representation is required for character set sensitive I/O devices. A zoned format number carries its sign in the leftmost 4 bits of the low-order byte. The zoned format is not used in decimal arithmetic operations.

```
byte

| Zone | Digit | Zone |

byte

| Digit | Zone | Digit | Sign | Digit |
```
INTERNAL 6-BIT DISPLAY CODE

Six-bit coded information is represented in the central memory unit of the CDC system in 6-bit bytes, 10 bytes in a 60-bit word. Within a byte, information is stored:

```
  5 4 3 2 1 0
```

Within a word, 6-bit bytes are stored:

```
  59 54 48 42 36 30 24 18 12  6  0
  byte  1  byte  2  byte  3  byte  4  byte  5  byte  6  byte  7  byte  8  byte  9  byte 10
```

INTERNAL 8-BIT DISPLAY CODE DATA

The 8 bits in a string representing an ASCII code are numbered from left to right (A8 to A1) as follows:

```
A8 A7 A6 A5 A4 A3 A2 A1
```

The 8 bits in a string which represent EBCDIC code are numbered from left to right (E0 to E7) as follows:

```
E0 E1 E2 E3 E4 E5 E6 E7
```
Eight-bit data codes are represented in central memory in 12-bit bytes, right adjusted in the byte as shown below:

<table>
<thead>
<tr>
<th>bit position</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASCII bit number</td>
<td>A8</td>
<td>A7</td>
<td>A6</td>
<td>A5</td>
<td>A4</td>
<td>A3</td>
<td>A2</td>
<td>A1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EBCDIC bit number</td>
<td>E0</td>
<td>E1</td>
<td>E2</td>
<td>E3</td>
<td>E4</td>
<td>E5</td>
<td>E6</td>
<td>E7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CDC bit number</td>
<td>b1</td>
<td>b2</td>
<td>b3</td>
<td>b4</td>
<td>b5</td>
<td>b6</td>
<td>b7</td>
<td>b8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The unused 4 bits are set to zero when the byte is stored in a word and ignored when the character code is used.

Eight-bit data in ASCII or EBCDIC codes can be represented in central memory as 12-bit bytes, stored 5 bytes to a word. Character data must be aligned on the byte boundaries as shown in the following diagram:

<table>
<thead>
<tr>
<th>59</th>
<th>48</th>
<th>36</th>
<th>24</th>
<th>12</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte 1</td>
<td>byte 2</td>
<td>byte 3</td>
<td>byte 4</td>
<td>byte 5</td>
<td></td>
</tr>
</tbody>
</table>

**ARITHMETIC DATA**

**INTEGER**

Integer data is stored in a 60-bit central memory word in the following format:

```
  58
/     Integer
     
__________________ Sign (Bit 59) ________________ Binary Point ________________
```

The binary representation of the integer is right adjusted in the word. The sign is in bit 59; the binary point is at the right of bit 0. Negative numbers are represented in one's complement notation.
FLOATING POINT

Floating point data is stored in either single precision or double precision format, as follows:

**Single Precision**

<table>
<thead>
<tr>
<th>58</th>
<th>47</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Biased Exp</td>
<td>Integer Coefficient</td>
<td>Binary Point</td>
</tr>
</tbody>
</table>

The binary point is considered to be to the right of the integer coefficient, therefore the 48-bit integer coefficient is equivalent to a 14 digit (decimal) value. The sign of the coefficient is in bit 59. Negative numbers are carried in one's complement notation. The 11-bit exponent carries a bias of $2^{10}$ (2000 octal). As the coefficient is stored in unnormalized form, the bias is removed when the word is normalized for computation and restored when the word is returned to floating point format.

**Double Precision**

<table>
<thead>
<tr>
<th>58</th>
<th>47</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Biased Exp</td>
<td>Integer Coefficient</td>
<td>Most Significant Half</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>58</th>
<th>47</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Biased Exp-48</td>
<td>Integer Coefficient</td>
<td>Least Significant Half</td>
</tr>
</tbody>
</table>

In double precision format, two adjacent memory words (n and n+1) are used. The sign of the coefficient is carried in bit 59 of both words. The 96-bit integer coefficient is split, and the most significant 48 bits are stored in word n, the least significant 48 bits are in word n+1. The binary point is at the right of bit 0 in word n. Since the biased exponent of the least significant half of the coefficient is 48 less than the exponent of the most significant half, the two exponents are used to locate the true position of the binary point. If the exponent in word n represented 56, the exponent in word n+1 would be +8, indicating that the true position of the binary point is in the least significant half, 8 bits to the right of the biased exponent in word n+1. Conversely, if the exponent in word n represented 32, the exponent in word n+1 would be -16, indicating that the true position of the binary point is in the most significant half, 16 bits to the left of bit 0 in word n.
DISPLAY CODE NUMERIC SIGN OVERPUNCH

A string of display code decimal digits form a display code sign overpunch number. The sign of the number is indicated by transforming the units digit (low order) of the number as follows:

<table>
<thead>
<tr>
<th>Value</th>
<th>Units Digit</th>
<th>Transformed to:</th>
<th>Corresponds to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive</td>
<td>0</td>
<td>&gt;</td>
<td>12-0 card punch</td>
</tr>
<tr>
<td></td>
<td>1 through 9</td>
<td>A through I</td>
<td>12-1 through 12-9 punch</td>
</tr>
<tr>
<td>Negative</td>
<td>0</td>
<td>∧</td>
<td>11-0 card punch</td>
</tr>
<tr>
<td></td>
<td>1 through 9</td>
<td>J through R</td>
<td>11-1 through 11-9 punch</td>
</tr>
</tbody>
</table>

The sign overpunch numeric is CDC COBOL-defined, and insertion of the plus sign into the units digit is not automatic; therefore, all undefined, signed and positive numbers appear to have the same format. In the ∧ sign overpunch numeric, the 8-bit subroutines always will insert the plus sign into a positive value and transform the units digit.
CARD FORMATS

SCOPE

All card input to the routines must be in free-form binary format. Card input files in free-form binary must be preceded and followed by flag cards as described below. Free-form binary cards may contain 80 columns of data. Each card column is a 12-bit entity. Input may be any bit representation, as input is free-form. The internal 12-bit byte receives the card column bit pattern from each row, 12 through 9, left to right. This card representation may be converted on input to 8-bit or 6-bit character format or it may be handled as a bit representation.

