60495500

CONTROL DATA

8-BIT SUBROUTINES VERSION 1 REFERENCE MANUAL

CDC® OPERATING SYSTEMS:
NOS 1
NOS 2
NOS/BE 1

60495500

CONTROL DATA

8-BIT SUBROUTINES
VERSION 1
REFERENCE MANUAL

CDC® OPERATING SYSTEMS:

NOS 1
NOS 2
NOS/BE 1

REVISION RECORD

Revision	Description
A (11/01/75)	Original printing.
B (05/31/78)	Revised to incorporate standard diagnostic message format; inclusion of appendix H, Glossary.
C (07/17/81)	Revised to incorporate the 8-bit subroutines handling of variable length records and various technical changes. This revision reflects PSR level 528. This is a complete reprint.

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

Address comments concerning this manual to:

CONTROL DATA CORPORATION
Publications and Graphics Division
P. O. Box 3492
SUNNYVALE, CALIFORNIA 94088-3492

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.

Page	Revision
Front Cover	-
Title Sheet	-
ii	C
iii/iv	C
v	C
vi thru viii	C
ix	C
1-1	C
1-2	C
2-1 thru 2-10	C
3-1 thru 3-11	C
4-1 thru 4-11	C
5-1 thru 5-4	C
6-1 thru 6-3	C
7-1 thru 7-6	C
8-1 thru 8-3	C
A-1 thru A-16	C
B-1 thru B-6	C
C-1 thru C-4	C
D-1 thru D-3	C
E-1 thru E-5	C
F-1 thru F-3	C
G-1 thru G-5	C
H-1	C
I-1	C
Index-1 thru -3	C
Comment Sheet/Mailer	C
Back Cover	-

PREFACE

The 8-bit subroutines are designed for use with the following operating systems:

- NOS 1 for the CONTROL DATA® CYBER 180 Series; CYBER 170 Series; CYBER 70 Models 71, 72, 73, 74; and 6000 Series Computer Systems
- NOS/BE 1 for the CDC® CYBER 180 Series; CYBER 170 Series; CYBER 70 Models 71, 72, 73, 74; and 6000 Series Computer Systems

You can use the 8-bit subroutines to convert, move, compare, pack, and unpack IBM sequential 8-bit files and CDC internal files. A utility COPY8P exists to copy IBM format print files to CDC extended print files, maintaining uppercase and lowercase characters.

Except for the COPY8P utility, all input/output operations performed by the 8-bit subroutines are under control of CYBER Record Manager.

A familiarity with COBOL or FORTRAN is assumed. A knowledge of hexadecimal notation is also assumed. A general knowledge of character sets, IBM files (record types and blocking formats), and CDC files is essential.

RELATED PUBLICATIONS

You are expected to have some familiarity with the listed publications. The publications are listed alphabetically within groupings that indicate relative importance.

The NOS manual abstracts and the NOS/BE manual abstracts are instant-sized manuals containing brief descriptions of the contents and intended audience of all NOS and NOS product set manuals, and NOS/BE and NOS/BE product set manuals, respectively. The abstracts manuals can be useful in determining which manuals are of greatest interest to a particular user.

The Software Publications Release History serves as a guide in determining which revision level of software documentation corresponds to the Programming Systems Report (PSR) level of installed site software.

The following manuals are of primary interest to users of the 8-bit subroutines:

Publication	Publication Number
COBOL Version 5 Reference Manual	60497100
COMPASS Version 3 Reference Manual	60492600
CYBER Loader Version 1 Reference Manual	60429800
CYBER Record Manager Basic Access Methods Version 1.5 Reference Manual	60495700
FORM Version 1 Reference Manual	60496200
FORTRAN Extended Version 4 Reference Manual	60497800
NOS Version 1 Reference Manual, Volume 1 of 2	60435400
NOS Version 1 Reference Manual, Volume 2 of 2	60445300
NOS/BE Version 1 Reference Manual	60493800

The following manuals are of secondary interest to users of the 8-bit subroutines:

Publication	Publication Number
NOS Version 1 Manual Abstracts	84000420
NOS/BE Version 1 Manual Abstracts	84000470
Software Publications History	60481000

CDC manuals can be ordered from Control Data Corporation, Literature and Distribution Services, 308 North Dale Street, St. Paul, Minnesota 55103.

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

APPENDIXES

INDEX

FIGURES

TABLES

NOTATIONS

The notations used in the reference formats in this manual are described as follows:

UPPERCASE — Reserved words are shown in uppercase. These words must be spelled correctly and cannot be used in a source program except as specified in the calling sequences.

lowercase — Generic terms that represent the words or symbols you supply are shown in lowercase. 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 in brackets can be omitted or included at your option.

{ } — Braces enclose two or more vertically stacked items in a calling sequence, when you can use only one of the enclosed items.

... — Ellipses that immediately follow a pair of brackets or braces indicate that the enclosed material can be repeated at your option.

:= — The colon immediately followed by an equals sign represents the phrase "as defined as".

Punctuation symbols shown in the calling sequences are required unless otherwise noted. Unless otherwise specified, all references to numbers are to decimal values.

Terms that are unique to the 8-bit subroutines, or terms that have special connotations when used with the 8-bit subroutines are defined in appendix C.

INTRODUCTION 1

You can use the 8-bit subroutines with COMPASS, COBOL, or FORTRAN to perform any of the following functions:

- Convert IBM sequential 8-bit format files to Control Data (CDC) internal format files, maintaining 8-bit significance where necessary.
- Perform data moves, comparisions, packing, and expanding of converted data in which 8-bit significance has been maintained.
- Convert CDC internal files to IBM format files.
- Copy IBM format print files to CDC extended print files, maintaining uppercase and lowercase characters.

CDC offers guidelines for the use of the software described in this manual. These guidelines appear in appendix I. Before using the software described in this manual, you are strongly urged to review the content of this appendix. The guidelines recommend use of this software in a manner that reduces the effort required to migrate application programs to future hardware or software systems.

DESCRIPTION OF 8-BIT SUBROUTINES

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 (CDC) and external (IBM) data types and character sets
- Utility subroutines that manipulate, compress, or expand character strings in display code, ASCII, or EBCDIC forms

COPY8P, a stand-alone program, provides the capability to copy an IBM format print file into a CDC-compatible print file without loss of 8-bit (uppercase and lowercase) significance.

OVERVIEW OF INPUT/OUTPUT SUBROUTINES

The input/output subroutines are as follows:

- XFILE

  Defines a file for subsequent use by the 8-bit input/output subroutines. XFILE performs no input/output operations by itself. A call to XFILE must precede a call to any of the other 8-bit input/output subroutines.

- XREAD or XREREAD

  Reads or rereads one record from an input file and places the data into the user record area. Optional data conversion can be performed during reading or rereading.

- XWRITE

  Writes one record at a time to an output file. Optional data conversion can be performed as specified.

OVERVIEW OF UTILITY SUBROUTINES

The utility subroutines include the following:

- XPACK

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

- XPAND

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

- XMOVE

  Moves character strings internally, and, optionally converts ASCII, EBCDIC, and display code during the move.

- XCOMP

  Compares two strings of like or differing character sets. The 8-bit subroutines return status information which indicates the result of the comparison.

OVERVIEW OF COPY8P UTILITY

COPY8P is unrelated to the subroutines previously described. Callable by a control statement, COPY8P copies (without loss of 8-bit significance) an IBM file to a CDC file formatted for printing on a CDC 596-6 extended print train.

CONVERSION METHODS

The 8-bit subroutines translate data via data conversion strings discussed in section 2. These conversion strings are optional parameters of the XWRITE, XREAD, and XREREAD subroutines. The user can specify data conversion between IBM data formats and CDC data formats. The IBM and CDC internal data formats that can be processed by the 8-bit subroutines are described in appendix E.

The most basic form of a conversion string is a simple item conversion. Data is converted only at this level. Other parts of the conversion string provide control information used to determine which conversions are performed.

MAINTAINING 8-BIT SIGNIFICANCE

Maintaining 8-bit significance in data converted from IBM format files is necessary when such files contain character codes not included in the CDC graphic 63- or 64-character set. Lowercase characters and several special characters such as @ ? ! and # are included in the IBM EBCDIC and ASCII character sets but not in the CDC graphic 63- or 64-character set.

Some control characters included in the IBM EBCDIC and ASCII character sets are not members of the 95-graphic ASCII character set (see appendix A) used by the CDC 596-6 print train. These characters can be processed by the 8-bit subroutines but cannot be printed on CDC equipment. You must decide the necessity of maintaining special character codes. In the following cases, 8-bit significance need not be maintained:

- Files containing only characters that appear in the CDC graphic 63- or 64-character set can be converted to CDC 6-bit display code.

- Files containing packed decimal data, in which each digit occupies 4 bits, can be converted to CDC 6-bit numeric display fields.

- Files containing binary arithmetic data can be converted to CDC binary arithmetic or display numeric data according to your specification.

- Files containing IBM arithmetic data that does not exceed CDC double-precision format need not maintain 8-bit significance to retain accuracy.

- Files containing IBM arithmetic data that does exceed CDC double-precision format can maintain 8-bit significance by using bit image conversion; however, you must provide routines to process such data.

DATA CONVERSION STRINGS 2

You can use data conversion strings as input parameters to the XREAD, XREREAD, and XWRITE subroutines. Conversion strings specify how data items in a record are to be translated. Through the use of data conversion strings you can specify:

- Conversion between IBM data formats and CDC data formats

- Conversion between CDC data formats and IBM data formats

- Selection of data to be converted dependent upon a relational test

A conversion string consists of 6-bit display code characters. With the exception of literal string parameters, blanks are ignored and can be used freely to improve readability.

OVERVIEW OF CONVERSION METHODS

A conversion item is the most basic element of a conversion string. The simplest form of a conversion item is one that specifies only one conversion; this form is called a simple item conversion. A simple item conversion causes data translation. A conversion item can also consist of a repeat count followed by a simple item conversion or a repeat count followed by a simple item conversion enclosed in parentheses. A conversion item can comprise part or all of a conversion specification, or part or all of a conversion string.

A conversion specification is composed of an optional selector expression followed by one or more conversion items. If a conversion specification contains a selector expression, a relational test is performed on the selector expression. If the result of this relational test is true, the conversion items associated with the selector expression are executed.

A conversion string can consist of one or more conversion specifications enclosed in parentheses. Since a conversion item can also be a conversion specification, a conversion string can also consist of a conversion item enclosed in parentheses. Up to seven levels of conversion strings can be nested. Each nested conversion string must be enclosed in parentheses.

Figure 2-1 shows the conversion formats. Examples are given to orient you to the conversion formats. Conversion formats are explained in greater detail later in this section.

Conversion item:=	
Simple item conversion	X20
Conversion string	(X20)
Repeat count simple item conversion	5X20
Repeat count (conversion string)	5(X20)
Conversion specification:=	
Conversion item-1	X80X80
Conversion item-1[,conversion item-2],...	5X20,X10X5
[Selector expression:]conversion item-1[,conversion item-2],...	X1 EQ A: 5X20,X10X5
Conversion string:=	
(Simple item conversion)	(X20)
([Repeat count](conversion string))	(5(X20))
(Conversion specification-1[;conversion specification-2]...)	(X1 EQ A:5x20,X10X5;X1X1)

Figure 2-1. Conversion Formats

STRING POSITION

Each record in a file is considered to be a string of variable length bytes with the length determined by the storage device used. Bytes within IBM format files contain 8 bits; bits are numbered 1 through 8, from left to right. CDC format files contain 6-bit bytes; bits are numbered 0 through 5, from right to left.

FIELD ALIGNMENT

When data conversion is initiated, internal pointers are established for the source and destination record areas, each pointer initially pointing to the first bit of the first byte of the record string. These bits are the initial next field positions. The word next has special meaning in this context.

When a next source item is converted to a next destination item, the pointers are modified as follows:

- Prior to conversion, if the bit pointer for a byte does not equal 1, the pointer 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 type B (bit) source or destination items.

- 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 in the middle of a word, the remainder of the word is not converted.

Alignment is never forced to a boundary more significant than a byte position. If word boundary or other alignment is needed, you must supply the proper fill items explicitly. Data alignment requirements are given in sections 7 and 8.

CONVERSION ITEMS

Conversion items are elements of a conversion string that provide directions for translating data items from a source record to a destination record. Conversion items have the following formats:

- [repeat count] simple item conversion

- [repeat count] conversion string

REPEAT COUNT

A decimal integer is used as the optional repeat count to indicate the number of times a conversion item is to be repeated. Using a repeat count is equivalent to writing the conversion n times, separated by commas. No repetition occurs if the repeat count is one 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. Only a simple item conversion specification causes data to be converted. Other parts of the conversion string provide control information and determine the kinds of conversions to be performed.

The format of a simple item conversion is as follows:

$Tm_1[Tm_2]$

T A one-character mnemonic code indicating the data type of the data field.

m A decimal integer specifying the length in bytes (bits if a type B item) of the data item; m must be omitted for H, W, G, F, L, E, I, U, and D data types. A default of 1 is assumed if m is omitted for data types B, X, P, S, N, and Z.

Valid values for T and m are shown in table 2-1. Data type X can be any member of the character set in use. S, N, Z, and P data types are subsets of X and can contain numeric characters. These data types, along with type B, describe variable length data fields; m indicates the length of the field. All other data types describe fixed length fields; m cannot be used with these data types.

If the Tm_1Tm_2 format is used, Tm_1 must be a valid item descriptor type for the source medium, which can be a tape file, internal file, or a card file; Tm_2 must be a valid item descriptor type for the destination medium, which can be a tape file, internal file, print file, or card file. If the Tm_1 format is specified with Tm_2 omitted, a default value is selected for Tm_2 as specified in table 2-2. A simple item conversion causes data to be moved from the source field to the destination field and translated to the destination field format. Translation rules for all possible simple item conversions appear in appendix G.

Some examples of simple item conversions are shown in table 2-3.

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; the relational test of the selector expression must be true for the associated conversion items to be executed. If the selector expression is omitted, the conversion specification is treated as though it was preceded by a selector expression with a relational test that is always true. Conversion items in a conversion specification are executed in sequence, from left to right. The format of a conversion specification is as follows:

[selector expression:]conversion $item_1$
[,conversion $item_2$]

TABLE 2-1. VALID T AND m VALUES

T	Description	m[†]
IBM Format:[††]		
B	Bits	Size of field in bits
X	8-bit characters	Size of field in 8-bit characters
H	Half-word (16-bit) integer	--
W	Whole-word (32-bit) integer	--
G	Double-word (64-bit) integer	--
F	Floating-point (32-bit)	--
L	Long floating-point (64-bit)	--
E	Extended-precision floating-point (128-bit)	--
P	Packed decimal (IBM COMP-3 COBOL items)	Size of field in 8-bit bytes
S	Decimal signed numeric	Size of field in 8-bit bytes
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 (ASCII)	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)	--
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

[†] A double dash (--) indicates m must be omitted.

[††] IBM format ASCII and EBCDIC 8-bit sequential tapes can include all these data items.

TABLE 2-2. DEFAULT CONVERSIONS IF Tm$_2$ IS NOT SPECIFIED

Tm$_1$	Default Tm$_2$	
Tape:	Internal:	
Bm	Bm	
Xm	Xm	
H	I	
W	I	
G	I	
F	E	
L	E	
E	D	
Pm	Sn (n=2m)	
Sm	Sm	
Card†:	Internal:	
Bm	Bm	
Xm	Xm	
Internal:	Tape:	Card /Print:
Bm	Bm	Bm
Xm	Xm	Xm
Am	Xm	Xm
Cm	Xm	X
I	W	X20
U	F	X20
E	F	X20
D	L	X20
Sm	Sm	Xm
Nm	Sm	Xm
Zm	Xm	Xm

†The term card refers to 80-column binary cards (literal input or absolute binary for NOS; free-form binary for NOS/BE)

Figure 2-2 shows the elements of a conversion specification. Examples of conversion specifications are shown in table 2-4.

SELECTOR EXPRESSIONS

A selector expression appears in a conversion specification to indicate that a relational test is to be made. One attribute of one data field is tested. If the result of the test is true, all conversion items associated with the selector expression are executed. If the result is false, the associated conversion items are not executed.

A selector expression is composed of a relational operator that is preceded by an item locator. The relational operator is followed by either another item locator or by a value field. A selector expression is separated from its associated conversion items by a colon. The scope of a selector expression is a single conversion specification which is terminated by a semicolon. The format for selector items and the associated relational operators are shown in figure 2-3.

TABLE 2-3. EXAMPLES OF SIMPLE ITEM CONVERSIONS

Example	Explanation
$\underbrace{\text{X5}}_{\text{Tm1}}\underbrace{\text{X5}}_{\text{Tm2}}$	Translates five 8-bit characters to five 6-bit internal display code characters.
$\underbrace{\text{X120}}_{\text{Tm1}}\underbrace{\text{X150}}_{\text{Tm2}}$	Translate a 120 character field to a 150-character field with blanks as fill on the right.
$\underbrace{\text{X4}}_{\text{Tm1}}\underbrace{\text{H}}_{\text{Tm2}}$	Translates four display code characters to an IBM half-word integer.
$\underbrace{\text{G}}_{\text{Tm1}}\underbrace{\text{U}}_{\text{Tm2}}$	Translates one IBM 64-bit integer to a 60-bit word containing a CDC unnormalized floating-point number.
$\underbrace{\text{B60}}_{\text{Tm1}}\underbrace{\text{I}}_{\text{Tm2}}$	Translates a 60-bit stream on tape to an internal 60-bit integer field.
$\underbrace{\text{60}}_{\text{repeat count}}\underbrace{\text{B6}}_{\text{Tm1}}\underbrace{\text{B10}}_{\text{Tm2}}$	Moves 60 consecutive internal 6-bit fields to consecutive 10-bit fields on tape. Each 10-bit destination field will contain 4 bits of binary zero fill on the right.
$\underbrace{\text{4}}_{\text{repeat count}}\underbrace{\text{X10}}_{\text{Tm1}}\underbrace{\text{P6}}_{\text{Tm2}}$	Translates four consecutive 10-character display code fields to 4 consecutive 6-byte packed decimal fields.
$\underbrace{\text{6}}_{\text{repeat count}}\underbrace{\text{E}}_{\text{Tm1}}\underbrace{\text{F}}_{\text{Tm2}}$	Translates six CDC floating-point numbers to six IBM floating-point numbers
$\underbrace{\text{B60}}_{\text{Tm1}}$	Moves a 60-bit field to a 60-bit field.
$\underbrace{\text{U}}_{\text{Tm1}}$	Translates a 60-bit internal CDC unnormalized floating-point number to an IBM 32-bit floating-point field.
$\underbrace{\text{X5}}_{\text{Tm1}}$	Translates five characters to five characters.
$\underbrace{\text{40}}_{\text{repeat count}}\underbrace{\text{P5}}_{\text{Tm1}}$	Tranlates 40 consecutive IBM packed decimal fields to internal CDC 10-digit signed overpunch numeric 6-bit display code fields.