Free-form binary cards do not contain sequence numbers or checksums. A card having 6/7/8/9 multipunch in column 1 and at least one punch in any other column can be read as a free-form binary card rather than as the end-of-file card it would normally be taken to be.

Free-form binary cards must be set up in a special logical record (section) having the following format: The first card must have all 12 rows punched in both columns 1 and 2; the card must not contain any other punches. This flag card is not read as containing information; it signals that free-form binary cards follow.

Any number of cards may follow; none may have the same form as the free-form flag card described above, and none may have the same form as a 6/7/8/9 end-of-file card. The free-form binary cards are read into memory in 16-word increments. The free-form binary card section must terminate with another flag card having 12 rows punched in columns 1 and 2. This card signals the end of the free-form binary deck, and indicates that standard binary or Hollerith cards will follow.

If it is necessary for a free-form binary card with the same appearance of the flag card to appear in the deck, a flag card may be created in a slightly different form. Any card having punches in all 12 rows in column 1 and all 12 rows in any other column, with no other punches on the card, is recognized as a free-form flag; therefore, 79 variations are possible.

Free-form binary cards are produced when the disposition code for the output punch file is P8 (octal 0014). A standard end-of-record card will be produced by the operating system following the last free-form binary card. The flag cards are not punched as part of the output.

KRONOS

CARD INPUT

The KRONOS system reads cards in coded and binary formats; the following conditions apply in both formats.

Card with 7/8/9 in column 1 is an EOR mark.

Card with 6/7/9 in column 1 is an EOF mark.

Card with 6/7/8/9 in column 1 is an EOI mark.

In each instance, the remainder of the card is ignored.
CODED CARDS

Cards are read in Hollerith punch code. The 3447 card reader controller converts the Hollerith code in internal BCD code and passes the data to the card reader driver. The driver converts the data from internal BCD code to display code. Up to 80 characters can be transferred per card. Trailing spaces are deleted.

Several conversion modes exist for the Hollerith punch code and all data is converted in 026 mode unless a conversion mode change is specified. The data conversion mode is changed when a card containing a 5/7/9 punch in column 1 is encountered. The conversion mode is punched in column 2. The conversion change remains in effect until another change card is read or the end of job is reached.

Literal input allows absolute binary data to be read while input is transmitted in coded mode. Cards are read (16 central memory words per card) until a card identical to a conversion card is read (5/7/9 in column 1 and 4/5/6/7/8/9 in column 2). The next card then can be the conversion mode.

The following conversion codes can be specified in column 2; these codes cannot be used with a 200 user terminal:

<table>
<thead>
<tr>
<th>Column 2</th>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>026</td>
<td>Character set equivalent in appendix A</td>
</tr>
<tr>
<td>9</td>
<td>FORTRAN 029</td>
<td>Character set equivalent in appendix A</td>
</tr>
<tr>
<td>8</td>
<td>COBOL 029</td>
<td>Character set equivalent in appendix A</td>
</tr>
<tr>
<td>8/9</td>
<td>SNOBOL 029</td>
<td>Character set equivalent in appendix A</td>
</tr>
<tr>
<td>4/5/6/7/8/9</td>
<td>Literal input</td>
<td>Card contains literal input</td>
</tr>
</tbody>
</table>

BINARY CARDS

Binary cards, denoted by a 7/9 punch in column 1, can contain up to 15 central memory words. The 3447 card reader controller reads the binary data and passes it to the card reader driver in 12-bit codes. Each card column row corresponds to a bit position. The driver checks the checksum figure when this option is specified. The driver then passes the data to the central memory buffer.

Binary card fields are:

<table>
<thead>
<tr>
<th>Columns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7/9 punch indicates a binary card</td>
</tr>
<tr>
<td></td>
<td>4 punch ignores checksum punch in column 2</td>
</tr>
<tr>
<td></td>
<td>Rows 0, 1, 2, and 3 contain the binary equivalent of the word count of the card</td>
</tr>
<tr>
<td>2</td>
<td>Binary data checksum (modulo 4095)</td>
</tr>
<tr>
<td>3 through 77</td>
<td>15 central memory words of binary data</td>
</tr>
<tr>
<td>78</td>
<td>Blank</td>
</tr>
<tr>
<td>79 and 80</td>
<td>24-bit binary card sequence number</td>
</tr>
</tbody>
</table>
SUMMARY

The punches in card column 1 are interpreted as follows by the card reader driver.

<table>
<thead>
<tr>
<th>Punch</th>
<th>Represents</th>
</tr>
</thead>
<tbody>
<tr>
<td>7/8/9</td>
<td>End of record</td>
</tr>
<tr>
<td>6/7/9</td>
<td>End of file</td>
</tr>
<tr>
<td>6/7/8/9</td>
<td>End of information</td>
</tr>
<tr>
<td>5/7/9</td>
<td>Change code conversion</td>
</tr>
<tr>
<td>7/9</td>
<td>Binary card</td>
</tr>
<tr>
<td>Not 7 and 9</td>
<td>Coded card</td>
</tr>
</tbody>
</table>

CARD OUTPUT

KRONOS punched cards can be in three formats.

- Coded (punch Hollerith)
- Binary
- Absolute binary

The following conditions apply to all three formats.