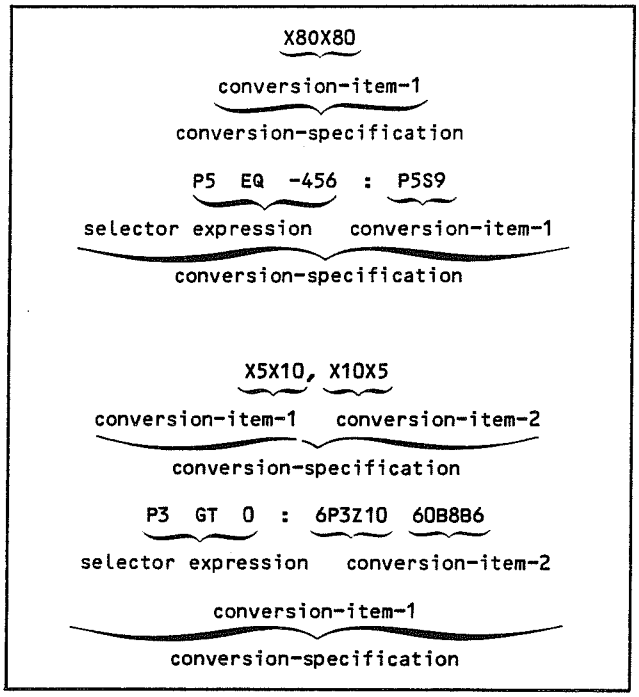


Figure 2-2. Elements of a Conversion Specification

TABLE 2-4. CONVERSION SPECIFICATION EXAMPLES

Example	Explanation
X5X6	Moves five characters from a source file to a six-character field. The rightmost character is a blank.
X5	Converts five 8-bit EBCDIC characters to five 6-bit display code characters.
X1 EQ A: X5X6	Moves five characters from a source file to a six-character field if the first character on the tape is an A.
3X1 EQ C: X5X6	Moves five characters from a source file to a six-character field if the third character in the source record is a C.

Item Locators

Item locators specify which data fields in the current source record are used in the relational test. Item locators describe data fields in terms of starting position within the field, data type, and data size.

Format:

item-locator-1 relational operator value field

item-locator-1 relational operator item-locator-2

Relational operators:

LE	less than or equal to
LT	less than
EQ	equal to
NE	not equal to
GT	greater than
GE	greater than or equal to

Figure 2-3. Selector Expression Format

The item locator formats are:

Tm iTm i/wTm

i A byte index denoting the initial byte position of the data field; i can be interpreted as an absolute index, or as a relative index.

As an absolute index, i denotes a byte position relative to the beginning of the field and must be written as an unsigned decimal integer.

As a relative index, i denotes a byte position relative to the current byte and must be written as a signed decimal integer. The sign indicates the direction of the move (+ forward, - backward). The initial current byte is the current next byte position. If i is omitted, the index is assumed to be positive zero, designating the current byte.

i/w A byte and bit index separated by a slash. The i is a byte index as described previously and the 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 size of the byte in the source medium. The maximum value for w is 6 for CDC data stored internally, 8 for IBM format data, and 12 for binary data on cards.

T A one-character mnemonic code indicating the item data type specification. T can be any value representing a legal data type for the record media.

m   A decimal integer specifying the size in bytes (12-bit, 8-bit, 6-bit, 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. A default of 1 is assumed if m is omitted for a variable length data item.

Tables 2-1 and 2-2 contain allowable and default values for T and m. Table 2-5 shows examples of item locators.

TABLE 2-5. EXAMPLES OF ITEM LOCATORS

Locator Format	Example	Description
Tm	X10	A 10-byte character field
	B4	A 4-bit string
	S4	A 4-character signed numeric field
iTm	3X4	A 4-byte character field, starting at byte 3
	2N4	A 4-character numeric (leading zeros changed to blanks) field, starting at byte 2
	4Z4	A 4-character numeric (leading zeros changed to blanks) field, starting at byte 4
i/wTm	16/4B2	A 2-bit binary field, starting at the fourth bit position of byte 16

Value Fields

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

Literal Strings

A literal character string is written in the value field as a literal enclosed in identical delimiter characters. You can use the characters $ and * interchangeably as delimiters. These characters are not considered part of the string. Blanks are valid within literal character strings. If a literal contains one of the delimiting characters, you should use the other delimiting character as the delimiter. If the delimiting character must be part of the string, you must double each enclosed occurence of the delimiter in the literal. Assume the character string ABC*DEF. The character string literal can be represented by $ABC*DEF$ or by *ABC**DEF*.

A literal must not exceed 80 characters, excluding delimiters, and can be composed of any character in the display code character set. Table 2-6 shows examples of character strings and their corresponding literals.

TABLE 2-6. CHARACTER STRINGS AND LITERALS

Character String	Literal	Literal String Count
BYE	*BYE*	3
$10.20	$$$10.20$	6
*	****	1
HI*$HO	$HI*$$HO$	6

Numeric Values

Numeric values are 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 shown in figure 2-4.

```
┌ ±     d           ┐
│                   │
│ ±     d E ± exp   │
└                   ┘
where:

   ±d       Is in the form of a floating-point
            or integer constant:

               d
               d.d
               .d

   exp      Is an unsigned integer exponent
```

Figure 2-4. Numeric Field Format

Numeric values can be expressed in any of the following forms:

±n   ±n.n   ±.n   ±n.E±s   ±n.nE±s   ±.nE±s

The numeric value is represented by n, and the value of the exponent is represented by s. If E is present, the decimal point must also appear. The omission of a positive or negative sign implies a positive value or exponent. Examples of numeric values are shown in table 2-7.

COMPARISON MODES FOR SELECTOR EXPRESSIONS

Before you can perform a comparison between the elements in the selector expression, both elements must be reduced to a common mode: either a character string or a numeric value. Perform the comparison according to the procedures outlined in tables 2-8 and 2-9.

TABLE 2-7. NUMERIC FIELD EXAMPLES

Numeric Field	Represents
0	0
2.5	2.5
-10	-10
425.E6	425000000
-818.62E3	-818620.0
.57E-10	.000000000057

TABLE 2-8. IBM FORMAT COMPARISON MODES FOR SELECTOR EXPRESSIONS

Item-locator-1 Data Type	Item-locator-2 Data Type			
	String Literal	Numeric Literal	B,X	H,W,G,F, L,E,P,S
BX	string mode	numeric mode	string mode	numeric mode
H,W,G,F, L,E,P,S	numeric mode	numeric mode	numeric mode	numeric mode

TABLE 2-9. CDC FORMAT COMPARISON MODES FOR SELECTOR EXPRESSIONS

Item locator-2 Data Type	Item-locator-2 Data Type			
	String Literal	Numeric Literal	B,X	I,E,U,D, S,N,Z
B,X	string mode	numeric mode	string mode	numeric mode
I,E,U,D, S,N,Z	numeric mode	numeric mode	numeric mode	numeric mode

Numeric quantities are kept to an accuracy of at least 96 bits. Character strings of data types A and C are folded to 6-bit form for comparison against data type X character strings (6-bit). Folding is the process of mapping more than one source character to a single destination character. Uppercase and lowercase alphabetic characters are folded to a single uppercase character set. Appendix A provides the translations for display code, EBCDIC, and ASCII characters that occur during folding. Shorter strings are treated as if they were extended on the right with blanks so that both strings are the same length. (Exception: a string deriving from a bit string is extended with zeros.)

Examples of selector expressions are shown in table 2-10.

TABLE 2-10. SELECTOR EXPRESSION EXAMPLES

Expression	Explanation
X6 EQ $ABCDEF$	A string of six 8-bit characters on a tape file are compared with the literal ABCDEF.
L LT -4.67E+02	The long floating-point field beginning at the current byte of the record is compared with the numeric value -467.0.
6X1 EQ 10X1	One 8-bit character in byte 6 of the record is compared to one character in byte 10 of the record.
+4W EQ -8G	A full-word integer starting 4 bytes beyond the current byte position is compared with the double-word integer starting 8 bytes before the current byte.
6/4B2 EQ 10	A 2-bit field starting with bit 4 of byte 6 is compared with the literal value 10_2.

CONVERSION STRINGS

A conversion string is enclosed in parentheses and consists of one or more alternative conversion specifications. If a conversion string containing a selector expression is encountered during execution, a relational test is performed on the selector expression. The selector expression of each conversion specification is evaluated in sequence from left to right, until the value of the relationship of a selector expression is determined to be true. This conversion specification is executed. None of the following conversion specifications contained in the conversion string are executed.

Figure 2-5 shows elements of a conversion string. Conversion string examples are shown in table 2-11.

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 terminated by a semicolon. This feature is of special significance when items are nested in parentheses in a conversion string.

NESTED CONVERSION STRINGS

You can use a conversion string wherever a simple item conversion is allowed in a conversion item. You can nest conversion strings within conversion items to a maximum depth of seven levels.

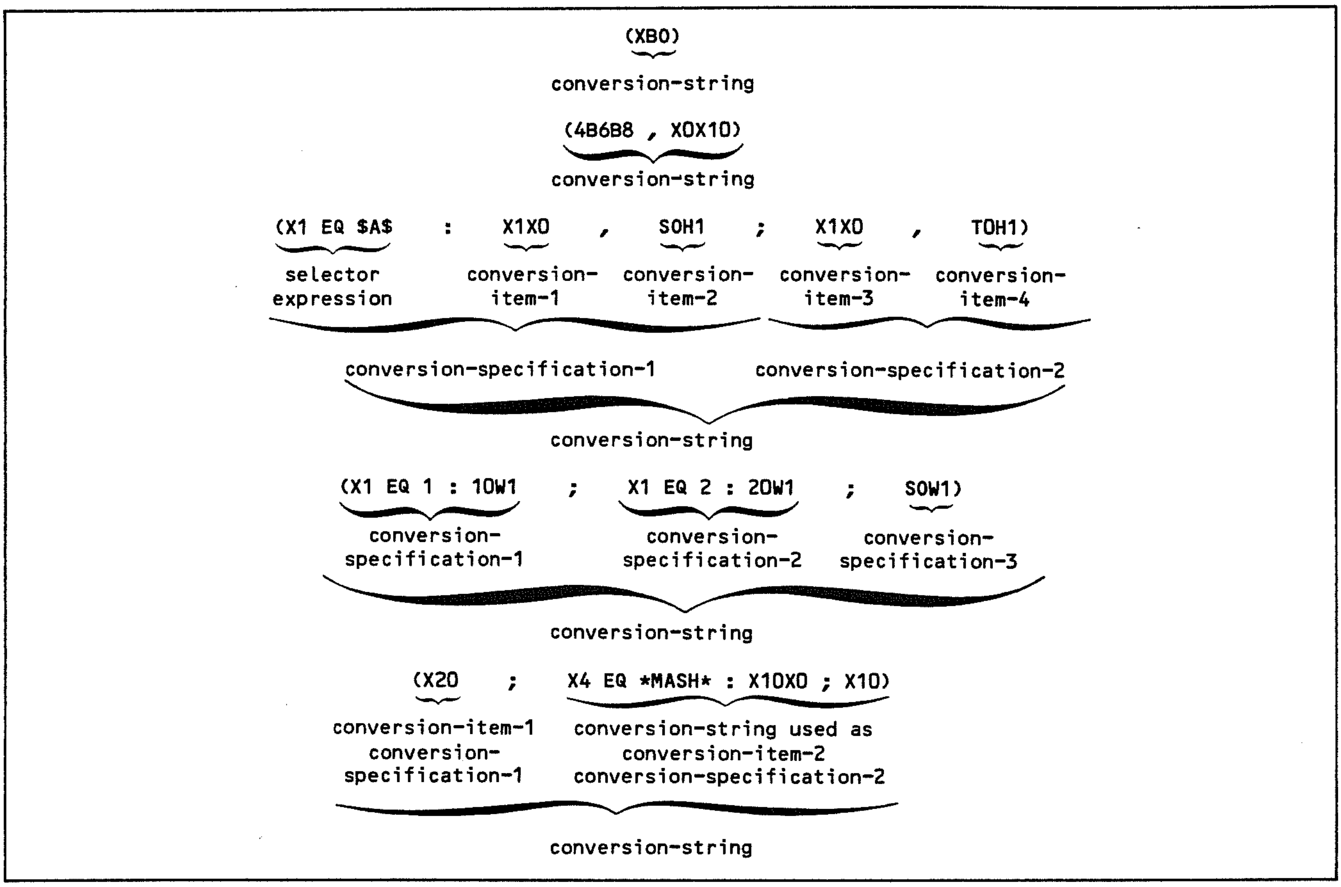

(XBO)
conversion-string
(4B6B8 , XOX10)
conversion-string
(X1 EQ A : X1XO , SOH1 ; X1XO , TOH1)
selector expression
conversion-item-1
conversion-item-2
conversion-item-3
conversion-item-4
conversion-specification-1
conversion-specification-2
conversion-string
(X1 EQ 1 : 10W1 ; X1 EQ 2 : 20W1 ; SOW1)
conversion-specification-1
conversion-specification-2
conversion-specification-3
conversion-string
(X20 ; X4 EQ *MASH* : X10XO ; X10)
conversion-item-1
conversion-specification-1
conversion-string used as
conversion-item-2
conversion-specification-2
conversion-string

Figure 2-5. Elements of a Conversion String

Nesting conversion strings allows you to specify alternate conversions at various positions within a record. Nested conversion strings are useful when converting a file with records containing fixed initial data followed by a variable format segment. Conversion string nesting is also useful when a conversion string alternative can itself have other conversion string alternatives. Assume the following conversion string

```
    (X3C3,(X1 EQ $D$ :X1C1;X3C3),X5C5)
```

The first three characters of the source record are moved. If the fourth character is D, the next character of the source record is moved to the destination record. The rest of the conversion specification up to the matching right parenthesis is skipped. Conversion for that entry to the conversion string is completed when the next five characters are moved to the destination record. If the fourth character in the source record is not D, the conversion item associated with the selector expression is skipped and the next three characters are moved to the destination record. Conversion for that entry to the conversion string is completed when the next three characters are moved to the destination record. Selection of the alternative conversion specification to be executed occurs separately for each entry to the conversion string.

CONVERSION TERMINATION

The Q (Quit) specification is an optional control code. Execution of a Q specification terminates all conversion for a record. You can insert this specification wherever a conversion item is valid. Conversion continues up to the point where the Q specification is encountered. No further conversion takes place after the Q specification.

SPECIAL CONVERSION RULES

Rules pertaining to all possible conversions appear in appendix G. Some general principles are described in the following paragraphs.

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 from left to right. Each zero bit is translated to the character 0 and each one bit to the character 1.

TABLE 2-11. CONVERSION STRING EXAMPLES

Example†	Explanation
(X30X30) (30X1X1) (30X1) (X30)	Thirty EBCDIC 8-bit characters are converted to 30 characters of 6-bit display code.
(X5A5)	Five 8-bit EBCDIC characters are converted to five 12-bit ASCII characters.
(X1 EQ *A*:X5C5)	Five characters are moved from the 8-bit EBCDIC tape record to a 12-bit EBCDIC internal format record if the first character on the tape is A.
(X1 EQ 1:X5C5;X1 EQ *A*:C3)	One character on the tape record is compared with the literal 1. Since the value of the comparison is false, the X5C5 conversion item is ignored. The value of the second selector expression is true; therefore, the characters ABC are moved.
(3X1 EQ *C*:X5C5)	Five characters are moved from tape to an internal EBCDIC format field if the third character in the source record is C.
(X3C3,(-3X1 EQ A:X23C23)	Three characters on the tape record 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.

†The examples assume a 9-track tape containing only 8-bit EBCDIC characters as the input source to be converted to internal CDC format. The tape contains multiple repetitions of the alphabet in uppercase only.

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.

STRING TO NUMERIC

A literal string of an X, C, or A item compared to a numeric item must conform to the rules for a numeric value described in this section. An error results 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 the destination field size in the conversion items. The conversion item B10B0 causes 10 bits to be skipped; X5X0 causes five characters to be skipped. The conversion item X10X5 causes five characters to be transferred and the next five 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. The conversion item X0C5 causes five EBCDIC spaces to be placed in the destination field. X10X100 causes 10 characters to be transferred to the destination field with 90 blanks on the right.

FLOATING POINT TO INTEGER

Conversions are possible between the valid formats listed in table 2-1 within the restrictions for each conversion noted in appendix G, such as conversions between IBM floating-point formats of 32, 64, and 128 bits and the CDC floating-point and double-precision floating-point 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.

BINARY DATA

Any data can be considered binary and manipulated on a bit-by-bit basis. You can copy bits in strings, or you can copy bits selectively, by skipping bits or replacing groups in a string with zeros.

You can convert bit strings to any other valid format within the limitations shown in appendix G.

Item locators in selector expressions can address any bit in a character or bit string. The item locator 2/5B1 references the fifth bit of the second byte in the source record. The conversion string (2/5B1 EQ 1: X1X0,X1C1) translates the first character to the internal record only if the fifth bit of the second byte 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 for 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 the internal format are limited to 6-bit bytes; this reduces the number of problems that would otherwise be encountered in other comparisons.

INPUT/OUTPUT SUBROUTINES 3

You can use the subroutines XFILE, XREAD, XREREAD, and XWRITE described in this section for the following:

- Reading or writing 9-track IBM sequential tape files

- Reading or punching 80-column binary cards

All input and output is through CYBER Record Manager (CRM), but records are neither converted nor blocked/deblocked at this level. Accordingly, special conventions are necessary for CRM when using the input/output subroutines. The special CRM conventions are discussed in section 6.

XFILE SUBROUTINE

You must use the XFILE subroutine to define a file to be processed by the input/output subroutines. XFILE performs no operations on the file; however, XFILE defines workspace and other specific information for the file. A calling sequence to the XFILE subroutine must precede the first reference to the file by any of the other input/output subroutines.

The workspace defined by XFILE cannot be used simultaneously for any other purpose. The workspace can be freed at the your discretion, but no other references to the file can be made until the file is redefined by another XFILE calling sequence.

FORTRAN EXTENDED 4 XFILE CALLING SEQUENCE

The FORTRAN Extended 4 XFILE calling sequence is used to define a file to be processed by the input/output subroutines. The FORTRAN Extended 4 XFILE calling sequence is shown in figure 3-1.

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

Figure 3-1. FORTRAN Extended 4 XFILE Calling Sequence

The FORTRAN XFILE parameters are as follows:

- file

  Specifies the file (file name) to be associated with the workspace area. You can write the file parameter as a tape number or as the logical file name left-justified with zero fill.

- workspace

  Specifies the working storage area to be used by the input/output subroutines. You must write the workspace parameter as an array name that is dimensioned or equivalenced to satisfy workspace size requirements as determined by the size parameter.

- file-string

  Specifies required file information in keyword form. The file-string parameter must be enclosed in parentheses and contain the file string keywords and parameters shown in figure 3-2. For FORTRAN Extended 4 the file-string parameter must be written as a variable, an array name, or a left-justified Hollerith constant.

  For tape and print files, three keywords with parameters are required as follows:

  (FT=ft,USE=use,RECFM=form,opt_1,...,opt_n)

  For 80-column binary card files, two keywords with parameters are required as follows:

  (FT=ft,USE=use,opt_1,...,opt_n)

- size

  Specifies the size of the workspace area to be used. You can write the size parameter as an integer constant, variable, or expression. The size of the workspace area in central memory words is determined by the file type parameter of the file string. Figure 3-3 shows the method of determining the workspace area size.

File Usage

Files written or read by the 8-bit subroutines cannot be processed subsequently by normal read or write functions in the same FORTRAN Extended 4 program. The 8-bit subroutines alter the CYBER Record Manager file information table (FIT), and the original information contained in the FIT for the file is not viable for normal FORTRAN input or output.

FORTRAN Extended 4 XFILE Example

An example of a FORTRAN Extended 4 XFILE call is shown in figure 3-4. Ten 8-bit EBCDIC characters are to be moved from a tape file. Before any reference to the XREAD or XWRITE subroutines, a call to the XFILE subroutine must appear in the program.

(FT=ft,USE=use,RECFM=form,BLKSIZE=nnnn,LRECL=nnnn,CODE=set,FMT=f)

The FT, USE, and RECFM keywords and parameters are required for tape and print files. Only the FT and USE keywords and parameters are required for card files.

FT=ft File type keyword; ft can have the values:

T	IBM format tape file
P	Print file with extended character set print train
C	Card file (read or punch)

USE=use File usage keyword; use can have the values:

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

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

The following keyword is required for tape and print files. The parameter defines the IBM record/block format used.

RECFM=form Record format keyword (omitted for card files); form can have the values:

F	Fixed format
FB	Fixed blocked format
FS	Fixed spanned format
U	Unspecified format
UB	Unspecified blocked format
US	Unspecified spanned format
USB	Unspecified spanned blocked format
V	Variable format
VB	Variable blocked format
VS	Variable spanned format
VSB	Variable spanned blocked format

Appendix D contains more information about IBM record/block formats.

The following keywords and parameters are required for tape files:

BLKSIZE=nnnn Block size measured in 8-bit bytes. The value of BLKSIZE can range from 1 to 32767.

LRECL=nnnn Record size of the longest logical record, measured in 8-bit bytes. Required for blocked tape files (FB, UB, UBS) and variable formats (VS, VB, VSB). The value for LRECL can range from 1 to 32760.

The following parameters are optional:

CODE=set External code set definition keyword; set can have the value:

A	ASCII
C	EBCDIC (default)

FMT=f Print format keyword; f can have the values:

1	Single space
2	Double space
3	Triple space
A	Record character 1 is used as a carriage control (default)

RECFM, BLKSIZE, and LRECL have the same meaning as their IBM job control counterparts, and generally these parameters should be copied from the associated IBM data definition statement. Appendix E includes a description of IBM data formats.

Figure 3-2. File String Keywords and Parameters

Workspace area, specified in units of central memory, number of words is determined by the file type parameter in the file string. The following sizes are possible:

FT=C size=20 (required)

FT=P size=32 (required)

FT=T Workspace area is determined by the record format in the file string, as shown:

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

$$size=6+\left[\frac{BLKSIZE}{7.5}\right] \text{ words}$$

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

$$size=6+\left[\frac{LRECL}{7.5}\right]+\left[\frac{BLKSIZE}{7.5}\right]\text{words}$$

Figure 3-3. Determination of Workspace Size

```
  .
  .
  .
  CALL XFILE(1,WS1,"FT=T,USE=W,RECFM=VSB,
+           BLKSIZE=100,LRECL=150,CODE=C)",40)
  .
  .
  .
  CALL XWRITE(WS1,NEWRAY,"(C10,X10)",STAT)
  .
  .
```

Figure 3-4. FORTRAN Extended 4 XFILE Example

COBOL XFILE CALLING SEQUENCE

The COBOL XFILE calling sequence is used to define a file to be processed by the input/output subroutines. The COBOL XFILE calling sequence is shown in figure 3-5.

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

Figure 3-5. COBOL XFILE Calling Sequence

The COBOL XFILE parameters are as follows:

- file

  Specifies the file (file name) to be associated with the workspace area.

- workspace

  Specifies the data working area to be used by input/output procedures. You must write the workspace parameter as a data item name which begins on a word boundary (synchronized or 01 level item).

- file-string

  Specifies file information in keyword form. The file-string parameter must be enclosed in parentheses and contain the file string keywords and parameters shown in figure 3-2. You can write the file-string parameter as a data name or as a COBOL alphanumeric literal constant.

  For tape and print files, three keywords with parameters are required, as follows:

  (FT=ft,USE=use,RECFM=form,opt_1,...,opt_n)

  For 80-column card files, two keywords with parameters are required as follows:

  (FT=ft,USE=use,opt_1,...,opt_n)

  Figure 3-2 lists the file string parameters.

- size

  Specifies the size of the workspace area to be used. You must write the optional size parameter as a COMPUTATIONAL-1 item consisting of no more than 14 digits. The COBOL default value for size is the length of the workspace data item in central memory words. The size of the workspace area in words is determined by the file type parameter of the file string. Figure 3-3 shows the method of determining the workspace area size.

File Usage

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

COBOL XFILE Example

A file TT9, containing only 30-character EBCDIC coded records (no binary data), is to be read with a COBOL program. Before the file TT9 can be processed, the file must be opened and the XFILE routine called. This operation appears in the Procedure Division as shown in figure 3-6.

```
DATA DIVISION.
  .
  .
  .
WORKING-STORAGE SECTION.
01  FILEX-PARAMETERS.
    02  SZ        PIC 999 USAGE IS COMP-1 VALUE 46.
    02  FSTRING  PIC X(60)     VALUE "(FT=T, USE=R,
        "RECFM=FB, BLKSIZE=150, LRECL=30, CODE=C)".
  .
  .
  .
    02  WS1       PIC X(460)    VALUE SPACES.
  .
  .
  .
PROCEDURE DIVISION.
  .
  .
  .
    OPEN INPUT TT9.
    ENTER COMPASS XFILE USING TT9, WS1, FSTRING,
      SZ.
```

Figure 3-6. COBOL XFILE Example

The file type must be specified in the XFILE file-string parameter list, along with usage, record format, and block size. These parameters should be copied from the data definition statement in the IBM Job Control Language, if this statement is available. Data Division entries in the COBOL program appear as shown in figure 3-6.

COMPASS XFILE CALLING SEQUENCE

The COMPASS XFILE calling sequence is used to define a file to be processed by the input/output subroutines. The COMPASS XFILE calling sequence is shown in figure 3-7.

```
           SA1          plist
           RJ           XFILE
           .
           .
           .
plist      VFD          42/0,18/file
           VFD          42/0,18/workspace
           VFD          42/0,18/file-string
           VFD          42/0,18/size
           BSSZ         1
```

Figure 3-7. COMPASS XFILE Calling Sequence

The COMPASS XFILE parameters are as follows:

- plist

  Specifies the symbolic location of the parameter list; plist must be terminated by a word of binary zeros.

- file

  Specifies the file (file name) to be associated with the workspace area. The file parameter must be the symbolic location of the word containing the identification of the file to be associated with the workspace area. You can write the file parameter either as a constant (in the format nLstring or nZstring) or as the address of a word containing the file name in display code, left-justified and zero-filled.

- workspace

  Specifies the working storage area to be used by the input/output subroutines. You must write the workspace parameter as the symbolic location of the workspace area. The size of the area (determined by the size parameter) must be sufficient to meet the requirements of the character strings to be processed.

- file-string

  Specifies the required file information in keyword form. The file-string parameter must be enclosed in parentheses and contain the file string keywords and parameters shown in figure 3-2. You can write the file-string parameter as the symbolic location of a left-justified character data string in the format nLstring, nHstring, or nZstring.

  For tape and print files, three keywords with parameters are required as follows:

  (FT=ft,USE=use,RECFM=form,opt$_1$,...,opt$_n$)

  For 80-column binary card files, two keywords with parameters are required as follows:

  (FT=ft,USE=use,opt$_1$,...,opt$_n$)

- size

  Specifies the size of the workspace area to be used. The parameter size must specify the workspace area size in number of central memory words. You can write the size parameter either as an integer constant or as the address of a word containing an integer value. Figure 3-3 shows the method of determining the workspace area size.

An example of a COMPASS XFILE calling sequence is shown in figure 3-8.

XWRITE SUBROUTINE

The XWRITE subroutine takes data from a record area in memory, converts it (if specified in the calling sequence), and writes it in the file workspace area. The workspace area for the file must have been previously defined by a call to XFILE. When enough data is collected in the workspace area to form a record, XWRITE sends the record to CYBER Record Manager, which outputs the record to a physical device.

```
        SA1  T1LIST
        RJ   XFILE
        .
        .
        .
T1LIST  VFD  42/0,18/FNAME1,42/0,18/WSA1,42/0,18/FSTR1,42/0,18/SIZE1,60/0
        .
        .
        .
FSTR1   DATA 51L(FT=T,USE=R,RECFM=VSB,BLKSIZE=100,LRECL=150,CODE=C)
        .
        .
        .
WSA1    BSS  40
        .
        .
        .
FNAME1  VFD  60/5LFILE1
SIZE    VFD  60/40
```

Figure 3-8. COMPASS XFILE Example

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

XWRITE HANDLING OF VARIABLE LENGTH RECORDS

When handling variable length records, XWRITE keeps track of the size of the output record (after conversion). If the IBM record type is V, VB, VS, or VSB, the source record length is used as the destination record length when the record is written.

For COBOL source items with a fixed length, conversion from internal format is terminated when the end of the item is encountered. No source field can extend beyond the end of item (destination record length), with the following exception:

> X, A, C, or B fields can extend beyond the item length if the fields are being converted to X or B items. The source item m value is reduced to the remaining record size. The destination m value is unchanged.

FORCING TERMINATION OF OUTPUT

You can use the XWRITE subroutines to force termination of output to a file, and to cause the writing of any partially filled workspace. When blocked tape formats (FB, VB, VSB, UB, or USB) are used, a final termination call to XWRITE should be issued after all other XWRITE calls to ensure that all data is output. This final call does not close the file.

The FORTRAN Extended 4, COBOL, and COMPASS calling sequences used to force termination of file output are shown in figure 3-9.

```
   CALL XWRITE(workspace)

COBOL:

   ENTER COMPASS XWRITE USING workspace

COMPASS:

           SA1     plist
           RJ      XWRITE
           .
           .
           .
  plist    VFD     42/0,18/workspace
           BSSZ    1
```

Figure 3-9. XWRITE Calling Sequence To Force Termination Of File Output

FORTRAN EXTENDED 4 XWRITE CALLING SEQUENCE

The FORTRAN Extended 4 XWRITE calling sequence is used to write data to the file workspace area defined by the XFILE calling sequence. The FORTRAN Extended 4 XWRITE calling sequence is shown in figure 3-10.

The FORTRAN XWRITE parameters are as follows:

- workspace

  Specifies the working storage area for a file, 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.

```
                                      ⎡⎧,conversion string       ⎫⎤
[CALL]XWRITE(workspace,source ⎢⎨                          ⎬⎥)
                                      ⎣⎩,conversion string,status⎭⎦
```

Figure 3-10. FORTRAN Extended 4 XWRITE Calling Sequence

- source

  Specifies the data to be written by the XWRITE operation. You must write the source parameter as the name of the array containing the source data.

- conversion-string

  Specifies the conversion string to be used. The conversion-string parameter must be enclosed in parentheses as described in section 2. For FORTRAN Extended 4, you can write the conversion-string parameter as a variable, as an array name, or as a left-justified Hollerith constant. If this parameter is omitted, the record size is not converted before the record is written to the workspace area. Record size default values are as follows:

card files	80 characters
print files	137 characters (136 characters if a format is specified)
tape files	LRECL

- status

  Specifies the result of the XWRITE operation. You must write the optional status parameter as a real variable name to which a status value is returned. The status values returned after the write operation are shown in table 3-1.

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

TABLE 3-1. STATUS VALUES RETURNED FOR XWRITE CALL

Status Value	Definition
0.0	No error, no abnormal condition
1.0	Error during conversion
2.0	Nonrecoverable error in the output file

If the XWRITE subroutine is called as a FORTRAN Extended 4 function, the status value is returned as the value of the function. For example, consider the following:

```
ISTAT=XWRITE(A,B,"(X10X10)")
```

In this case, the conversion is performed and the status parameter is returned as a real value for the value of the function. The real value is then converted to an integer to replace the value of ISTAT.

Figure 3-11 shows an example of a FORTRAN Extended 4 XWRITE calling sequence. Internal data is to be written to TT9, an EBCDIC-coded tape file. The record format of the output file is specified in the XFILE calling sequence. Before the first write can take place, the file must be opened and the XFILE subroutine called.

```
       .
       .
       .
       INTEGER FSTRING(5),IRAY(30),WSA(46)
       DATA FSTRING/"(FT=T,USE=W,RECFM=FB,
     +   BLKSIZE=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 30
  30   FORMAT("FAILED IN WRITE")
       .
       .
       .
```

Figure 3-11. FORTRAN Extended 4 XWRITE Example

COBOL XWRITE CALLING SEQUENCE

The COBOL XWRITE calling sequence is used to write data to the file workspace area defined by the XFILE calling sequence. The COBOL XWRITE calling sequence is shown in figure 3-12.

The COBOL XWRITE parameters are as follows:

- workspace

  Specifies the working storage area for a file, 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

  Specifies the data to be written by the XWRITE operation. You must write the source parameter as the data item containing the source data.

```
                                            ┌┌,conversion-string        ┐┐
ENTER COMPASS XWRITE USING workspace,source ││                          ││ .
                                            └└,conversion-string,status ┘┘
```

Figure 3-12. COBOL XWRITE Calling Sequence

- conversion-string

  Specifies the conversion string to be used. The conversion-string parameter must be enclosed in parentheses as described in section 2. You can write the conversion-string parameter either as a data name or as a COBOL alphanumeric literal string enclosed in quotation marks. If this parameter is omitted, the record is not converted before being written into the workspace area. Item size is passed in the parameter list. Record length is determined by either item size or record size, whichever is shorter.

- status

  Specifies the result of the XWRITE operation. You must write the optional status parameter as a COMPUTATIONAL-2 (real) item to which a status value is returned. The status values returned after the write operation are shown in table 3-1.

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

The Procedure Division of a COBOL XWRITE program is shown in figure 3-13. As in the preceding FORTRAN XWRITE example, internal data is to be written to TT9, an EBCDIC-coded tape file. The record format of the output file is specified in the XFILE calling sequence. Before the first write can take place, the file must be opened and the XFILE routine called.

```
PROCEDURE DIVISION.
  OPEN OUTPUT TT9.
  ENTER COMPASS XFILE USING TT9, WS1, FSTRING,
    SZ.
  ENTER COMPASS XWRITE USING WS1, CS1,
    "(C30X30)", STAT.
  IF STAT NOT EQUAL 0.0 DISPLAY "FAIL W1".
```

Figure 3-13. COBOL XWRITE Example

COMPASS XWRITE CALLING SEQUENCE

The COMPASS XWRITE calling sequence is used to write data to the file workspace area defined by the XFILE calling sequence. The COMPASS XWRITE calling sequence is shown in figure 3-14.

The COMPASS XWRITE parameters are as follows:

- plist

  Specifies the symbolic location of the parameter list; plist must be terminated by a word of binary zeros.

- workspace

  Specifies the working storage area for a file as defined in the XFILE calling sequence. You can write the workspace parameter as the symbolic location of the workspace area for a file. The XWRITE call is linked with a specific file only by the workspace name.

- source

  Specifies the data to be written by the XWRITE operation. You must write the source parameter as the location of the area containing the source data.

- conversion-string

  Specifies the conversion string to be used. You must write the conversion-string parameter as the symbolic location of a left-justified character data string with the form nLstring, nHstring, or nZstring. The conversion-string parameter must be enclosed in parentheses as described in section 2. If this parameter is omitted, the record is not converted before being written to the workspace area.

- status

  Specifies the result of the XWRITE operation. You must write the optional status parameter as the symbolic location of a word to which the real (floating-point) status value is returned. Status is an optional parameter. The status values returned after the write operation are shown in table 3-1.

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

Figure 3-15 shows an example of a COMPASS XWRITE calling sequence. Ten EBCDIC characters are written from the area IRAY to working storage area WS1; from WS1 the characters are written to tape. The XFILE subroutine must be called before the first write can take place.

XREAD/XREREAD SUBROUTINES

You can use the XREAD subroutine to read a next record from the file workspace area, to convert the record (if specified in the calling sequence), and to place the record in a destination area in memory. You can also use XREAD for skipping records. A call to XFILE must precede a call to XREAD/XREREAD for each file processed. The call to XFILE identifies the workspace area for a file.

You can use the XREREAD subroutine to reread the current record, with either the same or a different conversion specification.

```
          SA1   plist
          RJ    XWRITE
          .
          .
          .
    plist VFD   42/0,18/workspace
          VFD   42/0,18/source
              [{42/0,18/conversion-string                    }]
          VFD [{                                              }]
              [{42/0,18/conversion-string,42/0,18/status     }]
          VFD   60/0
```

Figure 3-14. COMPASS XWRITE Calling Sequence

```
        SA1  A1LIST
        RJ   XFILE
        SA1  WRLIST
        RJ   XWRITE
        .
        .
        .
A1LIST  VFD  42/0,18/FNAME1,42/0,18/WS1,42/0,18/FSTR1,42/0,18/SIZE1,60/0
WRLIST  VFD  42/0,18/WS1,42/0,18/IRAY1,42/0,18/CSTR1,42/0,18/STAT,60/0
FSTR1   DATA 51L(FT=T,USE=W,RECFM=VSB,BLKSIZE=100,LRECL=150,CODE=C)
CSTR1   DATA 8L(C10X10)
WS1     BSS  52
STAT    BSS  1
FNAME1  VFD  60/5LFILE1
SIZE1   VFD  60/52
IRAY1   VFD  60/4LIRAY
```

Figure 3-15. COMPASS XWRITE Example

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

XREAD/XREREAD HANDLING OF VARIABLE LENGTH RECORDS

If you use XREAD/XREREAD to handle variable length records with IBM record types V, VB, VS or VSB, the input record length can be any size up to the value specified for LRECL. Conversions are terminated when the end of the input record is reached. No source field can extend beyond the end-of-record, with the following exception:

> X or B fields can extend beyond the end-of-record if the source data is being converted to X, A, C, or B data types. The source data m value is reduced to the remaining record size. The destination m value is unchanged.