When an EOR is encountered, a card is punched with 7/8/9 in columns 1 and 80.† This card is offset.

When an EOF is encountered, a card is punched with 6/7/9 in columns 1 and 80; the remainder of the card is blank.† This card is offset.

When an EOI is encountered, a card is punched with 6/7/8/9 in columns 1 and 80; the remainder of the card is blank.† This card is offset.

The blank card preceding the deck is offset; when the card leaves the card punch, it is shifted so that it protrudes from the deck.

If a compare error is encountered, the erroneous card and the following card are offset. These two cards are repunched until no error is detected. An EOI card with 6/7/8/9 punches in columns 1 and 80 contains, in column 40, a binary count of the number of compare errors.

The system uses the following methods to punch the three card forms.

†This card cannot be submitted from a 200 user terminal.
CODED CARDS (PUNCH)

Data is retrieved from central memory until 140 characters are obtained or an end-of-line (a zero byte) is sensed. Only the first 80 characters of the line are punched, however, remaining data is lost. The data is converted from display code to internal BCD, 026 format (refer to appendix A for character set equivalences) by the card punch driver and passed to the 3446 card punch controller. The controller converts the internal BCD code to Hollerith punch code.

BINARY CARDS (PUNCHB)

The card punch driver retrieves 15 words of binary data from central memory. The driver generates a checksum for the data and issues a card number. The card punch controller receives the binary data and punches it on the card unchanged, in 12-bit codes. Each row in a card column corresponds to a bit position. The driver formats the binary card in the following manner.

<table>
<thead>
<tr>
<th>Columns</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7/9 punch denotes binary card</td>
</tr>
<tr>
<td></td>
<td>Rows 0, 1, 2, and 3 contain the binary equivalent of the word count of the card</td>
</tr>
<tr>
<td>2</td>
<td>Binary data checksum (modulo 4095)</td>
</tr>
<tr>
<td>3 through 77</td>
<td>15 central memory words of binary data</td>
</tr>
<tr>
<td>78</td>
<td>Blank</td>
</tr>
<tr>
<td>79 and 80</td>
<td>24-bit binary card sequence number</td>
</tr>
</tbody>
</table>

ABSOLUTE BINARY CARDS (P8)

Absolute binary cards are central memory images in 12-bit codes. Each row in a card column corresponds to a bit position; 16 central memory words are punched per card with no special punches or fields added.
† PRINT FORMAT — CDC 595-6 PRINT TRAIN

Information may be printed from the 6-bit Display Code format on any available printer; 8-bit data in the internal 12-bit format (leftmost 4 bits unused) must be converted to ASCII format and output to a CDC 512 Line Printer using a CDC 595-6 Print Train having the full ASCII graphic set of 95 characters. The ASCII character set appears on page 5-4.

For printing upper and lowercase data using the CDC 595-6 Print Train, the following conventions apply. Each print line consists of up to 137 12-bit characters, in ASCII code (identical to internal data type A). A maximum of 137 characters can be specified for a line, but no more than 136 will be printed. The character in position 1 is interpreted as a carriage control code. The leftmost 4 bits of each 12-bit ASCII character must be zero. Legal hexadecimal values for print characters range from 20 through 7E, space through 1F. Values outside this range cause an error condition. Refer to appendix F of the SCOPE 3.4 Reference Manual.

The entire file must be in the above format (12-bit characters) if disposition code PE is to be used. Such files must be assigned to a CDC 512 Line Printer with disposition code PE and must adhere to specific format rules as follows:

1. All characters must be in ASCII.

2. The end of a print line must be indicated by a zero byte in the lower 12 bits of the last central memory word of the line. Any other unused characters in the last word should be filled with zeros. No line should be longer than 137 characters.

3. Each line must start in the upper 12 bits of a central memory word.

4. The first character of a line is the carriage control, which specifies spacing as shown in the following table. It will never be printed, and the second character in the line will appear in the first position. A maximum of 137 characters can be specified for a line, but only 136 characters will be printed.

When FMT = 1, 2, or 3, XWRITE presets column 1 to blank, zero, or minus respectively; the record starts with column 2. Thus, only 136 characters are available in the print line.

†This appendix is not applicable to KRONOS.
## CARRIAGE CONTROL CHARACTERS

<table>
<thead>
<tr>
<th>Character</th>
<th>Action Before Printing</th>
<th>Action After Printing</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Space 1</td>
<td>Skip to top of next page†</td>
</tr>
<tr>
<td>B</td>
<td>Space 1</td>
<td>Skip to last line of page†</td>
</tr>
<tr>
<td>C</td>
<td>Space 1</td>
<td>Skip to channel 6</td>
</tr>
<tr>
<td>D</td>
<td>Space 1</td>
<td>Skip to channel 5</td>
</tr>
<tr>
<td>E</td>
<td>Space 1</td>
<td>Skip to channel 4</td>
</tr>
<tr>
<td>F</td>
<td>Space 1</td>
<td>Skip to channel 3</td>
</tr>
<tr>
<td>G</td>
<td>Space 1</td>
<td>Skip to channel 2</td>
</tr>
<tr>
<td>H</td>
<td>Space 1</td>
<td>Skip to channel 11</td>
</tr>
<tr>
<td>I</td>
<td>Space 1</td>
<td>Skip to channel 7</td>
</tr>
<tr>
<td>J</td>
<td>Space 1</td>
<td>Skip to channel 8</td>
</tr>
<tr>
<td>K</td>
<td>Space 1</td>
<td>Skip to channel 9</td>
</tr>
<tr>
<td>L</td>
<td>Space 1</td>
<td>Skip to channel 10</td>
</tr>
<tr>
<td>1</td>
<td>Skip to last line on page†</td>
<td>No space</td>
</tr>
<tr>
<td>2</td>
<td>Skip to last line on page</td>
<td>No space</td>
</tr>
<tr>
<td>3</td>
<td>Skip to channel 6</td>
<td>No space</td>
</tr>
<tr>
<td>4</td>
<td>Skip to channel 5</td>
<td>No space</td>
</tr>
<tr>
<td>5</td>
<td>Skip to channel 4</td>
<td>No space</td>
</tr>
<tr>
<td>6</td>
<td>Skip to channel 3</td>
<td>No space</td>
</tr>
<tr>
<td>7</td>
<td>Skip to channel 2</td>
<td>No space</td>
</tr>
<tr>
<td>8</td>
<td>Skip to channel 11</td>
<td>No space</td>
</tr>
<tr>
<td>9</td>
<td>Skip to channel 7</td>
<td>No space</td>
</tr>
<tr>
<td>X</td>
<td>Skip to channel 8</td>
<td>No space</td>
</tr>
<tr>
<td>Y</td>
<td>Skip to channel 9</td>
<td>No space</td>
</tr>
<tr>
<td>Z</td>
<td>Skip to channel 10</td>
<td>No space</td>
</tr>
<tr>
<td>+</td>
<td>No space</td>
<td>No space</td>
</tr>
<tr>
<td>0(zero)</td>
<td>Space 2</td>
<td>No space</td>
</tr>
<tr>
<td>-(minus)</td>
<td>Space 3</td>
<td>No space</td>
</tr>
<tr>
<td>blank</td>
<td>Space 1</td>
<td>No space</td>
</tr>
</tbody>
</table>