SKIPPING RECORDS

If you want to skip a record, the XREAD calling sequence can be used to read the next record from the workspace area without either converting the record or moving it to the destination area. The subsequent record is then available as the current record and can be processed by XREREAD. The XREAD calling sequences used by FORTRAN Extended 4, COBOL, and COMPASS for skipping records are shown in figure 3-16.

```
FORTRAN Extended 4:

   CALL XREAD(workspace)


COBOL:

   ENTER COMPASS XREAD USING workspace.


COMPASS:

            SA1     plist
            RJ      XREAD
            .
            .
            .
plist       VFD     42/0,18/workspace
            BSSZ    1

plist       Specifies the symbolic location
            of the parameter list.

workspace   Specifies the working storage
            area of the file containing the
            record to be skipped.
```

Figure 3-16. XREAD Calling Sequences for Skipping Records

ERROR HANDLING

When errors are detected in processing input or output calling sequences, an error message (appendix B) is written to the job's OUTPUT file and dayfile. Error status codes are returned in the status location given in the XREAD/XREREAD parameter list. For a FORTRAN Extended 4 or COBOL program, an attempt is made to provide traceback information.

No error is fatal to execution; however, all data conversion errors terminate translation at the point of detection. For example, when a conversion error is found by XREAD, the remainder of the record is not converted. XREREAD can then be used as an alternative procedure to reprocess the record with a different conversion string. Input/output errors cause unpredictable results.

FORTRAN EXTENDED 4 XREAD/XREREAD CALLING SEQUENCE

You can use the FORTRAN Extended 4 XREAD calling sequence to read the next record. You can use the FORTRAN Extended 4 XREREAD calling sequence to reread the current record. Both calling sequences are shown in figure 3-17.

The FORTRAN XREAD/XREREAD parameters are as follows:

- workspace

  Specifies the 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

  Specifies the area where the processed record is placed after the XREAD/XREREAD operation. You must write the destination parameter as an array name dimensioned to contain the number of words in the record.

- conversion-string

  Specifies the conversion string to be used. The conversion-string parameter must be enclosed in parentheses as described in section 2. For FORTRAN Extended 4, you can write the conversion-string parameter as a variable, an array name, or a left-justified Hollerith constant. If the conversion-string parameter is omitted, the record is not converted as it is transferred to the destination area.

- status

  Specifies the result of the XREAD/XREREAD operation. You must write the optional status parameter as a real variable to which a status value is returned. The status values returned after the read operation are shown in table 3-2.

TABLE 3-2. STATUS VALUES RETURNED FOR XREAD/XREREAD CALL

Status Value	Definition
-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 or abnormal condition
1.0	Error during conversion
2.0	Nonrecoverable error in input file

If either XREAD or XREREAD is used as a function, the status value is returned as the value of the function.

Figure 3-18 shows an example of a FORTRAN Extended 4 XREAD calling sequence. A tape containing only 30-character EBCDIC-coded records (no binary data) is to be read by a FORTRAN program. The characters are converted to CDC format. The conversion to be performed is specified by the conversion-string parameter in the XREAD calling sequence.

```
FORTRAN Extended 4 Example:

   INTEGER SZ,CSTRING,FSTRING(5),IRAY(30),WSA(46)
   DATA STAT/0.0/,CSTRING/"(X30C30)"/,SZ/46/
   DATA FSTRING/"(FT=T,USE=W,RECFM=FB,
 +   BLKSIZE=150,LRECL=30,CODE=C)"/
   .
   .
   .
   CALL XFILE(1,WSA,FSTRING,SZ)
   CALL XREAD(WSA,IRAY,CSTRING,STAT)
   .
   .
   .
```

Figure 3-18. FORTRAN Extended 4 XREAD Examples

COBOL XREAD/XREREAD CALLING SEQUENCE

You can use the COBOL XREAD calling sequence to read the current record. You can use the COBOL XREREAD calling sequence to read the current record. Both calling sequences are shown in figure 3-19.

```
[CALL]XREAD(workspace,destination   [{,conversion-string        }])
                                     {,conversion-string,status }

[CALL]XREREAD(workspace,destination [{,conversion-string        }])
                                     {,conversion-string,status }
```

Figure 3-17. FORTRAN Extended 4 XREAD/XREREAD Calling Sequences

```
ENTER COMPASS XREAD USING workspace, destination  [{,conversion-string        }]
                                                  [{,conversion-string,status }]

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

Figure 3-19. COBOL XREAD/XREREAD Calling Sequences

The COBOL XREAD/XREREAD parameters are as follows:

- workspace

  Specifies the 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

  Specifies the area where the processed record is placed after the XREAD/XREREAD operation. You must write the destination parameter as the data name of the destination area which is to contain a processed record.

- conversion-string

  Specifies the conversion string to be used. This parameter must be enclosed in parentheses as described in section 2. You can write the conversion-string parameter as a data name or as a COBOL alphanumeric literal string enclosed in quotation marks. If the conversion-string parameter is omitted, the record is transferred to the destination area without conversion.

- status

  Specifies the result of the XREAD/XREREAD operation. You must write the optional parameter status as a COMPUTATIONAL-2 data name to which a status value is returned. The status values returned after the read operation are shown in table 3-2.

A COBOL XREAD example is shown in figure 3-20. As in the FORTRAN XREAD example, TT9, a tape containing 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. The conversion to be performed is specified by the conversion-string parameter in the XREAD calling sequence.

```
DATA DIVISION.
 .
 .
 .
WORKING-STORAGE SECTION.
77  SZ           PIC 999      USAGE COMP-1  VALUE 46.
77  STAT                      USAGE COMP-1  VALUE  0.
01  CSTRING      PIC X(60)          VALUE "(X90C30)".
01  FSTRING      PIC X(60)          VALUE "(FT=T,
-      "USE=R, RECFM=FB, BLKSIZE=150, LRECL=30,
-         "CODE=C)".
01  TMP1         PIC X(30).
01  WS1          PIC X(460)         VALUE SPACES.
 .
 .
 .
PROCEDURE DIVISION.
 .
 .
 .
 OPEN INPUT TT9.
 ENTER COMPASS XFILE USING TT9, WS1, FSTRING, SZ.
 ENTER COMPASS XREAD USING WS1, TMP1, CSTRING,
   STAT.
 IF STAT NOT EQUAL 0 DISPLAY "FAIL IN READ".
```

Figure 3-20. COBOL XREAD Example

H2 COMPASS XREAD CALLING SEQUENCE

You can use the COMPASS XREAD calling sequence to read the next record. You can use the COMPASS XREREAD calling sequence to read the current record. Both calling sequences are shown in figure 3-21.

The COMPASS XREAD/XREREAD parameters are as follows:

- plist

  Specifies the symbolic location of the parameter list; plist must be terminated with a word of binary zeros.

- workspace

  Specifies the working storage area of a file, as defined in the XFILE calling sequence. The read calls are linked to a specific file only by the workspace name.

- destination

  Specifies the area which contains a record after the XREAD/XREREAD operation. You must write the destination parameter as the symbolic location of the area where the processed record is to be placed.

- conversion-string

  Specifies the conversion string to be used. The conversion-string parameter must be enclosed in parentheses as described in section 2. You must write the conversion-string parameter as the symbolic location of a left-justified character data string in the format nLstring, nHstring, or nZstring. If the conversion-string parameter is omitted, the record is not converted as it is transferred to the destination area.

- status

  Specifies the result of the XREAD/XREREAD operation. You must write the optional status parameter as the symbolic location of a word to which a real (floating-point) value indicating the status of the read is returned. The status values returned after the read operation are shown in table 3-2.

Figure 3-22 shows an example of a COMPASS XREAD calling sequence. The program reads ten 8-bit EBCDIC characters from tape file working storage area WS1, converts the characters to internal EBCDIC, and places the characters in area IRAY.

```
         SA1   plist
         RJ    XREAD
          .
          .
          .
   plist VFD   42/0,18/workspace,42/0,18/destination
             ┌ ┌ 42/0,18/conversion-string                    ┐ ┐
         VFD │ │                                              │ │
             └ └ 42/0,18/conversion-string,42/0,18/status     ┘ ┘
         VFD   60/0


         SA1   plist
         RJ    XREREAD
          .
          .
          .
   plist VFD   42/0,18/workspace,42/0,18/destination
             ┌ ┌ 42/0,18/conversion-string                    ┐ ┐
         VFD │ │                                              │ │
             └ └ 42/0,18/conversion-string,42/0,18/status     ┘ ┘
         VFD   60/0
```

Figure 3-21. COMPASS XREAD/XREREAD Calling Sequences

```
        SA1   T1LIST
        RJ    XFILE
         .
         .
         .
        SA1   RDLST
        RJ    XREAD
         .
         .
         .
T1LIST  VFD   42/0,18/FNAME2,42/0,18/WS1,42/0,18/FSTR1
        VFD   42/0,18/SIZE1,60/0
RDLST   VFD   42/0,18/WS1,42/0,18/IRAY,42/0,18/CSTR1
        VFD   42/0,18/STAT,60/0
FSTR1   DATA  51L(FT=T,USE=R,RECFM=VSB,BLKSIZE=100,LRECL=150,CODE=C)
         .
         .
         .
CSTR1   DATA  8L(X10C10)
         .
         .
         .
WS1     BSS   40
         .
         .
         .
IRAY    BSS   60
STAT    BSS   1
FNAME1  VFD   60/5LFILE1
         .
         .
         .
SIZE1   VFD   60/40
         .
         .
         .
```

Figure 3-22. COMPASS XREAD Example

UTILITY SUBROUTINES 4

You can use the utility subroutines XCOMP, XMOVE, XPACK, and XPAND to:

- Compare data strings
- Move data strings
- Compress data strings
- Expand data strings

The subroutines operate on data in any of the following internal formats:

- 12-bit bytes containing 7-bit ASCII codes
- 12-bit bytes containing 8-bit EBCDIC codes
- 6-bit bytes containing 6-bit display codes

The examples for each utility routine in this section use strings from the character groups S1DISPC and S2ASCII, shown in figure 4-1. Each string is stored in central memory. S1DISPC is stored in display code in 6-bit bytes; S2ASCII is stored in ASCII code in 12-bit bytes.

XCOMP SUBROUTINE

You can use the XCOMP subroutine to compare two character strings. The character strings need not consist of the same character codes. The character sets of each string are determined by the xy parameter of the XCOMP calling sequence. The characters that make up the xy parameter and their associated character sets are shown in table 4-1. The collating sequence used in the XCOMP subroutine is also determined by the xy parameter as shown in table 4-2.

FORTRAN EXTENDED 4 XCOMP CALLING SEQUENCE

The FORTRAN XCOMP calling sequence is used to compare two character strings. The FORTRAN XCOMP calling sequence is shown in figure 4-2. XCOMP can be called as a subroutine or as a function.

TABLE 4-1. CHARACTERS USED IN XY PARAMETER FOR XCOMP AND XMOVE

Character	Definition
A	Indicates a character string made up of 12-bit bytes containing 7-bit ASCII codes
C	Indicates a character string made up of 12-bit bytes containing 8-bit EBCDIC codes
X	Indicates a character string made up of 6-bit bytes containing 6-bit display codes

TABLE 4-2. COLLATING SEQUENCES USED BY XCOMP

XY Parameter	Collating Sequence
AC	Specifies the ASCII collating sequence (numbers collate low)
CA	Specifies the EBCDIC collating sequence (numbers collate high)
XA or XC	Specifies that the string is to be case-folded logically (see appendix A) so that uppercase and lowercase letters are equal
AX or CX	Specifies that uppercase and lowercase letters are to retain their identity and collate separately

S1DISPC:

Word 1	Word 2
A B 1 2 4 6 7 C D E	F G H 2 3 . X Y Z 2

S2ASCII:

Word 1	Word 2	Word 3	Word 4	Word 5
Z Y X A B	1 2 4 6 7	F G H 2 3	. X Y Z Z	3 4 5 6 7

Figure 4-1. S1DISPC and S2ASCII

```
                                                       ⎡⎧,position-1           ⎫⎤
[CALL]XCOMP(xy,source-1,source-2,status,length⎢⎨                        ⎬⎥)
                                                       ⎣⎩,position-1,position-2⎭⎦
```

Figure 4-2. FORTRAN Extended 4 XCOMP Calling Sequence

The FORTRAN XCOMP parameters are as follows:

- xy

  Specifies the character sets of each string to be compared, where:

  x Describes the source-1 character string code

  y Describes the source-2 character string code

  The xy parameter can consist of any combination of the characters shown in table 4-1. When xy is present in the call as a literal, you can write xy as a left-justified Hollerith constant. When a variable name is used in the call, the variable must have been defined previously as containing the characters for xy left-justified.

  The type of source-1 string in the xy parameter determines the collating sequence used, as shown in table 4-2. In the 8-bit ASCII and EBCDIC collating sequences, ascending binary code values correspond to ascending collating values. A variation can be provided to 6-bit display codes by installation option. The collating sequences are shown in appendix A.

- source

  Specifies the character strings to be compared. You can write one source parameter as an actual Hollerith string in the calling sequence.

- status

  Specifies the result of the XCOMP operation. You must write the status parameter as a real variable to which the result of the comparison is returned. The meaning of the status value returned is shown in figure 4-3.

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

Status value:	Meaning:
-1.0	Source-1 < Source-2
0.0	Source-1 = Source-2
+1.0	Source-1 > Source-2

Figure 4-3. Meaning of Status Values Returned From XCOMP

- length

  Specifies the number of characters to be compared. You can write the required length parameter as an integer constant or as a previously defined integer variable or expression.

- position-1/position-2

  Specifies the position, within the field, of the first character of the string to be compared. You can write the optional parameter position-i as an integer constant or as a previously defined integer variable or expression. Position-1 indicates the location of the first character to be compared in string-1; position-2 indicates the location of the first character to be compared in string-2. If position-i is omitted, the default is 1, denoting the leftmost (first) character in the string.

In figure 4-4 XCOMP is used as a function. The IF statement is used to direct the execution to appropriate parts of the program depending on the results of the comparison.

```
INTEGER S2ASCII(5),S1DISPC(2)
.
.
.
CALL XCOMP("XA",S1DISPC,S2ASCII,XSTAT,20,1,4)
.
.
.
```

Figure 4-4. FORTRAN Extended 4 XCOMP Example

Figure 4-5 shows an example of a FORTRAN comparison. Twenty characters, starting from character position 1 of S1DISPC, are compared with twenty characters, starting from position 4 of S2ASCII. The status information is returned in the variable XSTAT.

```
XSTAT=XCOMP("XA",S1DISPC,S2ASCII,20,1,4)
IF(XSTAT)10,20,30
```

Figure 4-5. Example of XCOMP Used as a FORTRAN Extended 4 Function

COBOL XCOMP CALLING SEQUENCE

The COBOL XCOMP calling sequence is used to compare two character strings. The COBOL XCOMP calling sequence is shown in figure 4-6.

The COBOL XCOMP parameters are as follows:

- xy

  Specifies the character set of each string to be compared. You can write the xy parameter as a COBOL alphanumeric literal or as a DISPLAY data item which you describe in the Data Division as alphabetic or alphanumeric, size 2. VALUE must be set to the desired two-character string, where:

  x Describes the source-1 character string code

  y Describes the source-2 character string code

  The xy parameter can consist of any combination of the characters shown in table 4-1.

  The type of source-1 string in the xy parameter determines the collating sequence used, as shown in table 4-2. In the 8-bit ASCII and EBCDIC collating sequences, ascending binary code values correspond to ascending collating values. A variation can be provided to 6-bit display codes by installation option. Appendix A gives the 6-bit display code collating sequences.

- source

  Specifies the character string to be compared. You must write the source parameter as the data item containing the first character of each string.

- status

  Specifies the result of the XCOMP operation. You must write the status parameter as a COMPUTATIONAL-2 item to which a value is returned as the result of the comparison. The meaning of the status value returned is shown in figure 4-3.

- length

  Specifies the number of characters to be compared. You must write the optional length parameter as a COMPUTATIONAL-1 item. If the length parameter is 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 the length parameter is given, the length parameter overrides the actual string length.

- position-1/position-2

  Specifies the initial character position to be used for comparison in the string. You must write the optional position-i parameter as a COMPUTATIONAL-1 item of no more than 14 digits. Position-1 indicates the location of the first character to be compared in string-1; position-2 indicates the location of the first character to be compared in string-2. When the position-i parameter is omitted, the default value is 1, denoting the first character in the string. If position-i is designated, length must also be specified.

Figure 4-7 shows an example of a COBOL comparison. Twenty characters from S1DISPC, starting at character position 1, are compared to twenty characters from S2ASCII, starting at character position 4. The status information is returned in item XSTAT.

```
                                                      ⎡⎧,length                      ⎫⎤
ENTER COMPASS XCOMP USING xy,source-1,source-2,status ⎢⎨,length,position-1           ⎬⎥.
                                                      ⎣⎩,length,position-1,position-2⎭⎦
```

Figure 4-6. COBOL XCOMP Calling Sequence

```
DATA DIVISION.
01  EXWHY              PICTURE(XXX)          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 COMPASS XCOMP USING EXWHY, S1DISPC, S2ASCII, XSTAT,
    LENGTH, POSONE, POSTWO.
```

Figure 4-7. COBOL XCOMP Example

COMPASS XCOMP CALLING SEQUENCE

The COMPASS XCOMP calling sequence is used to compare two character strings. The COMPASS XCOMP calling sequence is shown in figure 4-8.

```
      SA1   plist
      RJ    XCOMP
      .
      .
      .
plist VFD   42/0,18/xy,42/0,18/source-1
      VFD   42/0,18/source-2,42/0,18/status
           [{42/0,18/position-1                         }]
      VFD  [{42/0,18/position-1,42/0,18/position-2 }]
      VFD   60/0
```

Figure 4-8. COMPASS XCOMP Calling Sequence

The COMPASS parameters are as follows:

- plist

  Specifies the symbolic location of the parameter list; plist must end with a word of binary zeros.

- xy

  Specifies the character set of each string to be compared. The xy parameter must be the symbolic location of a word that contains the code set of each character string. You can write the xy parameter either as a constant or as a literal written in the format nLstring, nHstring or nZstring, where:

  x Describes the source-1 character string code

  y Describes the source-2 character string code

  The xy parameter can consist of any combination of the characters shown in table 4-1.

  The type of source-1 string in the xy parameter determines the collating sequence used, as shown in table 4-2. In the 8-bit ASCII and EBCDIC collating sequences, ascending binary code values correspond to ascending collating values. A variation can be provided to 6-bit display codes by installation option. The collating sequences are shown in appendix A.

- source

  Specifies the character strings to be compared. You must write the source parameter as the symbolic location of a word containing the first character in a string.

- status

  Specifies the result of the XCOMP operation. You must write the status parameter as the symbolic location of a word to which a floating-point value is returned. The meaning of the status value returned is shown in figure 4-3.

- length

  Specifies the number of characters to be compared. You must write the length parameter as the symbolic location of a word containing an integer value representing the number of sequential characters to be compared.

- position-1/position-2

  Specifies the position, within the field, of the first character to be compared in the string. You must write the optional position parameter as the symbolic location of an integer value. Position-1 indicates the location of the first character to be compared in string-1; position-2 indicates the location of the first character to be compared in string-2. When comparison begins with the first character in the source string, the position is 1. If the parameter is omitted, the default is 1, denoting the leftmost (first) character in the string.

Figure 4-9 shows an example of a COMPASS comparison. Twenty characters starting at position 1 of S1DISPC are compared to twenty characters starting at position 4 of S2ASCII. The status information is returned in item XSTAT.

```
        SA1  PLIST
        RJ   XCOMP
        .
        .
        .
PLIST   VFD  42/0,18/XY,42/0,18/S1DISPC,42/0,18/S2ASCII,42/0,18/XSTAT
        VFD  42/0,18/LGTPOS,42/0,18/P1,42/0,18/P2,60/0
        .
        .
        .
XY      VFD  18/2LXA,42/0
S1DISPC BSS  2
S2ASCII BSS  5
XSTAT   BSSZ 1
LGTPOS  DATA 20
P1      DATA 1
P2      DATA 4
```

Figure 4-9. COMPASS XCOMP Example

XMOVE SUBROUTINE

You can use the XMOVE subroutine to move a designated character string from a source location to a destination location. The character string can be translated from one character code to another during the move. The xy parameter of the XMOVE subroutine determines character conversion. Uppercase characters can be converted to lowercase characters when the three-character xy parameter shown in table 4-3 is used.

TABLE 4-3. THREE-CHARACTER XY PARAMETER USED IN XMOVE

Characters	Definition
XAL	Converts 6-bit display codes to 7-bit ASCII codes stored in 12-bit bytes, with case reversed.
XCL	Converts 6-bit display codes to 8-bit EBCDIC codes stored in 12-bit bytes, with case reversed.

When no character conversion is involved, XMOVE transfers character strings in 60-bit 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. Characters might not move as expected when the source and destination fields overlap.

FORTRAN EXTENDED 4 XMOVE CALLING SEQUENCE

The FORTRAN XMOVE calling sequence is used to move character strings. The FORTRAN XMOVE calling sequence is shown in figure 4-10.

The FORTRAN parameters are as follows:

- xy

  Specifies the source and destination string character sets. You can write the source parameter either as a left-justified Hollerith constant or as a variable that you have defined to contain the appropriate characters.

  x Describes the source character set

  y Describes the destination character set

  The xy parameter can consist of two or three characters. The two-character parameter can be any combination of the characters shown in table 4-1. When the three-character xy parameter is used, characters appear in the destination string as lowercase characters rather than as uppercase characters. Special symbolic characters are also case-reversed; as a result, printable characters might become unprintable. The three-character xy parameter begins with X as shown in table 4-3.

- source

  Specifies the character string to be moved. You can write the source parameter as a variable name, an array name, a subscripted array name, or optionally, a Hollerith constant when the source string code is X.

- destination

  Specifies the location in the destination field to which the source character string is to be moved. You can write the destination parameter as a variable name, an array name, or a subscripted array name.

- length

  Specifies the number of characters to be moved. You can write the length parameter as an integer or as an integer variable.

- position-s

  Specifies the first character in the source string that is to be moved. You can write the optional position-s parameter either as an integer or as an integer variable. If the position-s parameter is omitted, the default value is 1, denoting that the leftmost character is the first character to be moved in the string.

- position-d

  Specifies the first receiving character in the destination field. You can write the optional postion-d parameter either as an integer or as an integer variable. If the position-d parameter is omitted, the default value is 1, denoting that the leftmost character in the destination field is the first character to receive data.

Figure 4-11 shows an example of a FORTRAN move. Using the S2ASCII character string, a 20-character ASCII code string starting at character position 4 is converted to EBCDIC and placed in array WS1 starting at character position 1. Omission of a value for the parameter position-d implies character position 1 by default.

COBOL XMOVE CALLING SEQUENCE

The COBOL XMOVE calling sequence is used to move character strings. The COBOL XMOVE calling sequence is shown in figure 4-12.

```
                                           ⎡⎧,position-s            ⎫⎤
CALL XMOVE(xy,source,destination,length    ⎢⎨                       ⎬⎥)
                                           ⎣⎩,position-s,position-d ⎭⎦
```

Figure 4-10. FORTRAN Extended 4 XMOVE Calling Sequence

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

Figure 4-11. FORTRAN Extended 4 XMOVE Example

The COBOL parameters are as follows:

- xy

  Specifies the source string and destination string character sets. You can write the parameter xy either as a data item or as an alphabetic literal descriptor which you describe in the Data Division as alphabetic or (COBOL) alphanumeric, size 2 or size 3. VALUE must be set to the desired two- or three-character string.

  x Describes the source character set

  y Describes the destination character set

  The xy parameter can consist of two or three characters. The two-character parameter can be any combination of the characters shown in table 4-1.

  When the three-character xy parameter is used, characters appear in the destination string as lowercase characters rather than as uppercase characters. Special symbolic characters are also case-reversed; as a result, printable characters might become unprintable. The three-character xy parameter begins with X as shown in table 4-3.

- source

  Specifies the starting location of the character string to be moved. You can write the source parameter as a data name, a literal subscripted data name, or a COBOL alphanumeric literal.

- destination

  Specifies the area to which the source character string is moved. You can write the destination parameter as a data-name, a literal subscripted data-name, or a COBOL alphanumeric literal.

- length

  Specifies the number of characters to be moved. You must write the optional length parameter as a COMPUTATIONAL-1 item of no more than 14 digits. If omitted, the default value is the length of the destination field in number of characters. If the default value is used and the source string and destination field are of unequal lengths, the source string is either truncated or blank-filled. When the length specification is given, the actual character length of the destination field is overridden.

- position-s

  Specifies the first character in the source string to be moved. You must write the optional position-s parameter as a COMPUTATIONAL-1 data name. If the position-s parameter is omitted, the default is 1, indicating that the leftmost character in the source string is the first character to be moved.

- position-d

  Specifies the position of the first receiving character in the destination field. You must write the optional position-d parameter as a COMPUTATIONAL-1 data name. If the position-d parameter is omitted, the default is 1, indicating that the leftmost character in the destination field is the first character to receive data.

Figure 4-13 shows an example of a COBOL move. This example moves a 20-character 12-bit byte ASCII code string from ASCII to WS1. The code string starting with character 4 of the group stored in S2ASCII is converted to 12-bit EBCDIC and placed in the first character position of WS1.

```
                                                 [{ ,length                      }]
   ENTER COMPASS XMOVE USING xy,source,destination [{ ,length,position-s           }] .
                                                 [{ ,length,position-s,position-d }]
```

Figure 4-12. COBOL XMOVE Calling Sequence

```
01  XY                   PIC XXX           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 COMPASS XMOVE USING XY, S2ASCII, WS1, LENGTH, POSIT.
```

Figure 4-13. COBOL XMOVE Example

Assume the COBOL XMOVE calling sequence

```
ENTER COMPASS XMOVE USING "AC",S2ASCII,WS2
```

In this sequence, a 25-character, 12-bit byte ASCII code string from S2ASCII is converted to 12-bit EBCDIC.

COMPASS XMOVE CALLING SEQUENCE

The COMPASS XMOVE calling sequence is used to move character strings. The COMPASS XMOVE calling sequence is shown in figure 4-14.

```
      SA1   plist
      RJ    XMOVE
      .
      .
      .
plist VFD   42/0,18/source
      VFD   42/0,18/destination
      VFD   42/0,18/length
            [{42/0,18/position-s                          }]
      VFD   [{42/0,18,position-s,42/0,18/position-d}]
      VFD   60/0
```

Figure 4-14. COMPASS XMOVE Calling Sequence

The COMPASS parameters are as follows:

- plist

  Specifies the symbolic location of a parameter list; plist must end with a word of binary zeros.

- xy

  Specifies the source and destination string character codes. The xy parameter must be the symbolic location of a word containing the source and destination string character codes. You can write the xy parameter either as a constant or as a literal in the form nLstring or nHstring, where:

  x Describes the source character set

  y Describes the destination character set

  The xy parameter can consist of two or three characters. The two-character parameter can be any combination of the characters shown in table 4-1.

  When the three-character xy parameter is used, characters appear in the destination string as lowercase characters rather than as uppercase characters. Special symbolic characters are also case-reversed; as a result printable characters might become unprintable. The three character xy parameter is preceded by an X as shown in table 4-2. When case reversal is used, the parameter xy should be defined as either 3Hxy or 3Lxy.

- source

  Specifies the string to be moved. You must write the source parameter as the symbolic location of the area containing the character string to be moved.

- destination

  Specifies the area to which the source character string is moved. You must write the destination parameter as the symbolic location of the destination area that is to receive the character string.

- length

  Specifies the number of characters to be moved. You must write the length parameter as the symbolic location of a word containing an integer value, denoting the length of the string.

- position-s

  Specifies the first character that is to be moved in the source string. You must write the optional position-s parameter as the symbolic location of an integer value. If position-s is omitted, the default value is 1, indicating the leftmost character in the string is the first character to be moved.

- position-d

  Specifies the first receiving character in the destination field. You must write the optional parameter position-d as the symbolic location of an integer value. If position-d is omitted, the default value is 1, indicating the first character moved is placed in the leftmost (first) character position of the destination field.

Figure 4-15 shows a COMPASS XMOVE example. A 20-character ASCII code string from S2ASCII is to be moved to WS1. Starting from character position 4 of S2ASCII, the 20-code ASCII string is converted to EBCDIC and placed in WS1, starting in character position 1.

```
        SA1  PLIST
        RJ   XMOVE
        .
        .
        .
PLIST   VFD  42/0,18/XY
        VFD  42/0,18/S2ASCII
        VFD  42/0,18/WS1
        VFD  42/0,18/LEN
        VFD  42/0,18/LEN+1
        BSSZ 1
        .
        .
        .
XY      VFD  18/2LAC,42/0
S2ASCII BSS  5
WS1     BSS  5
LEN     DATA 20,4
```

Figure 4-15. COMPASS XMOVE Example

XPACK SUBROUTINE

You can use the XPACK subroutine to compress 8-bit character data from a 5-character per word (12-bit byte) internal format containing ASCII or EBCDIC characters. When packing is performed, seven 8-bit character codes are right-justified in a word; the leftmost 4 bits, as well as any unused character positions, are zero-filled. The packed word format is shown in figure 4-16.

No conversion of character data takes place during string compression.

During string compression, several characters are moved at once. If the source and destination fields overlap, the characters might not be moved as expected.

FORTRAN EXTENDED 4 XPACK CALLING SEQUENCE

The FORTRAN XPACK calling sequence is used to unpack character data. The FORTRAN XPACK calling sequence is shown in figure 4-17.

The FORTRAN parameters are as follows:

- string-u

  Specifies the 12-bit byte (unpacked) character source string. You must write the parameter string-u as the name of the array containing the unpacked string.

- string-p

  Specifies the 8-bit byte (packed) character destination string. You must write the parameter string-p as the name of the array which receives the packed string.

- length

  Specifies the number of characters to be moved and packed. You can write the length parameter as either an integer constant or an integer variable.

- position

  Specifies the position of the first character in the string to be packed. You can write the optional position parameter as an integer constant or as an integer variable.

A FORTRAN XPACK example is shown in figure 4-18. A 20-character string starting from character position 4 of S2ASCII is packed into WS1.

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

Figure 4-17. FORTRAN Extended 4 XPACK Calling Sequence

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

Figure 4-18. FORTRAN Extended 4 XPACK Example

The first two words of the destination array WS1 are filled with seven characters each. The third word contains six characters and an 8-bit byte that is binary zero-filled.

COBOL XPACK CALLING SEQUENCE

The COBOL XPACK calling sequence is used to unpack character data. The COBOL XPACK calling sequence is shown in figure 4-19.

The COBOL parameters are as follows:

- string-u

  Specifies the 12-bit byte (unpacked) character source string.

- string-p

  Specifies the 8-bit byte (packed) character destination string. You must write the parameter string-p as a data item that begins on a word boundary. Every 01 level or synchronized left item in COBOL begins on a word boundary.

- length

  Specifies the number of characters to be moved and packed. You must write the optional length parameter as a COMPUTATIONAL-1 item of no more than 14 digits. If the length parameter is omitted, the default is the length in number of characters in the shorter string. The receiving field is then binary zero-filled as needed.

	59 57	56 49	48 41	40 33	32 25	24 17	16 8	7 0
CM Word	0	Character 1	Character 2	Character 3	Character 4	Character 5	Character 6	Character 7

Figure 4-16. Packed Word Format

- position

  Specifies the first character in string-u to be packed in string-p. You must write the optional position parameter as a COMPUTATIONAL-1 item. If the position parameter is omitted, the default is 1.

A COBOL XPACK example is shown in figure 4-20. A 20-character string starting with character 4 of S2ASCII is packed into WS1.

COMPASS XPACK CALLING SEQUENCE

The COMPASS XPACK calling sequence is used to unpack character data. The COMPASS XPACK calling sequence is shown in figure 4-21.

The COMPASS parameters are as follows:

- plist

  Specifies the symbolic location of the parameter list; plist must end with a word of binary zeros.

- string-u

  Specifies the 12-bit byte (unpacked) character source string. You must write the parameter string-u as the symbolic location of a word containing the unpacked string.

- string-p

  Specifies the 8-bit byte (packed) character destination string. You must write the parameter string-u as the symbolic location of the area that receives the packed string.

- length

  Specifies the number of characters to be packed. You must write the length parameter as the symbolic location of a word containing an integer value.

```
            SA1 plist
            RJ  XPACK
            .
            .
            .
      plist VFD 42/0,18/string-u
            VFD 42/0,18/string-p
            VFD 42/0,18/length
            VFD 42/0,18/position
            VFD 60/0
```

Figure 4-21. COMPASS XPACK Calling Sequence

- position

  Specifies the position of the first character to be packed in the source string. You must write the optional position parameter as the symbolic location of an integer value indicating character position within the field of the first character of the string to be packed. If the position parameter is omitted, the default is 1.

A COMPASS XPACK example is shown in figure 4-22. A 20-character string starting at character 4 of S2ASCII is packed into storage area WS1.

XPAND SUBROUTINE

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, the strings must be unpacked into 12-bit bytes for proper handling. You can use the XPAND subroutine to reverse the process performed by the XPACK subroutine and to unpack the 8-bit compressed string into words containing five 12-bit character bytes.

FORTRAN EXTENDED 4 XPAND CALLING SEQUENCE

The FORTRAN XPAND calling sequence is used to unpack character data. The FORTRAN XPAND calling sequence is shown in figure 4-23.

```
                                                  ┌┌,length          ┐┐
ENTER COMPASS XPACK USING string-u,string-p       │┤                  ├│ .
                                                  └└,length,position  ┘┘
```

Figure 4-19. COBOL XPACK Calling Sequence

```
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 COMPASS XPACK USING S2ASCII, WS1, LENGTH, POSIT.
```

Figure 4-20. COBOL XPACK Example

```
         SA1  PLIST
         RJ   XPACK
         .
         .
         .
PLIST    VFD  42/0,18/S2ASCII,42/0,18/WS1,42/0,18/LENGTH
         VFD  42/0,18/LENGTH1,60/0
S2ASCII  BSS  5
WS1      BSS  3
LENGTH   DATA 20,4
```

Figure 4-22. COMPASS XPACK Example

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

Figure 4-23. FORTRAN Extended 4 XPAND Calling Sequence

The FORTRAN parameters are as follows:

- string-u

  Specifies the 12-bit byte (unpacked) destination area. You must write the parameter string-u as the name of the array which receives the unpacked data.

- string-p

  Specifies the 8-bit byte character source string. You must write the parameter string-p as the name of the array containing the packed string.

- length

  Specifies the number of characters to be unpacked. You can write the length parameter as either an integer constant or as an integer variable.

- position

  Specifies the first character of the string to be unpacked. You can write the optional position parameter as either an integer constant or as an integer variable.

Figure 4-24 shows a FORTRAN example of the XPAND subroutine. A 20-character string that has been read from a file into central memory storage area WS1 is expanded. The expanded string is restored to S2ASCII, starting at character position 4.

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

Figure 4-24. FORTRAN Extended 4 XPAND Example

COBOL XPAND CALLING SEQUENCE

The COBOL XPAND calling sequence is used to unpack character data. The COBOL XPAND calling sequence is shown in figure 25.

The COBOL parameters are as follows:

- string-u

  Specifies the destination area of the unpacked 12-bit byte string. You must write the parameter string-u as a data item which receives the unpacked data.

- string-p

  Specifies the 8-bit byte (packed) source string. The packed character string must begin on a word boundary. You must write the string-p parameter as an 01 or synchronized left item.

- length

  Specifies the number of characters to be unpacked. You must write the optional parameter length as a COMPUTATIONAL-1 item of not more than 14 digits. If the length parameter is omitted, the default is the character length of the shorter string; binary zero fill completes the destination area.

- position

  Specifies the position of the first character in the string-u (destination) area to receive the unpacked data. You must write the optional position parameter as a COMPUTATIONAL-1 item. If the position parameter is omitted, the default value is 1, designating the leftmost character in the destination area.

The calling sequence shown in figure 4-26 expands a 20-character string that has been read from a file into WS1, a memory storage area. The expanded string is then restored to S2ASCII starting at character position 4.

COMPASS XPAND CALLING SEQUENCE

The COMPASS XPAND calling sequence is used to unpack character data. The COMPASS XPAND calling sequence is shown in figure 4-27.

```
                                               ┌ ┌,length          ┐ ┐
ENTER COMPASS XPAND USING string-u,string-p    │ {                  } │ .
                                               └ └,length,position ┘ ┘
```

Figure 4-25. COBOL XPAND Calling Sequence

```
    ENTER COMPASS XPAND USING S2ASCII, WS1,
      LENGTH, POS.
```

Figure 4-26. COBOL XPAND Example

```
          SA1     plist
          RJ      XPAND
          .
          .
          .
 plist    VFD     42/0,18/string-u,42/0,18/string-p
          VFD    [{42/0,18/length                      }]
                 [{42/0,18/length,42/0,18/position     }]
          VFD     60/0
```

Figure 4-27. COMPASS XPAND Calling Sequence

The COMPASS parameters are as follows:

- plist

  Specifies the symbolic location of the parameter list; plist must be terminated by a word of binary zero.

- string-u

  Specifies the destination area of the 12-bit byte (unpacked) character string. You must write the string-u parameter as the symbolic location of the destination area for the unpacked character string.

- string-p

  Specifies the 8-bit byte (packed) source string. The parameter string-p must be the storage location of the packed character string.

- length

  Specifies the number of characters to be moved and unpacked. You must write the length parameter as the symbolic location of an integer value denoting the number of characters to be unpacked in the string.

- position

  Specifies the position of the first character in the string-u (destination) area to receive unpacked data. You must write the optional position parameter as the symbolic location of an integer value indicating the position of the first character in the string-u (destination) area to receive data. If the position parameter is omitted, the default value is 1, designating the leftmost character in the destination area.

Figure 4-28 shows a COMPASS XPAND example. A 20-character string that has been read from a file into memory storage area WS1 is expanded. The expanded string is stored into area S2ASCII, starting in character position 4.