†The top of a page is indicated by a punch in channel 1 of the carriage control tape for the CDC 512 Line Printer. The bottom of page is channel 12.
When the following characters are used for carriage control, no printing takes place. The remainder of the line is ignored.

Q  Clears auto page eject
R  Selects auto page eject
S  Clears 8 vertical lines per inch
T  Selects 8 vertical lines per inch

PM (col 1-2)  Outputs remainder of line (up to 30 characters) on the B display and the dayfile and waits for the operator to type in the JANUS command /OKuu. n.GO allows the operator to change forms or carriage control tapes.

any other  Acts as a blank

Any pre-print skip operation of 1, 2, or 3 lines following a post-skip operation will be reduced to 0, 1, or 2 lines.

The S and T functions should be used only at the top of a page. In other positions, S and T can interfere with the stated spacing. Q and R need not be used at the top of a page, as each will cause a page eject before performing its function.
DIAGNOSTIC MESSAGES

Every diagnostic message generated by the 8-bit subroutines is printed in the dayfile and, where possible, in the job OUTPUT file as well. When the call appears to be from a FORTRAN Extended or COBOL program, appropriate traceback information is attempted. The format of the diagnostic message is as follows:

ERROR DETECTED BY xname — cause — message

xname    Name of the subroutine which detected the error
cause    Probable cause of the error
message  Diagnostic message as listed in this appendix

A typical set of error messages occurring in a FORTRAN Extended program is shown below, including appropriate traceback information, which gives the calling routine name and the source program line number.

ERROR DETECTED BY XFILE - CALLED FROM BJC83B
PARAMETER-FILE NOT DECLARED AT LINE 7

ERROR DETECTED BY XWRITE - CALLED FROM BJC83B
PARAMETER-FILE NOT SPECIFIED AS WRITE MODE AT LINE 12

ERROR DETECTED BY XMOVE - CALLED FROM BJC83B
PARAMETER-UNRECOGNIZED MOVE-COMpare TYPE AT LINE 38

ERROR DETECTED BY XWRITE - CALLED FROM BJC83B
I-O - UNRECOVERABLE ERROR ON WRITE FILE AT LINE 33
BAD OFFSET IN PARAMETER DESCRIPTOR
   Illegal bit or byte position specified.

BAD SYNTAX IN Z, S, N, OR P FIELD
   Data being converted is in illegal format.

BIT SPECIFICATION ILLEGAL FOR NON BIT FIELD
   w in i/w item locator format in a selector expression can be specified only if the T field is B

BLKSIZE EXCEEDS 32760 BYTES
   BLKSIZE parameter in the XFILE call is too big.

BLKSIZE NOT SPECIFIED
   BLKSIZE parameter in the XFILE call has been omitted.

BLOCK SHORTER THAN V-HEADER
   Input IBM record is in wrong format. Block descriptor word contents or record descriptor word contents do not agree with actual block/record size.

CONVERSION STRINGS NESTED TOO DEEPLY
   May be nested only to depth of seven levels.

DOUBLY SPECIFIED PARAMETER IN FILE STRING
   Duplicate parameter specified in the file-string parameter of the XFILE call.

EMPTY BLOCK IN VS-RECORD
   No data in input block.

FIELD EXTENDS PAST END OF RECORD
   Conversion string specifies reading or writing past the last character in the record.

FILE NOT DECLARED
   Wrong file number specified in the XFILE call or a missing file declaration on FTN program card.

FILE NOT SPECIFIED AS READ MODE
   USE parameter in the XFILE call has not been specified as R.
FILE NOT SPECIFIED AS WRITE MODE

USE parameter in the XFILE call has not been specified as W.

FILE PARAMETER IS NOT A FILE NAME

File name is misspelled in an XFILE call or does not appear in a COBOL SELECT clause.

FILE STRING DOES NOT BEGIN WITH -(-

File-string parameter in the XFILE call always must begin with a left parenthesis.

FILE STRING NOT TERMINATED WITH -)-

File-string parameter in the XFILE call always must be terminated with a right parenthesis.

FILE TYPE NOT SPECIFIED

FT specification in the file-string parameter in an XFILE call has been omitted.

FILE USAGE NOT SPECIFIED

USE specification in the file-string parameter in an XFILE call has been omitted.