```
                SA1   PLIST
                RJ    XPAND
                .
                .
                .
        PLIST   VFD   42/0,18/S2ASCII
                VFD   42/0,18/WS1
                VFD   42/0,18/LENPOS
                VFD   42/0,18/LENPOS+1
                BSSZ  1
        WS1     BSS   3
        S2ASCII BSS   5
        LENPOS  DATA  20,4
```

Figure 4-28. COMPASS XPAND Example

COPY8P UTILITY 5

You can use COPY8P as a separate utility to copy IBM print files to CDC compatible print files. Since loss of 8-bit significance is avoided, uppercase and lowercase character capability is maintained. IBM options with respect to record type, block type, and print format selection are available. COPY8P does not use CYBER Record Manager when copying print files; therefore, you do not need to supply any FILE control statements in conjunction with this utility.

COPY8P CONTROL STATEMENT

You can call COPY8P by a control statement in the job stream. The COPY8P control statement is as follows:

COPY8P,$\mathrm{lfn_{in}}$,$\mathrm{lfn_{out}}$,$\mathrm{opt_1}$,$\mathrm{opt_2}$,...,$\mathrm{opt_n}$.

The following parameters are required with the COPY8P control statement.

- $\mathrm{lfn_{in}}$

  Specifies the logical file name of the input file containing 8-bit ASCII or EBCDIC character data in IBM format.

  If the input file is on magnetic tape, use a LABEL control statement to make the file available to the job.

  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 available. A copy of a file from tape to disk should provide one system-logical-record per block.

- $\mathrm{lfn_{out}}$

  Specifies the logical file name of the output tape or disk file chosen to contain the data in a format suitable for the printer.

  If full uppercase and lowercase information is to be printed, the output file must be directed to a printer with an extended character set print train by use of the ROUTE control statement discussed later in this section.

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

- ,RECFM=rf

  Describes the record format of the input file. Values for rf are as follows:

F	Fixed
V	Variable
U	Unidentified length
FB	Fixed blocked
VB	Variable blocked

  The values have the same meaning as the equivalent IBM Job Control Language (JCL) specification. The default is U.

- ,BLKSIZE=nnnn

  Defines the maximum block length in 8-bit byte characters. The parameter nnnn is a decimal count. BLKSIZE has the same meaning as the equivalent IBM JCL specification. The default is 137.

- ,LRECL=nnnn

  Defines the maximum logical record size in 8-bit byte characters. The parameter nnnn is a decimal count. LRECL has the same meaning as the equivalent IBM JCL specification. If omitted, the value of LRECL is assumed to be the same as the value of BLKSIZE.

- ,CODE (A/C)

  Defines the character set code present on the input file as follows:

A	ASCII
C	EBCDIC

  If the input file is on magnetic tape, the CODE parameter must match the code parameter used on the input file LABEL control statement.

- ,FOLD

  Causes output to be folded to a 64-character set (6-bit display code representation) for printing. Special characters that do not have 64-character set representation are not printed. If the parameter is omitted, uppercase and lowercase information is preserved for printing.

When a print file is folded, a system restriction prohibits the character pair :: from occuring at certain points in a print line corresponding to the lower 12-bits of central memory words. COPY8P examines the data to determine whether the character pair :: is present. If COPY8P finds the character pair :: in a position that would result in inadvertent line termination, COPY8P replaces the second : with a space.

- ,FMT=f

  Defines the print spacing convention to be used. The parameter f is defined as follows:

  1 Single space

  2 Double space

  3 Triple space

  A Indicates the first character of each line image is assumed to contain a format control character. The carriage control characters recognized when A is specified are shown in table 5-1.

  M Indicates the first character byte of each line image is one of the IBM printer hardware control codes or commands. Table 5-2 shows the IBM character byte codes recognized if M is specified.

TABLE 5-1. FORMAT CONTROL CHARACTERS WHEN A IS SPECIFIED ON FMT.

Character	Intended Action
+	No space before printing
blank	Single space before printing
0	Double space before printing
-	Triple space before printing
1	Eject page before printing
any other	Single space before printing

OUTPUT OF COPY8P FILES

If an output file produced by COPY8P is to be printed with full uppercase and lowercase characters, you must direct the file to an extended character set printer by the use of the following ROUTE control statement:

```
ROUTE,lfn,DC=PR,IC=ASCII,EC=A9.
```

TABLE 5-2. CHARACTERS RECOGNIZED WHEN M IS SPECIFIED ON FMT

Hexadecimal Code	Intended Action
Control Codes:	
01	No space after printing
09	Single space after printing
11	Double space after printing
19	Triple space after printing
89	Eject page after printing
any other	Single space after printing
Control Commands:†	
0B	Single space
13	Double space
1B	Triple space
8B	Eject page

†The remainder of the line is not printed if control commands are used.

If the FOLD option of the COPY8P control statement is used, the output file contains only uppercase characters. The file can either be copied directly to the job OUTPUT file, or can be sent to the printer by using the following ROUTE control statement:

```
ROUTE,lfn,DC=PR.
```

The preceding formats of the ROUTE control statement are explained in section 6.

CHARACTER SET RESTRICTIONS

The 95-graphic ASCII character subset is the extended printer character set available on the CDC 596-6 model print train (refer to appendix F). Certain EBCDIC characters input to COPY8P might be converted to other graphics as shown in table 5-3. If an ASCII or EBCDIC character is input that does not correspond to an available character, a blank is printed. The print conversion for the COPY8P utility is not identical to that given in appendix A for other routines.

TABLE 5-3. COPY8P PRINT CONVERSION TABLE

Full (95-character) EBCDIC Set			Full (95-character) ASCII Set			Folded† (64-character) CDC Set		Folded† (64-character) ASCII Subset	
	prints as →					folds to →		or	
SP	@	`	SP	@	`	SP	≤	SP	@
!	A	a	!	A	a	∨	A	!	A
"	B	b	"	B	b	≠	B	"	B
#	C	c	#	C	c	≡	C	#	C
$	D	d	$	D	d	$	D	$	D
%	E	e	%	E	e	%	E	%	E
&	F	f	&	F	f	^	F	&	F
′	G	g	′	G	g	↑	G	′	G
(	H	h	(	H	h	(	H	(	H
)	I	i	)	I	i	)	I	)	I
*	J	j	*	J	j	*	J	*	J
+	K	k	+	K	k	+	K	+	K
,	L	l	,	L	l	,	L	,	L
−	M	m	−	M	m	−	M	−	M
.	N	n	.	N	n	.	N	.	N
/	O	o	/	O	o	/	O	/	O
0	P	p	0	P	p	0	P	0	P
1	Q	q	1	Q	q	1	Q	1	Q
2	R	r	2	R	r	2	R	2	R
3	S	s	3	S	s	3	S	3	S
4	T	t	4	T	t	4	T	4	T
5	U	u	5	U	u	5	U	5	U
6	V	v	6	V	v	6	V	6	V
7	W	w	7	W	w	7	W	7	W
8	X	x	8	X	x	8	X	8	X
9	Y	y	9	Y	y	9	Y	9	Y
:	Z	z	:	Z	z	:	Z	:	Z
;	¢	{	;	[	{	;	[	;	[
<	\	:	<	\	:	<	≥	<	\
=	!	}	=	]	}	=	]	=	]
>	¬	~	>	⌒	~	>	¬	>	⌒
?	_		?	_		↓	↱	?	_
	other			other			space		space

†In folded representations, column 2 stands for both columns 2 and 3 of the full table.

COPY8P EXAMPLE

Figure 5-1 shows a CYBER Control Language (CCL) procedure file that uses the COPY8P utility to copy an IBM format file, EB1, to a CDC compatible file. The file is folded to a 64-character set, 6-bit display code representation for printing. The ROUTE control statement is used to route the file to the printer. Figure 5-2 shows the resulting output.

```
S358244038ADAMS      BARBARA   220070900141400
S570327591BURCHELL   DONALD    220070670152200
S463445549CLEVELAND  WILLIAM   220070200170500
S207243050DAVIES     DAVID     220070510219000
S571649674ELLIS      ALAN      220070680081500
S562460661FERRERA    ROBERT    220070060137100
$REVERT.CCL
```

Figure 5-2. Sample Output From COPY8P Utility

```
.PROC,C1.
LIBRARY(BIT8LIB)
GET,EB1.
COPY8P,EB1,EBNEW,RECFM=F,BLKSIZE=80,LRECL=80,CODE=C,FMT=1,FOLD.
ROUTE(EBNEW,DC=PR)
REVERT,C1.
```

Figure 5-1. NOS CCL Procedure File Used With COPY8P Utility

INTERFACE OPERATION 6

When you use the 8-bit subroutines to process extended character set (ASCII and EBCDIC) files, a variety of system interfaces occur among the 8-bit subroutines, the operating system, CYBER Record Manager (CRM), and the CYBER Loader.

OPERATING SYSTEM INTERFACE

Character set and code set support varies by operating system. Appendix A contains information about the character sets and code sets supported by both the NOS and NOS/BE operating systems.

All information in the following subsections refers to both the NOS and NOS/BE operating systems unless otherwise stated.

TAPE FILES

Either 8-bit ASCII or 8-bit EBCDIC files on 9-track IBM sequential tapes can be processed by the 8-bit subroutines. NOS requires a RESOURC control statement for any job that uses more than one tape concurrently.

You should make any 9-track tapes that are to be processed by the 8-bit subroutines or the COPY8P utility available to the job by the LABEL control statement. A complete description of the 9-track parameters of the LABEL control statement can be found in the appropriate operating system reference manual.

Provided that the data format is identical to that of IBM tapes read by CRM, input tape files can be copied to disk or other devices that can be substituted for tapes.

CARD FILES

All cards files input to the 8-bit subroutines must be binary cards that do not contain sequence numbers or checksums.

To create an input file in binary form (literal input) under NOS, the deck must be preceded and followed by flag cards with a 5/7/9 punch in column 1 and a 4/5/6/7/8/9 punch in column 2. The deck must be in a system record by itself (preceded and followed by a 7/8/9 card).

To create an input file in binary form (free-form binary format) under NOS/BE, the binary deck must be preceded and followed by flag cards with punches in all 12 rows of both column 1 and column 2 (or any other column as long as the cards are identical). The deck must be in a system-logical record by itself (preceded by and followed by a 7/8/9 card).

Each binary card can contain 80 columns of 12-bit data. Internally the operating system receives the card column bit pattern from row 12 through row 9, reading from left to right. The deck setup of a literal input data deck and a free-form binary data deck is shown in figure 6-1.

```
Job Statement
USER Statement        NOS only
CHARGE Statement      NOS only
ACCOUNT Statement     NOS/BE only
  .
  .
  .
7/8/9                 End-of-record
Flag card

   Binary Deck

Flag card
                      (End-of-file; signals end-
6/7/9                 <of-input to the loader
                      ( (NOS only)

7/8/9                 (Two end-of-record cards;
7/8/9                 <signals end-of-input to
                      (the loader (NOS/BE only)

   Rest of the Job
     .
     .
     .
6/7/8/9               End-of-information
```

Figure 6-1. Deck Setup for Binary Input

If a run is to produce card output files punched in absolute binary format, you must specify the disposition code on the ROUTE control statement as shown in figure 6-2.

PRINT FILES

When an extended character set output file produced by either COPY8P or the 8-bit subroutines is to be printed with full uppercase and lowercase characters, the file must be directed to the CDC 595-6 printer through the ROUTE control statement shown in figure 6-2.

The format of the ROUTE[†] control statement to be used when routing a file to the punch queue is as follows:

NOS:

```
ROUTE,lfn,DC=P8.
```

NOS/BE:

```
ROUTE,lfn,DC=P80C
```

The format of the ROUTE[†] control statement to be used when routing an extended character set output file to the print queue is as follows:

NOS and NOS/BE:

```
ROUTE,lfn,DC=PR,IC=ASCII,EC=A9.
```

lfn	Names the file to be routed

DC=P8
DC=P80C
	Indicates the disposition code is 80-column punch

DC=PR	Indicates the file can be printed on any printer.
IC=ASCII	Defines the internal characteristics of a file.
EC=A9	Defines the external characteristics of the print file. A9 indicates the ASCII 95-character set. If a characteristic is specified that is invalid at the site, the file cannot be output.

[†] Refer to the appropriate operating system reference manual for more information about the ROUTE control statement.

Figure 6-2. Formats of the ROUTE Control Statement Used With the 8-Bit Subroutines

CYBER RECORD MANAGER INTERFACE

Since CYBER Record Manager cannot block or deblock records in IBM format, the user must provide a FILE control statement for each sequential IBM file of 8-bit bytes processed by the 8-bit subroutines. The COPY8P utility does not require a FILE control statement. The control statement is written as follows:

FILE(lfn, keyword=option$_n$,...)

- lfn

  Names the file to be processed

- keyword=option

  Symbolic name of the FIT field and the option selected

Parameters in the FILE control statement provide record and block information that is used to update the file information table when a given file is opened for the first time in the job. Refer to the CYBER Record Manager reference manual for complete information about the FILE control statement.

TAPE FILES

The parameters required in the FILE control statement for tape files are:

- RT=S

  Record type; S indicates system record type.

- MBL=nnnn

  Maximum Block Size; nnnn indicates block size in terms of 6-bit character bytes. The parameter nnnn is used instead of BLKSIZE since BLKSIZE refers to 8-bit character bytes and CRM expects sizes in terms of 6-bit character bytes.

  nnnn=(BLKSIZE times 4)/3; a fraction must be rounded up to the next higher integer.

- MRL=nnnn

  Maximum Record Length; nnnn indicates record size in terms of 6-bit character bytes. The parameter nnnn is used instead of BLKSIZE because BLKSIZE refers to 8-bit character bytes and CRM expects sizes in terms of 6-bit character bytes.

  nnnn=(BLKSIZE times 4)/3; a fraction must be rounded up to the next higher integer.

- CM=NO

  Conversion mode; NO indicates CRM does not convert sequential tape files from external to internal code.

For 9-track tape, files two additional FILE control statement parameters are necessary for noise skipping:

- MNR=24

  Minimum Record Length; the 24 indicates the system only recognizes records over 24 character bytes long.

- MNB=24

  Minimum Block Length; the 24 indicates the system only recognizes blocks over 24 character bytes long.

Any block smaller than MNR or MNB is considered noise and is discarded by the system.

For a COBOL calling program, the clause BLOCK CONTAINS nnn CHARACTERS should be included with each description of an 8-bit byte file; otherwise, CRM might diagnose an inadequate buffer size (BFS) specification.

CARD FILES

You must specify the following information to CRM as FILE control statement parameters to permit reading of literal input (NOS) or free-form (NOS/BE) binary card image files. This information must also be specified to permit writing of card image files that are to be routed to punch in absolute (NOS) or free-form (NOS/BE) binary format:

- BT=C

  Block type; C indicates character count

- RT=F

  Record type; F indicates fixed length

- MRL=160

  Maximum record length of 160 6-bit character bytes

PRINT FILES

For CRM handling of files formatted for the printer, you must specify the following FILE control statement parameters:

- BT=C

  Block type; C indicates character count

- RT=U

  Record type; U indicates undefined length

- MRL=280

  Maximum record length of 280 6-bit character bytes

LOADER INTERFACE

The 8-bit subroutines modules are in the library BIT8LIB. The loader must be told to use this library. The 8-bit modules can be accessed by prefacing the load sequence with the LDSET loader option as follows:

```
LDSET(LIB=BIT8LIB)
```

Usage of the 8-bit subroutines requires an additional 14K (octal) words if all modules are referenced.

OMITTING UNNEEDED 8-BIT MODULES

The loader directive LDSET is used to control the load process under a variety of conditions. If not all features of the 8-bit subroutines are needed, some 8-bit modules can be omitted from the library being loaded to conserve space in the field length. The OMIT option of the LDSET statement is as follows:

```
LDSET(OMIT=name/name...)
```

The names given are entry points of the modules to be omitted from the load. The entry point names and the corresponding module functions are shown in table 6-1. Because the entry point names contain the special character . (period), you must surround the names by $ (dollar sign) characters in the name list as follows:

```
LDSET(OMIT=$T.8TSTC$/$T8.TSTT$)
```

TABLE 6-1. ENTRY NAME AND 8-BIT MODULE FUNCTION

Entry Point Name	Module Function
T8.TST6	Selector expressions on internal data
T8.TSTT	Selector expressions on tape data
T8.TSTC	Selector expressions on card data
T8.CN6T	Conversion, internal to tape
T8.CN6C	Conversion, internal to card
T8.CN6P	Conversion, internal to print
T8.CNT6	Conversion, tape to internal
T8.CNC6	Conversion, card to internal

LOADER CONSIDERATIONS

The XWRITE and XREAD subroutines do not reference any module entry point names, rather the subroutines depend on XFILE to place an appropriate address in the working storage area. The entry names shown in table 6-1 are referenced only by XFILE.

When using overlays or segments, you should place XFILE in the main overlay or root segment. You can place XREAD/XREREAD or XWRITE in the main, primary or secondary overlay. You must ensure that the working code is placed in a module common to all, as XREAD, XWRITE, and XFILE share subroutines and some common areas.

COBOL USAGE 7

An IBM file created by an IBM COBOL source program can differ in data type from a CDC file created by a CDC COBOL program. This section clarifies the differences in data type declarations, storage allocations, and corresponding Tm values, that you can encounter when converting to or from an IBM compatible file using a COBOL program.

IBM COBOL DATA FORMATS

An IBM COBOL file can contain any combination of the following data types:

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	(S)

An IBM COBOL program cannot create 128-bit extended-precision floating-point data. Any of the IBM data types can also be considered bit string data (B).

The letters enclosed in parentheses are the corresponding T values as described in table 2-1 of section 2. To select the appropriate m value for each of the data types, refer to table 2-1 of section 2. A complete description of IBM data type formats is given in appendix E.

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

The relationship among IBM COBOL usage clauses, picture clauses, IBM byte storage allocation, and the corresponding 8-bit Tm values to be used in converting data types are shown in table 7-1.

The examples shown in table 7-2 show the relationship between some IBM COBOL elementary data description entries and the corresponding 8-bit Tm values. Table 7-1 was used to compute the Tm value. If no usage clause is specified, usage is assumed to be DISPLAY.

For each example, m is the size of the data item in 8-bit bytes. The m specification must be omitted for COMP, COMP-1, and COMP-2 data items.

Calculation of m for COMP-3 data is more complicated than for other data types because COMP-3 data is packed two digits per 8-bit byte. The sign and the low order digit appear together in the rightmost byte. The following algorithm was used to compute m (the number of 8-bit bytes occupied by packed decimal data) for examples 12 and 13 in table 7-2.

- D was set to the number of digits specified in the PICTURE clause. (For EG12, D=11; for EG13, D=8.)
- E was obtained by adding 1 to D to allow for the sign half-byte that always appears in the right half of the rightmost byte along with the low order digit. (For EG12, E=12; for EG13, E=9.)
- If E was even, F was set to E. (For EG12, F=12.)
- If E was odd, F was set to E+1. (For EG13, F=13.)
- F was divided by 2 to obtain the proper value for m. (For EG12, m=6; for EG13, m=5.)

CDC COBOL DATA FORMATS

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

6-bit characters	(X)
12-bit byte containing 7-bit ASCII characters	(A)
12-bit byte containing 8-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 sign overpunch	(S)
Numeric display, unsigned	(N)
Numeric display, leading zeros suppressed	(Z)

TABLE 7-1. IBM COBOL - Tm VALUES

IBM COBOL USAGE Category	Picture Format	IBM Boundary Alignment	T	IBM 8-Bit Bytes Used (m)
DISPLAY	Alphabetic	None	X	1 per character or digit (except for V in external floating-point), limit 18
DISPLAY	Alphanumeric	None	X	1 per character or digit (except for V in external floating-point), limit 18
DISPLAY	Report format	None	X	1 per character or digit (except for V in external floating-point), limit 18
DISPLAY	External floating-point form	None	X	1 per character or digit (except for V in external floating-point), limit 18
DISPLAY	Numeric unsigned	None	N or X	1 per digit, limit 18
DISPLAY	Numeric signed	None	S	1 per digit, limit 18
COMP (binary)	1 to 4 digits	Half-word	H	2
COMP (binary)	5 to 9 digits	Full-word	W	4
COMP (binary)	10 to 18 digits	Full-word	G	8
COMP-1 (internal floating-point)	Not applicable	Full-word	F	4 (short-precision)
COMP-2 (internal floating-point)	Not applicable	Double-word	L	8 (long-precision)
COMP-3 (packed decimal)	Numeric unsigned or signed	None	P	1 byte per 2 digits plus 1 byte for low-order digit and sign

TABLE 7-2. RELATIONSHIP BETWEEN IBM COBOL ELEMENTARY DATA DESCRIPTION ENTRIES AND CORRESPONDING 8-BIT Tm VALUES.

Example	Elementary Data Entry	USAGE Category	Tm
1	01 EG1 PIC A(20).	DISPLAY (alphabetic)	X20
2	01 EG2 PIC X(53).	DISPLAY (alphanumeric)	X53
3	01 EG3 PIC $999,999.99-	DISPLAY (report form)	X12
4	01 EG4 PIC +.9(8)E+99.	DISPLAY (external floating-point form)	X14
5	01 EG5 PIC 9(8)V99.	DISPLAY (numeric unsigned)	N10 or X10
6	01 EG6 PIC S9(8)V99.	DISPLAY (numeric signed)	S10
7	01 EG7 PIC S9(3)v9 COMPUTATIONAL.	COMP, 1 thru 4 digits	H
8	01 EG8 PIC 9(6)COMPUTATIONAL.	COMP, 5 thru 9 digits	W
9	01 EG9 PIC 9(8)V9(9) COMPUTATIONAL.	COMP, 10 thru 18 digits	G
10	01 EG10 COMPUTATIONAL-1.	COMP-1	F
11	01 EG11 COMPUTATIONAL-2.	COMP-2	L
12	01 EG12 PIC S9(5)V9(6) COMP-3.	COMP-3	P6
13	01 EG13 PIC 9(4)V9(4) COMP-3.	COMP-3	P5

The letters in parentheses are the corresponding T values as described in table 2-1 of section 2. The A and C values for T are unique to 8-bit subroutine processing. A CDC COBOL program can neither create nor process 60-bit integer data. Any of the CDC data types can be considered as bit string data (B). The description of CDC internal format types appears in appendix E.

Table 7-3 shows the relationship among CDC COBOL USAGE clauses, PICTURE clauses, CDC storage allocation, and the corresponding 8-bit Tm values used in processing such data types.

The examples in table 7-4 show the relationship between some CDC COBOL elementary data description entries and the corresponding 8-bit Tm values derived by using table 7-3. USAGE is DISPLAY unless otherwise specified.

Except for examples 6 and 7, m is the size of the data item in 6-bit bytes. For 7-bit ASCII or 8-bit EBCDIC stored in 12-bit internal CDC data items, m is the size of the data item in 12-bit bytes. The m must be omitted for COMP-1 and COMP-2 data items.

COBOL SAMPLE PROGRAM

Figure 7-1 shows Tape-Con, a COBOL program used to convert 27 logical records on an IBM EBCDIC file (EB1). EB1 has been copied from tape to disk. Each record is 80 characters long. Each block contains one record. The file contains both uppercase and lowercase data. Each record is converted to display code with the lowercase data folded to the uppercase equivalent.

The control statements used to run the job interactively and some sample records are also shown in figure 7-1.

TABLE 7-3. CDC COBOL 5 - Tm VALUES

CDC COBOL Usage	Picture Format	Boundary Alignment	T	CDC 6-Bit Bytes Used in Storage Allocation (m)
DISPLAY	Alphabetic	None	X	1 per character or digit (except for V)
DISPLAY	Alphanumeric	None	X	1 per character or digit (except for V)
DISPLAY	Edited report form, leading zeros suppressed	None	X	1 per character or digit (except for V)
COMPUTATIONAL or DISPLAY	Numeric unsigned	None	N X	1 per digit, limit 18
COMPUTATIONAL or DISPLAY	Numeric signed	None	S	1 per digit, limit 18
DISPLAY	Edit, leading zeros suppressed	None	Z	1 per digit or space, limit 18
COMP-1	1 to 14 digits	Word	I	10
COMP-2	1 to 14 digits	Word	E	10
COMP-4	1 to 14 digits, numeric unsigned	Byte	B	Number of bits required to store the maximum decimal value that can be represented by the specified number of digits in the PICTURE clause, divided by 6 and rounded up
COMP-4	1 to 14 digits, numeric signed	Byte	B	Number of bits required by unsigned items plus 1, divided by 6 and rounded up

Control Statements:

```
GET,CBLFIL.
GET,EB1.
FILE,EB1,RT=S,MRL=107,MBL=107,CM=NO.
COBOL5(I=CBLFIL)
LDSET(LIB=BIT8LIB)
LGO.
```

COBOL 5 Source Program on File CBLFIL:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.                   TAPE-CON.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT FILE2 ASSIGN TO "EB1".
DATA DIVISION.
FILE SECTION.
FD FILE2
    LABEL RECORDS ARE OMITTED
    BLOCK CONTAINS 80 CHARACTERS.
WORKING-STORAGE SECTION.
01  REC-1                     PIC X(80).
01  WS1                       PIC X(170)                VALUE SPACES.
01  SZ                        PIC 99  USAGE COMP-1      VALUE 17.
01  STAT                              USAGE COMP-2      VALUE ZERO.
01  FILE-CNT                  PIC 99  USAGE COMP-1      VALUE ZERO.
01  FSTRING PIC X(50)                                   VALUE
    "(FT=T,USE=R,RECFM=F,BLKSIZE=80,LRECL=80)".
PROCEDURE DIVISION.
OPENING.
    OPEN INPUT FILE2.
    ENTER COMPASS XFILE USING FILE2, WS1, FSTRING, SZ.
    PERFORM RLOOP UNTIL FILE-CNT IS EQUAL TO 26.
    PERFORM PROCESS-EOF.
RLOOP.
    ENTER COMPASS XREAD USING WS1, REC-1, "(X80)", STAT.
    ADD 1 TO FILE-CNT.
    DISPLAY REC-1.
    IF STAT EQUALS 1 DISPLAY "BAD INPUT DATA:".
    IF STAT EQUALS 2 DISPLAY "READ ERROR".
PROCESS-EOF.
    DISPLAY "EOF ENCOUNTERED".
    CLOSE FILE2.
    STOP RUN.
```

Sample Output:

```
S358244038ADAMS       BARBARA    220070900141400
S570327591BURCHELL    DONALD     220070670152200
S463445549CLEVELAND   WILLIAM    220070200170500
S207243050DAVIES      DAVID      220070510219000
S571649674ELLIS       ALAN       220070680081500
S562460661FERRERA     ROBERT     220070060137100
```

Figure 7-1. COBOL 5/8-Bit Subroutines Interactive Job

TABLE 7-4. RELATIONSHIP BETWEEN CDC COBOL ELEMENTARY DATA DESCRIPTION ENTRIES AND CORRESPONDING Tm VALUES.

Example	Elementary Data Entry	USAGE Category	Tm
1	01 AG1 PIC A 20.	DISPLAY (alphabetic)	X20
2	01 AG2 PIC X 53.	DISPLAY (alphanumeric)	X53
3	01 AG3 PIC $999,999,99-.	DISPLAY (report form)	X12
4	01 AG4 PIC 9(8)V99.	DISPLAY (numeric unsigned)	N10 or X10
5	01 AG5 PIC S9(9)V99.	DISPLAY (numeric signed)	S10
6	01 AG6 PIC X(100).	DISPLAY (12-bit ASCII)	A50
7	01 AG7 PIC X(100).	DISPLAY (12-bit EBCDIC)	C50
8	01 AG8 PIC Z(9).	DISPLAY (leading zeros suppressed)	Z9
9	01 AG9 PIC S9(10) USAGE COMP-1.	COMP-1, 1 thru 14 digits	U
10	01 AG10 PIC S9(16) USAGE COMP-1.	COMP-1, 15 thru 18 digits	D
11	01 AG11 USAGE COMPUTATIONAL-2.	COMP-2	E

FORTRAN EXTENDED 4 USAGE 8

An IBM file created by an IBM FORTRAN source program can differ in data type from a CDC file created by FORTRAN Extended 4. This section clarifies the differences in data type declarations, storage allocations, and corresponding Tm values, that you can encounter when converting to or from an IBM compatible file using a FORTRAN program.

IBM FORTRAN DATA FORMATS

IBM FORTRAN programs can use 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. The length specification for each variable determines the number of bytes reserved for each variable type. Table 8-1 gives the associated length specification for constants and variables.

TABLE 8-1. IBM FORTRAN - CONSTANT AND VARIABLE SIZES

Type	8-Bit Bytes Allocated
Constant:	
Integer	2, 4, or 8
Real	4 (single-precision) 8 (double-precision) 16 (long-precision)
Complex	8 (two single-precision) 16 (two double-precision)
Logical	1 or 4
Literal	1≤n (n is string length)
Packed Decimal	1 byte for each 2 digits
Variable:	
Integer	4 (default) 2 or 8 (optional)
Real	4 (default) 8 or 16 (optional)
Complex	8 (default) 16 (optional)
Logical	4 (default) 1 (optional)

A tape file created by an IBM FORTRAN program can contain combinations of the following data types:

8-bit byte ASCII or EBCDIC characters	(X)
Half-word integers, 16 bits	(H)
Full-word integers, 32 bits	(W)
32-bit floating point	(F)
64-bit floating point	(L)

The letters enclosed in parentheses are the corresponding T values as described in table 2-1 of section 2. An IBM FORTRAN program cannot create 64-bit double word integers or 128-bit extended-precision floating point data. All of the IBM data types can be processed as bit string data (B).

Table 2-1 in section 2 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 E.

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

Table 8-2 contains examples of only explicit type declarations. COMPLEX declarations do not appear because the complex variable is composed of two real data items and follows the conventions for REAL data types. You must determine the conversion of LOGICAL data types.

CDC FORTRAN DATA FORMATS

The number of central memory words needed to represent a constant or variable in a CDC FORTRAN program is determined by the constant or variable type. Table 8-3 shows the constant and variable types and the corresponding 6-bit byte length allocations.

Internal records created by a CDC FORTRAN Extended 4 program can contain combinations of the following data types. The letter in parentheses is the corresponding T value as determined from table 2-1 of section 2.

6-bit display code characters	(X)
Full-word integer, 60 bits	(I)
Single-word floating point, 60-bits	(E)
Double-precision floating point, 120 bits	(D)

TABLE 8-2. IBM FORTRAN - Tm VALUES

IBM FORTRAN Type Declaration	IBM Boundary Alignment	T	8-Bit Bytes Used†
Integer*2	Half-word	H	2
Integer Integer*4	Full-word	W	4
Integer*8	Double-word	G	8
Real Real*4	Full-word	F	4 (Single-precision)
Real*8 Double-precision	Double-word	L	8 (Double-precision)
Real*16	Double-word	E	16
Logical	Full-word	Usage defined	4
Logical*1	None	Usage defined	1

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

All CDC data types can be processed as type B bit string data. Table 2-1 in section 2 should be consulted to determine the appropriate m value. While a full-word integer is a data type, integers used in multiplication and division operations are truncated to 48 bits. More information on this subject appears in appendix E and in the FORTRAN Extended 4 reference manual.

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

COMPLEX type declarations do not appear in the table because the complex variable is composed of two real data items and follows the conventions for REAL data types. You must determine the manner in which LOGICAL data items are converted.

TABLE 8-3. CDC FORTRAN - CONSTANT AND VARIABLE SIZES

Type	FORTRAN Extended 4 6-Bit Bytes Allocated
Constant:	
COMPLEX	20
DOUBLE PRECISION	20 (two real constants)
HOLLERITH	1≤n (n is string length)
INTEGER	10
LOGICAL	10
REAL	10
Variable:	
COMPLEX	20
DOUBLE PRECISION	20
INTEGER	10
LOGICAL	10
REAL	10

SAMPLE PROGRAM

Figure 8-1 shows CONVERT, a FORTRAN Extended 4 program used to convert EB1, an IBM EBCDIC file. EB1 has been copied from tape to disk. The file contains 27 logical records; each record is 80 characters long. Each block contains one record. The file contains both uppercase and lowercase data. Each record is converted to display code with lowercase data folded to the uppercase equivalent. Control statements used to run the program interactively are shown in figure 8-2. Some sample output is shown in figure 8-3.

TABLE 8-4. CDC FORTRAN - Tm VALUES

Type Declaration	CDC Boundary Alignment	T	FORTRAN Extended 4 6-Bit Bytes Used†
INTEGER	Full-word	I	10
REAL	Full-word	X	10
DOUBLE PRECISION	Double-word	D	20
LOGICAL	Full-word	Usage Defined	20

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

```
             PROGRAM CONVERT(INPUT,OUTPUT,EB1,TAPE6=EB1)
             INTEGER REC1(8),WKSP(17),CNT
             STAT=0.0
             CNT=0
             CALL XFILE(6,WKSP,"(FT=T,USE=R,RECFM=F,BLKSIZE=80,LRECL=80)",17)
        1    CNT=CNT+1
             IF(CNT .GT. 26)CALL EXIT
             CALL XREAD(WKSP,REC1,"(X80)",STAT)
             PRINT 30,REC1
             IF(STAT .NE. 0.0)PRINT 40,STAT
             GO TO 1
       30    FORMAT(8A10)
       40    FORMAT(F4.1)
             STOP
             END
```

Figure 8-1. FORTRAN Extended 4 Program Using The 8-Bit Subroutines

```
FORTRAN Extended 4 Control Statements:

    GET,CONVERT.
    GET,EB1.
    FILE,EB1,RT=S,MRL=107,MBL=107,CM=NO.
    FTN4,I=CONVERT.
    LDSET,LIB=BIT8LIB.
    LGO.
```

Figure 8-2. Control Statements Used to Run FORTRAN Extended 4 Program

```
S358244038ADAMS         BARBARA   220070900141400
S570327591BURCHELL      DONALD    220070670152200
S463445549CLEVELAND     WILLIAM   220070200170500
S207243050DAVIES        DAVID     220070510219000
S571649674ELLIS         ALAN     F220070680081500
S562460661FERRERA       ROBERT    220070060137100
S148169725GRAME         CARL      220070800105000
S566208909HARVEY        LAURENCE E220070450383500
S132246243IMMITT        SALVATOREJ220070690204300
S572548172JENSEN        HOWARD   M220070070091250
```

Figure 8-3. Sample Output From FORTRAN Extended 4 Program

CHARACTER DATA INPUT, OUTPUT, AND CENTRAL MEMORY REPRESENTATION

This appendix describes the code and character sets used by host computer operating system local batch device drivers, magnetic tape drivers, and terminal communication products. Some software products assume that certain graphic or control characters are associated with specific binary code values for collating or syntax processing purposes.

All references within this manual to the ASCII character set or the ASCII code set refer to the character set and code set defined in the American National Standard Code for Information Interchange (ASCII, ANSI Standard X3.4-1977). References in this manual to the ASCII character set do not necessarily apply to the ASCII code set.

CHARACTER SETS AND CODE SETS

A character set differs from a code set. A character set is a set of graphic and/or control character symbols. A code set is a numbering system used to represent each character within a character set. Characters exist outside the computer system and communication network; codes are received, stored, retrieved, and transmitted within the computer system and network.

GRAPHIC AND CONTROL CHARACTERS

A graphic character can be displayed at a terminal or printed by a line printer. Examples of graphic characters are the characters A through Z, a blank, and the digits 0 through 9. A control character initiates, modifies, or stops a control operation. An example of a control character is the backspace character, which moves the terminal carriage or cursor back one space. Although a control character is not a graphic character, some terminals can produce a graphic representation when they receive a control character.

CODED AND BINARY CHARACTER DATA

Character codes can be interpreted as coded character data or as binary character data. Coded character data is converted from one code set representation to another as it enters or leaves the computer system; for example, data received from a terminal or sent to a magnetic tape unit is converted. Binary character data is not converted as it enters or leaves the system. Character codes are not converted when moved within the system; for example, data transferred to or from mass storage is not converted.

The distinction between coded character data and binary character data is important when reading or punching cards and when reading or writing magnetic tape. Only coded character data can be properly reproduced as characters on a line printer. Only binary character data can properly represent characters on a punched card when the data cannot be stored as display code.

The distinction between binary character data and characters represented by binary data (such as peripheral equipment instruction codes) is also important. Only binary noncharacter data can properly reproduce characters on a plotter.

FORMATTED AND UNFORMATTED CHARACTER DATA

Products can interpret character codes as formatted character data or as unformatted character data. A product can store or retrieve formatted data in the form of the codes described for coded character data in the remainder of this appendix, or the product can alter that formatted data to another form during storage or retrieval; for example, a 1 can be stored as a character code or as an integer value. A product can treat unformatted data either as coded character data or as binary character data.

NETWORK OPERATING SYSTEMS

NOS and NOS/BE support the following character sets:

- CDC graphic 64-character set
- CDC graphic 63-character set
- ASCII graphic 64-character set
- ASCII graphic 63-character set
- ASCII graphic 95-character set

In addition, NOS supports the ASCII 128-character graphic and control set.

Each installation must select either a 64-character set or a 63-character set. The differences between the codes of a 63-character set and the codes of a 64-character set are described under Character Set Anomalies. Any reference in this appendix to a 64-character set implies either a 63- or 64-character set unless otherwise stated.

NOS supports the following code sets to represent its six listed character sets in central memory:

- 6-bit display code
- 12-bit ASCII code
- 6/12-bit display code

NOS/BE supports the following code sets to represent its five listed character sets in central memory:

- 6-bit display code
- 12-bit ASCII code

Under both NOS and NOS/BE, the 6-bit display code is a set of 6-bit codes from 00_8 to 77_8, inclusive.

Under both NOS and NOS/BE, the 12-bit ASCII code is the ASCII 7-bit code (as defined by ANSI Standard X3.4-1977) right-justified in a 12-bit byte. Assuming that the bits are numbered from the right starting with 0, bits 0 through 6 contain the ASCII code, bits 7 through 10 contain zeros, and bit 11 distinguishes the 12-bit ASCII 0000_8 code from the 12-bit 0000_8 end-of-line byte. The 12-bit codes are 0001_8 through 0177_8 and 4000_8.