FIRST CHARACTER OF CONVERSION-STRING IS NOT -(-

Conversion strings always must begin with a left parenthesis.

FIRST ITEM IN SELECTOR-EXPRESSION NOT RECOGNIZED

The itm field in the selector expression is specified incorrectly. See Table 2-1.

ILLEGAL FIRST ITEM TYPE

T in the Tm1 field is specified incorrectly. See Table 2-1.

ILLEGAL LENGTH PARAMETER DESCRIPTION

Length parameters in COBOL must be declared as COMPUTATIONAL-1.

ILLEGAL SECOND ITEM TYPE

T in the Tm2 field is specified incorrectly. See Table 2-1.

INCOMPLETE VS-RECORD AT END-OF-DATA

Data is missing from input file. Length specified in the header information of the last record does not agree with actual length.
INCONSISTENT PARAMETERS IN FILE STRING

File string parameters are inconsistent in XFILE call.

INDEFINITE SOURCE VALUE NOT REPRESENTABLE

Floating-point source item has indefinite value.

INDEFINITE VALUE FOR INTEGER DESTINATION FIELD

Item to be stored in integer destination field has indefinite value.

INFINITE SOURCE VALUE NOT REPRESENTABLE

Floating-point source item has infinite value.

INFINITE VALUE FOR INTEGER DESTINATION FIELD

Item to be stored in integer destination field has infinite value.

INTEGER VALUE TOO LARGE FOR FIELD

Receiving field does not contain enough characters to represent all digits in the source field.

INVALID DATA TYPE

Legal data types are A for ASCII, C for EBCDIC, and X for Display Code.

INVALID PARAMETER VALUE IN FILE STRING

See file-string parameters under XFILE discussion.

KEYWORD NOT FOLLOWED BY = IN FILE STRING

See file-string parameters under XFILE discussion.

LITERAL STRING IS TOO LONG

Literal strings in selector expression are limited to 80 characters.

LRECL NOT SPECIFIED

Necessary LRECL specification in file-string parameter of an XFILE call was omitted.

LRECL TOO SMALL FOR V-RECORD HEADER

Actual record length does not agree with length specified in record descriptor word (RDW).

LRECL TOO LARGE FOR BLKSIZE

If the blocking type is not spanned, LRECL must be less than BLKSIZE.
MISSING LENGTH PARAMETER

Length parameter is missing from FTN or COMPASS call to XPACK, XPAND, XMOVE, or XCOMP.

MISSING PARAMETER LIST

Subroutine called requires a parameter list.

MISSING RELATIONAL OPERATOR IN SELECTOR-EXPRESSION

Relationship must be stated between two items in a selector expression.

MISSING RIGHT PARENTHESIS

String parameter always must end with a right parenthesis.

MISSING RIGHT PARENTHESIS OR SEMICOLON

Missing punctuation in conversion string.

MISSING RIGHT STRING DELIMITER

Literal string terminator * or $ is missing.

MISSING SEPARATOR AFTER CONVERSION ITEM

Conversion item must be followed by a comma, semicolon, or right parenthesis, depending upon circumstances.

MISSING SOURCE-1 PARAMETER

Calling sequence parameter list is incomplete.

MISSING SOURCE-2 OR DESTINATION PARAMETER

Parameter list in calling sequence is incomplete.

MISSING STRING DELIMITER

Missing * or $ in a literal string.

MORE DATA AFTER RECORD IN V-UNBLOCKED FILE

Actual record length in input file exceeds that specified in record descriptor word (RDW).

MORE DATA AFTER VS-RECORD SEGMENT

Actual segment record length in the input file exceeds that specified in the segment descriptor word (SDW).
NC FILE STRING GIVEN

File-string parameter is missing in an XFILE call.

NC PARAMETERS

Necessary parameters were not specified.

NC PARAMETERS SUPPLIED TO SUBROUTINE

Called subroutine requires parameters.

NC STATUS PARAMETER

Status parameter is required.

NC WORKING STORAGE AREA PROVIDED

Workspace parameter in XFILE calling sequence is required.

NUMERIC LITERAL EXPONENT .GE. 512

Exponent value may not exceed 511.

NUMERIC LITERAL OUT OF RANGE (INFINITE)

Numeric literal has infinite value.

PARAMETER IS NOT A DATA ITEM

A literal was supplied for a data-name item in COBOL.

RECFM NOT SPECIFIED

RECFM parameter was omitted from the file-string of an XFILE calling sequence.

RELATIONAL OPERATOR NOT RECOGNIZED

Illegal relationship mnemonic specified in a selector expression.

SECOND SELECTOR-EXPRESSION ITEM NOT RECOGNIZED

The second iTm field in a selector expression is in illegal format. See Table 2-1.

SELECTOR-EXPRESSION NOT TERMINATED BY COLON

SIZE PARAMETER NOT NUMERIC TYPE

In a COBOL calling sequence to XFILE, the size parameter must be described as numeric.
SOURCE CHARACTER NOT 0 OR 1, TO BIT STRING

In a character-to-bit conversion-item, the source character may be only 0 or 1.

SOURCE EXPONENT TOO LARGE, NOT REPRESENTABLE

See Table 2-4 for limits.

STATUS RETURN NOT COMP-2

In a COBOL calling sequence, the status parameter must be declared as a COMPUTATIONAL-2 item.

STRING NOT IN NUMERIC SYNTACTIC FORM

The character string does not fit the format described in rule 13 of Table 2-4.

STRING RELATION IS NOT .EQ. OR .NE.

Only the relationships .EQ. or .NE. are legal for string value fields in selector expressions.

SYNTAX.. NO DIGIT AFTER -E- IN NUMERIC LITERAL

Numeric literals in value fields of selector expressions must fit the numeric literal definition appearing under Value Field in section 2.

TEST FIELD EXTENDS PAST END OF RECORD

The locator field specified in a selector expression begins within the record but extends beyond its logical length.

TEST FIELD NOT IN RECORD, ON LEFT

The locator field specified in a selector expression references a character preceding the first one in the logical record.

TEST FIELD NOT IN RECORD, ON RIGHT

The locator field specified in a selector expression references a character beyond the last one in the logical record.

TOO MANY DIGITS IN Z, S, N, OR P FIELD -- OVERFLOW

The magnitude of the number to be stored in the field exceeds the number of digits specified.

TOO MANY PARAMETERS

Extraneous parameters appear in the calling sequence.
UNRECOGNIZED CODE SET SPECIFIED

Legal code sets are ASCII (A), EBCDIC (C), or Display Code (X).

UNRECOGNIZED KEYWORD IN FILE STRING

A file string parameter in an XFILE call is misspelled.

UNRECOGNIZED MOVE-COMPARE TYPE

The xy parameter is incorrectly specified.

UNRECOVERABLE ERROR ON WRITE FILE

V-BLOCK HAS SHORT RECORD FRAGMENT

The actual record size is less than that specified in the record descriptor word (RDW).

V-RECORD LENGTH EXCEEDS BLOCK SIZE

The actual record size exceeds the specified block size.

V-RECORD LENGTH LESS THAN 4 BYTES

Variable records must contain 4 bytes for the record descriptor word (RDW).

VALUE TOO LARGE FOR FIELD WIDTH

Numeric value contains too many digits and/or symbols for the receiving field.

VS-RECORD FINAL SEGMENT MISSING

The final segment of a variable spanned logical record is missing from the input file.

VS-RECORD FOUND IN TYPE V FILE

A segment descriptor word (SDW) was found in a file that was not spanned.

VS-RECORD INITIAL SEGMENT MISSING

The first segment of a spanned logical record is missing from the input file.

WORKING STORAGE AREA TOO SMALL

Size of workspace buffer specified in the XFILE call is too small. See Workspace Area Size, section 3.

WSA NOT ALIGNED ON WORD BOUNDARY

The workspace parameter in a COBOL calling sequence to XFILE must have a beginning character position of 0.
The primary difference between FORM and 8-Bit Subroutines is that 8-bit subroutines have the capability of maintaining 8-bit significance in converted data; FORM does not have this capability. Other differences include conversion capabilities, management of print files, ease of usage, and minor functional differences.

No provision is made in FORM for maintaining IBM lower case characters or other characters not included in the CDC 64-character set. FORM folds any lower case character to its upper case equivalent. The 8-bit subroutines handle data in such formats as 8-bit ASCII, 8-bit EBCDIC, internal packed (7 characters per word) and unpacked (5 characters per word). Utility subroutines XMOVE, XPACK, XPAND, and XCOMP are available for processing the internal 8-bit fields.

Conversion capabilities differ slightly between FORM and the 8-bit subroutines. FORM cannot convert IBM 360/370 decimal signed numeric (zoned decimal) data or 128-bit, extended precision, floating-point data to CDC format; the 8-bit subroutines can perform such conversions. FORM has an automatic data field reformating capability (REF) that allows the user to reorder data fields and insert literals; in 8-bit subroutines, data fields in a record are reformatted sequentially only. The conversion string syntax in 8-bit subroutines is more powerful than the FORM search and item descriptors. Nested conversion strings allow the user to reformat all or part of the record in one of several ways, depending on key fields within the record. Nested conversion strings are permitted in 8-bit subroutine calls; FORM does not permit them. Conversion strings may be changed from record to record within a file at will when using 8-bit subroutines; whereas, FORM record specifications are set once per output file and may be varied from record to record only by comparing a record field to a literal quantity. This comparison is accomplished through the IDS parameter in the INP directive in conjunction with the RID parameter in the CON directive. FORM is primarily file-oriented and can be used to process random files; the 8-bit subroutines are primarily record-oriented and their use is restricted to sequential files.

Print files are handled differently by FORM than by the 8-bit subroutines. FORM allows automatic organization of print file functions such as paging, line spacing, and tilting, through the PRT directive. Also, FORM print files can contain only CDC display code characters. Print files processed by 8-bit subroutines must be organized by the user, but any character in the 95 graphic ASCII set may be used. A utility program, COPY8P, is provided in the 8-bit subroutines to print automatically IBM 360/370 print files using the CDC 595-6 Print Train.

Other minor functional differences between FORM and the 8-bit subroutines include:

FORM provides automatic sequencing of records through the SEQ directive; the 8-bit subroutines have no such provision.

FORM will process non-standard tape labels automatically using the NON directive. In the 8-bit subroutines, the calling program must process the labels.

The 8-bit subroutines will process IBM 360/370 card files as free-form binary input and output; FORM will not.
# INDEX

**Arithmetic**  
Arithmetic Data Formats D-2

**ASCII**  
ASCII Format 1-1, D-1, G-1  
ASCII Character Set A-3, A-5, A-7

**Binary**  
Binary Data Conversion 2-13  
Free-Form Binary Card Format E-1

**BLKSIZE**  
BLKSIZE Parameter 3-4, 5-1

**Block**  
IBM Tape File Record and Block Format B-1

**Card**  
Free-Form Binary Card Format E-1  
Card Files 0-2, 6-4  
DISPOSE Card 5-2, 6-2, 6-3  
FILE Card 6-3  
LDSET Card 6-4, 6-5  
REQUEST Card 6-1  
KRONOS Card Input E-1  
KRONOS Card Output E-3

**CDC**  
CDC COBOL Data Formats 7-4  
CDC COBOL Tm Values 7-5  
Sample CDC COBOL Job 7-15  
CDC FORTRAN Constants and Variables 8-3  
CDC FORTRAN Tm Values 8-4  
Sample CDC FORTRAN Job 8-4  
CDC FORTRAN Data Formats 8-3, D-1  
CDC Floating-Point Data D-3  
CDC Integer Data D-2  
Conversion - IBM to CDC Files 7-6

**Character**  
ASCII Character Set A-3, A-5, A-7  
Character Skipping 2-12  
COPY8P Character Set Restrictions 5-3  
EBCDIC Character Set A-6, A-7  
Character Folding 2-7, A-7  
Carriage Control Characters F-2

**COBOL**  
CDC COBOL Data Formats 7-4  
CDC COBOL Tm Values 7-5  
Sample CDC COBOL Job 7-15  
COBOL Examples 3-12, 3-15, 4-11, 4-14, 4-17, 7-6  
COBOL Usage 7-1  
IBM COBOL Tm Values 7-3
Code
  Internal Display Code Format  D-1
Collating
  Collating Sequence  4-5, A-3, A-4
Comparison
  Comparison Modes  2-16
  Comparison Examples  4-5
  XCOMP - String Comparison Subroutine  4-1
  8-Bit Subroutines / FORM Comparison  H-1
COMPASS
  COMPASS Examples  3-18, 4-7, 4-12, 4-15, 4-17
Constants
  CDC FORTRAN Constants and Variables  8-3
  IBM FORTRAN Constants and Variables  8-1
Control
  Carriage Control Characters  F-2
  Q Conversion Control Specification  2-5
Conversion
  Binary Data Conversion  2-13
  Q Conversion Control Specification  2-5
  Data Format Conversion  2-1, 2-13
  Data Conversion String Syntax  2-2
  Conversion Items  2-2
  Sample Item Conversion  2-3
  Conversion Specifications  2-8
  Conversion Strings  2-9
  Conversion String Punctuation  2-9
  Nested Conversion Strings  2-10
  Special Conversion Rules  2-12
  Integer Data Conversion  2-13
  Floating-Point Data Conversion  2-13
  Conversion Tm Tables  2-14
  Default Conversions  2-15
  Conversion Rules  2-17
  COPY8P Print Conversion Tables  5-4
  Conversion - IBM to CDC Files  7-6
  Conversion Strings Used as Conversion Items  2-5
  Hexadecimal / Octal Conversion  A-8
COPY8P
  COPY8P Character Set Restrictions  5-3
  COPY8P Print Conversion Tables  5-4
  COPY8P Program  5-1
  COPY8P Options  5-1

Data
  Arithmetic Data Formats  D-2
  Binary Data Conversion  2-13
  CDC COBOL Data Formats  7-4
  CDC FORTRAN Data Formats  8-3, D-1
  Data Format Conversion  2-1, 2-13
  Data Conversion String Syntax  2-2
  Integer Data Conversion  2-13
  Floating-Point Data Conversion  2-13
IBM Data Formats 8-1, C-1
IBM Floating-Point Hexadecimal Data C-2
CDC Floating-Point Data D-3
IBM FORTRAN Data Formats 8-1
CDC Integer Data D-2
Representation of 8-Bit Data D-1
Representation of 6-Bit Data D-1, D-2

Default
Default Conversions 2-15
Default Tm Values 2-15

Diagnostic
Diagnostic Error Messages G-1

Display
Display Code Numeric Sign Overpunch D-4
Internal Display Code Format D-1

DISPOSE
DISPOSE Card 5-2, 6-2, 6-3

EBCDIC
EBCDIC Format 1-1, C-1, D-1
EBCDIC Character Set A-6, A-7

Error
Diagnostic Error Messages G-1
Error Handling 3-12

Examples
COBOL Examples 3-12, 3-15, 4-11, 4-14, 4-17, 7-6
Comparison Examples 4-5
COMPASS Examples 3-18, 4-7, 4-12, 4-15, 4-17
FORTRAN Examples 3-17, 4-6, 4-11, 4-14, 4-17, 8-4
Selector Expression Examples 2-7
Input/Output Examples 3-12, 3-17, 3-18
Move Examples 4-11
Expand Examples 4-17
Pack Examples 4-14

Expand
Expand Examples 4-17

Expression
Selector Expression Examples 2-7
Selector Expression Relationships 2-7
Selector Expression Value Field 2-6

Field
Field Alignment 2-1
Selector Expression Value Field 2-6

File
IBM Tape File Record and Block Format B-1
File Name Parameter 3-1
FILE Card 6-3
XFILE - File Definition Subroutine 3-2
File Usage 3-5
Forcing Termination of File Output 3-7

Files
Card Files 6-2, 6-4
Print Files 6-2
Record Manager Print Files 6-4
Record Manager Tape Files 6-3
Tape Files 6-1
Conversion - IBM to CDC Files 7-6
File-String
File-String Parameters 3-3
Floating-Point
Floating-Point Data Conversion 2-13
IBM Floating-Point Hexadecimal Data C-2
CDC Floating-Point Data D-3
FMT
FMT Parameter 3-4, 5-3
FOLD
FOLD Parameter 5-2
Folding
Character Folding 2-7, A-7
Forcing
Forcing Termination of File Output 3-7
FORM
8-Bit Subroutines / FORM Comparison H-1
Format
ASCII Format 1-1, D-1, G-1
Free-Form Binary Card Format E-1
IBM Tape File Record and Block Format B-1
Data Format Conversion 2-1, 2-13
EBCDIC Format 1-1, C-1, D-1
565-6 Print Trajin Format G-1
Internal Display Code Format D-1
Formats
Arithmetic Data Formats D-2
CDC COBOL Data Formats 7-4
CDC FORTRAN Data Formats 8-3, D-1
IBM Data Formats 8-1, C-1
IBM FORTRAN Data Formats 8-1
FORTRAN
CDC FORTRAN Constants and Variables 8-3
CDC FORTRAN Tt Values 8-4
Sample CDC FORTRAN Job 8-4
CDC FORTRAN Data Formats 8-3, D-1
FORTRAN Examples 3-17, 4-6, 4-11, 4-14, 4-17, 8-4
IBM FORTRAN Data Formats 8-1
IBM FORTRAN Tt Values 8-2
IBM FORTRAN Constants and Variables 8-1
FT
FT Parameter 3-3, 3-4

Hexadecimal
IBM Floating-Point Hexadecimal Data C-2
Hexadecimal / Octal Conversion A-8

IBM
IBM Tape File Record and Block Format B-1
IBM COBOL Tt Values 7-3
IBM Data Formats 8-1, C-1

Index-4
IBM Floating-Point Hexadecimal Data C-2
IBM FORTRAN Data Formats 8-1
IBM FORTRAN Tm Values 8-2
IBM FORTRAN Constants and Variables 8-1
Conversion - IBM to CDC Files 7-6

Input/Output
Input/Output Examples 3-12, 3-17, 3-18
Input/Output Subroutine Parameters 3-1

Integer
Integer Data Conversion 2-13
Integer Parameters 3-1, 4-1
CDC Integer Data D-2

Interface
System Interface 6-1
Record Manager Interface 6-3
LOADER Interface 6-4

Internal
Internal Display Code Format D-1

Item
Sample Item Conversion 2-3

Items
Conversion Items 2-2
Conversion Strings Used as Conversion Items 2-5

LDSET
LDSET Card 6-4, 6-5

LOADER
LOADER Interface 6-4
LOADER Considerations 6-5

LRECL
LRECL Parameter 3-4, 5-2, B-2

Modules
Omitting Unneeded 8-Bit Modules 6-4
Move
Move Examples 4-11
XMOVE - String Move Subroutine 4-7

Notations
Notations Used in Calling Sequences 1-4
Numeric
Display Code Numeric Sign Overpunch D-4
Numeric Value Syntax 2-6

Octal
Hexadecimal / Octal Conversion A-8
Omitting
Omitting Unneeded 8-Bit Modules 6-4

Pack
Pack Examples 4-14
Index-6
Selector
Selector Expression Examples  2-7
Selector Expression Relationships  2-7
Selector Expression Value Field  2-6
Skipping
Character Skipping  2-12
Skipping Records  3-11
Specifications
Conversion Specifications  2-8
String
Data Conversion String Syntax  2-2
Conversion String Punctuation  2-9
XMOVE - String Move Subroutine  4-7
String Parameters  3-1  4-1
String Position  2-1
XPACK - String Compression Subroutine  4-12
XPAND - String Expression Subroutine  4-15
XCOMP - String Comparison Subroutine  4-1
Strings
Conversion Strings  2-9
Nested Conversion Strings  2-10
Conversion Strings Used as Conversion Items  2-5
Subroutine
XFILE - File Definition Subroutine  3-2
Input/Output Subroutine Parameters  3-1
XMOVE - String Move Subroutine  4-7
Utility Subroutine Parameters  4-1
XPACK - String Compression Subroutine  4-12
XPAND - String Expression Subroutine  4-15
XWRITE - Write a Record Subroutine  3-5
XCOMP - String Comparison Subroutine  4-1
8-Bit Subroutines / FORM Comparison H-1
XREAD/XREREAD - Read a Record Subroutine  3-8
Syntax
Data Conversion String Syntax  2-2
Numeric Value Syntax  2-6

Tables
Conversion Tm Tables  2-14
COPY8P Print Conversion Tables  5-4
Tape
IBM Tape File Record and Block Format  B-1
Record Manager Tape Files  6-3
Tape Files  6-1
Tm
CDC COBOL Tm Values  7-5
CDC FORTRAN Tm Values  8-4
IBM COBOL Tm Values  7-3
Conversion Tm Tables  2-14
IBM FORTRAN Tm Values  8-2
Default Tm Values  2-15
Usage
   COBOL Usage 7-1
   File Usage 3-5
USE
   USE Parameter 3-3, 3-4
Utility
   Utility Subroutine Parameters 4-1

Value
   Numeric Value Syntax 2-6
   Selector Expression Value Field 2-6
Values
   CDC COBOL Tm Values 7-5
   CDC FORTRAN Tm Values 8-4
   IBM COBOL Tm Values 7-3
   IBM FORTRAN Tm Values 8-2
   Default Tm Values 2-15
Variables
   CDC FORTRAN Constants and Variables 8-3
   IBM FORTRAN Constants and Variables 8-1

Workspace
   Workspace Area Size 3-5
Write
   XWRITE - Write a Record Subroutine 3-5

XCOMP
   XCOMP - String Comparison Subroutine 4-1
XFILE
   XFILE - File Definition Subroutine 3-2
XMOVE
   XMOVE - String Move Subroutine 4-7
XPACK
   XPACK - String Compression Subroutine 4-12
XPAND
   XPAND - String Expression Subroutine 4-15
XREAD/XREREAD
   XREAD/XREREAD - Read a Record Subroutine 3-8
XWRITE
   XWRITE - Write a Record Subroutine 3-5

6-Bit
   Representation of 6-Bit Data D-1, D-2

8-Bit
   Omitting Unneeded 8-Bit Modules 6-4
   Representation of 8-Bit Data D-1
   8-Bit Subroutines / FORM Comparison H-1
This form is not intended to be used as an order blank. Control Data Corporation solicits your comments about this manual with a view to improving its usefulness in later editions.

Applications for which you use this manual.

Do you find it adequate for your purpose?

What improvements to this manual do you recommend to better serve your purpose?

Note specific errors discovered (please include page number reference).

General comments:

FROM NAME: ___________________________ POSITION: ___________________________

COMPANY: ___________________________

NAME: ___________________________

ADDRESS: ____________________________

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.
FOLD ON DOTTED LINES AND STAPLE