Under NOS, the 6/12-bit display code is a combination of 6-bit codes and 12-bit codes. The 6-bit codes are 00_8 through 77_8, excluding 74_8 and 76_8. (The interpretation of the 00_8 and 63_8 codes is described under Character Set Anomalies later in this appendix.) The 12-bit codes begin with either 74_8 or 76_8 and are followed by a 6-bit code. Thus, 74_8 and 76_8 are considered escape codes and are never used as 6-bit codes within the 6/12-bit display code set. The 12-bit codes are 7401_8, 7402_8, 7404_8, 7407_8, and 7601_8 through 7677_8. All other 12-bit codes ($74xx_8$ and 7600_8) are undefined.

CHARACTER SET ANOMALIES

The operating system input/output software and some products interpret two codes differently when the installation selects a 63-character set rather than a 64-character set. If an installation uses a 63-character set, the colon graphic character is always represented by a 63_8 display code, display code 00_8 is undefined (it has no associated graphic or punched card code), and the % graphic does not exist.

However, under NOS, if the installation uses a 64-character set, output of a 7404_8 6/12-bit display code or a 00_8 display code produces a colon. A colon can be input only as a 7404_8 6/12-bit display code. The use of undefined 6/12-bit display codes in output files produces unpredictable results and should be avoided.

Under NOS/BE, if the installation uses a 64-character set, output of a 00_8 display code produces a colon. Display code 63_8 is the colon when a 63-character set is used. The % graphic and related card codes do not exist and translations yield a blank (55_8).

Under both NOS and NOS/BE, two consecutive 00_8 codes can be confused with the 12-bit 0000_8 end-of-line byte and should be avoided.

CHARACTER SET TABLES

The character set tables A-1 and A-2 are designed so that you can find the character represented by a code (such as in a dump) or find the code that represents a character. To find the character represented by a code, you look up the code in the column listing the appropriate code set and then find the character on that line in the column listing the appropriate character set. To find the code that represents a character, you look up the character and then find the code on the same line in the appropriate column.

Conversational Terminal Users

Table A-1 shows the character sets and code sets available to an Interactive Facility (IAF) or INTERCOM user at an ASCII code terminal using an ASCII character set. The octal and hexadecimal 7-bit ASCII code for each ASCII character can be used to convert codes from octal to hexadecimal. These octal and hexadecimal values are shown later in this appendix. (Under NOS using network product software, certain Terminal Interface Program commands require specification of an ASCII code.)

IAF Usage

When in normal time-sharing mode (specified by the IAF NORMAL command), IAF assumes the ASCII graphic 64-character set is used and translates all input and output to or from display code. When in ASCII time-sharing mode (specified by the IAF ASCII command), IAF assumes the ASCII 128-character set is used and translates all input and output to or from 6/12-bit display code.

The IAF user can convert a 6/12-bit code file to a 12-bit ASCII code file using the NOS FCOPY control statement. The resulting 12-bit ASCII file can be routed to a line printer but the file cannot be output through IAF.

IAF supports both character mode and transparent mode transmissions through the network. These transmission modes are described under Terminal Transmission Code Sets in this appendix. IAF treats character mode transmissions as coded character data; IAF converts these transmissions to or from either 6-bit or 6/12-bit display code. IAF treats transparent mode transmissions as binary character data; transparent mode communication between IAF and ASCII terminals using any parity setting occurs in the 12-bit ASCII code shown in table A-1.

INTERCOM Usage

By default, INTERCOM displays the ASCII graphic 64-character set and interprets all input and output as display code. Refer to the INTERCOM Reference Manual.

COMPASS and FORTRAN users can elect to use 12-bit ASCII code if the terminal in use supports the code set selected. BASIC users can elect to send and receive lowercase and uppercase character codes using the 12-bit ASCII code if the terminal in use supports the code set selected; BASIC represents coded character data in central memory using 6/12-bit display code in both the NOS and NOS/BE systems.

TABLE A-1. CONVERSATIONAL TERMINAL CHARACTER SETS

ASCII Graphic (64-Character Set)	ASCII Character (128-Character Set)	Octal 6-Bit Display Code	Octal 6/12-Bit Display Code	Octal 12-Bit ASCII Code	ASCII Graphic (64-Character Set)	ASCII Character (128-Character Set)	Octal 6-Bit Display Code	Octal 6/12-Bit Display Code†	Octal 12-Bit ASCII Code
: colon††		00††				^ circumflex		7402	0136
A	A	01	01	0101		: colon††		7404††	0072
B	B	02	02	0102		` grave accent		7407	0140
C	C	03	03	0103		a		7601	0141
D	D	04	04	0104		b		7602	0142
E	E	05	05	0105		c		7603	0143
F	F	06	06	0106		d		7604	0144
G	G	07	07	0107		e		7605	0145
H	H	10	10	0110		f		7606	0146
I	I	11	11	0111		g		7607	1047
J	J	12	12	0112		h		7610	0150
K	K	13	13	0113		i		7611	0151
L	L	14	14	0114		j		7612	0152
M	M	15	15	0115		k		7613	0153
N	N	16	16	0116		l		7614	0154
O	O	17	17	0117		m		7615	0155
P	P	20	20	0120		n		7616	0156
Q	Q	21	21	0121		o		7617	0157
R	R	22	22	0122		p		7620	0160
S	S	23	23	0123		q		7621	0161
T	T	24	24	0124		r		7622	0162
U	U	25	25	0125		s		7623	0163
V	V	26	26	0126		t		7624	0164
W	W	27	27	0127		u		7625	0165
X	X	30	30	0130		v		7626	0166
Y	Y	31	31	0131		w		7627	0167
Z	Z	32	32	0132		x		7630	0170
0	0	33	33	0060		y		7631	0171
1	1	34	34	0061		z		7632	0172
2	2	35	35	0062		{ left brace		7633	0173
3	3	36	36	0063		| vert. line		7634	0174
4	4	37	37	0064		} right brace		7635	0175
5	5	40	40	0065		~ tilde		7636	0176
6	6	41	41	0066		NUL		7640	4000
7	7	42	42	0067		SOH		7641	0001
8	8	43	43	0070		STX		7642	0002
9	9	44	44	0071		ETX		7643	0003
+ plus	+ plus	45	45	0053		EOT		7644	0004
- minus	- minus	46	46	0055		ENQ		7645	0005
* asterisk	* asterisk	47	47	0052		ACK		7646	0006
/ slash	/ slash	50	50	0057		BEL		7647	0007
(l. paren.	(l. paren.	51	51	0050		BS		7650	0010
) r. paren.	) r. paren.	52	52	0051		HT		7651	0011
$ dollar	$ dollar	53	53	0044		LF		7652	0012
= equal to	= equal to	54	54	0075		VT		7653	0013
space	space	55	55	0040		FF		7654	0014
, comma	, comma	56	56	0054		CR		7655	0015
. period	. period	57	57	0056		SO		7656	0016
# number	# number	60	60	0043		SI		7657	0017
[l. bracket	[l. bracket	61	61	0133		DEL		7637	0177
] r. bracket	] r. bracket	62	62	0135		DLE		7660	0020
% percent††	% percent††	63††	63††	0045		DC1		7661	0021
" quote	" quote	64	64	0042		DC2		7662	0022
_ underline	_ underline	65	65	0137		DC3		7663	0023
! exclam.	! exclam.	66	66	0041		DC4		7664	0024
& ampersand	& ampersand	67	67	0046		NAK		7665	0025
' apostrophe	' apostrophe	70	70	0047		SYN		7666	0026
? question	? question	71	71	0077		ETB		7667	0027
< less than	< less than	72	72	0074		CAN		7670	0030
> grtr. than	> grtr. than	73	73	0076		EM		7671	0031
@ coml. at		74				SUB		7672	0032
\ rev. slant	\ rev. slant	75	75	0134		ESC		7673	0033
^ circumflex		76				FS		7674	0034
; semicolon	; semicolon	77	77	0073		GS		7675	0035
	@ coml. at		7401	0100		RS		7676	0036
						US		7677	0037

†Generally available only on NOS, or through BASIC on NOS/BE.

††The interpretation of this character or code depends on its context. Refer to Character Set Anomalies in the text.

TABLE A-2. LOCAL BATCH DEVICE CHARACTER SETS

CDC Graphic (64-Character Set)	ASCII Graphic (64-Character Set)	ASCII Graphic (95-Character Set)	Octal 6-Bit Display Code	Octal 6/12-Bit Display Code†	Octal 12-Bit ASCII Code	Card Keypunch Code	
						026	029
: colon††	: colon††		00††			8-2	8-2
A	A	A	01	01	0101	12-1	12-1
B	B	B	02	02	0102	12-2	12-2
C	C	C	03	03	0103	12-3	12-3
D	D	D	04	04	0104	12-4	12-4
E	E	E	05	05	0105	12-5	12-5
F	F	F	06	06	0106	12-6	12-6
G	G	G	07	07	0107	12-7	12-7
H	H	H	10	10	0110	12-8	12-8
I	I	I	11	11	0111	12-9	12-9
J	J	J	12	12	0112	11-1	11-1
K	K	K	13	13	0113	11-2	11-2
L	L	L	14	14	0114	11-3	11-3
M	M	M	15	15	0115	11-4	11-4
N	N	N	16	16	0116	11-5	11-5
O	O	O	17	17	0117	11-6	11-6
P	P	P	20	20	0120	11-7	11-7
Q	Q	Q	21	21	0121	11-8	11-8
R	R	R	22	22	0122	11-9	11-9
S	S	S	23	23	0123	0-2	0-2
T	T	T	24	24	0124	0-3	0-3
U	U	U	25	25	0125	0-4	0-4
V	V	V	26	26	0126	0-5	0-5
W	W	W	27	27	0127	0-6	0-6
X	X	X	30	30	0130	0-7	0-7
Y	Y	Y	31	31	0131	0-8	0-8
Z	Z	Z	32	32	0132	0-9	0-9
0	0	0	33	33	0060	0	0
1	1	1	34	34	0061	1	1
2	2	2	35	35	0062	2	2
3	3	3	36	36	0063	3	3
4	4	4	37	37	0064	4	4
5	5	5	40	40	0065	5	5
6	6	6	41	41	0066	6	6
7	7	7	42	42	0067	7	7
8	8	8	43	43	0070	8	8
9	9	9	44	44	0071	9	9
+ plus	+ plus	+ plus	45	45	0053	12	12-8-6
- minus	- minus	- minus	46	46	0055	11	11
* asterisk	* asterisk	* asterisk	47	47	0052	11-8-4	11-8-4
/ slash	/ slash	/ slash	50	50	0057	0-1	0-1
(left paren.	(left paren.	(left paren.	51	51	0050	0-8-4	12-8-5
) right paren.	) right paren.	) right paren.	52	52	0051	12-8-4	11-8-5
$ dollar	$ dollar	$ dollar	53	53	0044	11-8-3	11-8-3
= equal to	= equal to	= equal to	54	54	0075	8-3	8-6
space	space	space	55	55	0040	no punch	no punch
, comma	, comma	, comma	56	56	0054	0-8-3	0-8-3
. period	. period	. period	57	57	0056	12-8-3	12-8-3
equivalence	# number	# number	60	60	0043	0-8-6	8-3
[left bracket	[left bracket	[l. bracket	61	61	0133	8-7	12-8-2 or 12-0†††
] right bracket	] right bracket	] r. bracket	62	62	0135	0-8-2	11-8-2 or 11-0†††
% percent††	% percent††	% percent††	63	63	0045	8-6	0-8-4

TABLE A-2. LOCAL BATCH DEVICE CHARACTER SETS (Contd)

CDC Graphic (64-Character Set)	ASCII Graphic (64-Character Set)	ASCII Graphic (95-Character Set)	Octal 6-Bit Display Code	Octal 6/12-Bit Display Code[†]	Octal 12-Bit ASCII Code	Card Keypunch Code	
						026	029
≠ not equal	" quote	" quote	64	64	0042	8-4	8-7
↱ concat.	_ underline	_ underline	65	65	0137	0-8-5	0-8-5
∨ logical OR	! exclamation	! exclamation	66	66	0041	11-0 or 11-8-2[§]	12-8-7 or 11-0[§]
∧ logical AND	& ampersand	& ampersand	67	67	0046	0-8-7	12
↑ superscript	′ apostrophe	′ apostrophe	70	70	0047	11-8-5	8-5
↓ subscript	? question	? question	71	71	0077	11-8-6	0-8-7
< less than	< less than	< less than	72	72	0074	12-0 or 12-8-2[§]	12-8-4 or 12-0[§]
> greater than	> greater than	> greater than	73	73	0076	11-8-7	0-8-6
≤ less/equal	@ commercial at		74			8-5	8-4
≥ greater/equal	\ reverse slant	\ rev. slant	75	75	0134	12-8-5	0-8-2
¬ logical NOT	^ circumflex		76			12-8-6	11-8-7
; semicolon	; semicolon	; semicolon	77	77	0073	12-8-7	11-8-6
		@ coml. at		7401	0100		
		^ circumflex		7402	0136		
		: colon[††]		7404[††]	0072		
		` grave accent		7407	0140		
		a		7601	0141		
		b		7602	0142		
		c		7603	0143		
		d		7604	0144		
		e		7605	0145		
		f		7606	0146		
		g		7607	0147		
		h		7610	0150		
		i		7611	0151		
		j		7612	0152		
		k		7613	0153		
		l		7614	0154		
		m		7615	0155		
		n		7616	0156		
		o		7617	0157		
		p		7620	0160		
		q		7621	0161		
		r		7622	0162		
		s		7623	0163		
		t		7624	0164		
		u		7625	0165		
		v		7626	0166		
		w		7627	0167		
		x		7630	0170		
		y		7631	0171		
		z		7632	0172		
		{ left brace		7633	0173		
		| vert. line		7634	0174		
		} right brace		7635	0175		
		~ tilde		7636	0176		

[†] Generally available only on NOS, or through BASIC on NOS/BE.

[††] The interpretation of this character or code depends on its context. Refer to Character Set Anomalies in the text.

[†††] Available for input only, on NOS.

[§] Available for input only, on NOS/BE or SCOPE 2.

Local Batch Users

Table A-2 lists the CDC graphic 64-character set, the ASCII graphic 64-character set, and the ASCII graphic 95-character set. This table also lists the code sets and card keypunch codes (026 and 029) that represent the characters.

The 64-character sets use display code as their code set; the 95-character set uses 12-bit ASCII code. The 95-character set is composed of all the characters in the ASCII 128-character set that can be printed at a line printer (refer to Line Printer Output). Only 12-bit ASCII code files can be printed using the ASCII graphic 95-character set. To print a 6/12-bit display code file (usually created in IAF ASCII mode), you must convert the file to 12-bit ASCII code. To do this, the NOS FCOPY control statement must be issued. The 95-character set is represented by the 12-bit ASCII codes 0040_8 through 0176_8.

Line Printer Output

The batch character set printed depends on the print train used on the line printer to which the file is sent. The following are the print trains corresponding to each of the batch character sets:

Character Set	Print Train
CDC graphic 64-character set	596-1
ASCII graphic 64-character set	596-5
ASCII graphic 95-character set	596-6

The characters of the default 596-1 print train are listed in the table A-2 column labeled CDC Graphic (64-Character); the 596-5 print train characters are listed in the table A-2 column labeled ASCII Graphic (64-Character); and the 596-6 print train characters are listed in the table A-2 column labeled ASCII Graphic (95-Character).

If a transmission error occurs during the printing of a line, NOS prints the line again. The CDC graphic print train prints a concatenation symbol (↱) in the first printable column of a line containing errors. The ASCII print trains print an underline instead of the concatenation symbol.

If an unprintable character exists in a line (that is, a 12-bit ASCII code outside of the range 0040_8 through 0176_8), the number sign (#) appears in the first printable column of a print line and a space replaces the unprintable character.

Punched Card Input and Output

A character represented by multiple punches in a single column has its punch pattern identified by numbers and hyphens. For example, the punches representing an exclamation point are identified as 11-0; this notation means both rows 11 and 0 are punched in the same column. A multiple punch pattern that represents something other than a character is identified by numbers and slashes. For example, the punches representing the end of an input file are identified as 6/7/8/9; this notation means rows 6 through 9 are punched in the same column.

Under NOS, coded character data is exchanged with local batch card readers or card punches according to the translations shown in table A-2. As indicated in the table, additional card keypunch codes are available for input of the ASCII and CDC characters [and]. The 95-character set cannot be read or punched as coded character data.

Depending on an installation or deadstart option, NOS assumes an input deck has been punched either in 026 or 029 keypunch code (regardless of the character set in use). The alternate keypunch codes can be specified by a 26 or 29 punched in columns 79 and 80 of any job card, 6/7/9 card, or 7/8/9 card. The specified code translation remains in effect throughout the job unless the translation is reset by specification of the alternate code translation on a subsequent 6/7/9 card or 7/8/9 card.

NOS keypunch code translation can also be changed before or after a 7/8/9 card by a card containing a 5/7/9 punch in column 1. A space (no punch) in column 2 indicates 026 conversion mode; a 9 punch in column 2 indicates 029 conversion mode. The conversion change remains in effect until another change card is encountered or the job ends.

The 5/7/9 card also allows literal input when 4/5/6/7/8/9 is punched in column 2. Literal input can be used to read 80-column binary character data within a punched card deck of coded character data.

Literal cards are stored with each column in a 12-bit byte (a row 12 punch is represented by a 1 in bit 11, row 11 by bit 10, row 0 by bit 9, and rows 1 through 9 by bits 8 through 0 of the byte), 16 central memory words per card. Literal input cards are read until a card identical to the previous 5/7/9 card (4/5/6/7/8/9 in column 2) is read. The next card can specify a new conversion mode.

Under NOS/BE, coded character data is exchanged with local batch card readers or card punches according to the translations shown in table A-2. As indicated in the table, additional card keypunch codes are available for input of the CDC characters ∨ and < or their ASCII equivalents ! and <. The 95-character set cannot be read or punched as coded character data.

Depending on an installation option, NOS/BE assumes an input deck has been punched either in 026 or in 029 keypunch code (regardless of the character set in use). The alternate keypunch codes can be specified by a 26 or 29 punched in columns 79 and 80 of the job statement or in columns 79 and 80 of any 7/8/9 card. The specified code translation remains in effect throughout the job unless it is reset by specification of the alternate code translation on a subsequent 7/8/9 card.

Under NOS/BE, a card with all 12 rows of column 1 punched and all of one other column punched can be followed by 80-column cards of free-form binary data. These binary data cards are read or punched as described for NOS literal data until another card with 12 punches in column 1 and in one other column occurs, or until the job ends. The next card is interpreted as coded data.

Remote Batch Users

When card decks are read from remote batch devices, the ability to select alternate keypunch code translations depends upon the remote terminal equipment.

NOS Usage

Remote batch terminal line printer, punched card, and plotter character set support is described in the Remote Batch Facility (RBF) reference manual. RBF support of console input and output is restricted to character mode transmission. Character mode is described under Terminal Transmission Code Sets in this appendix.

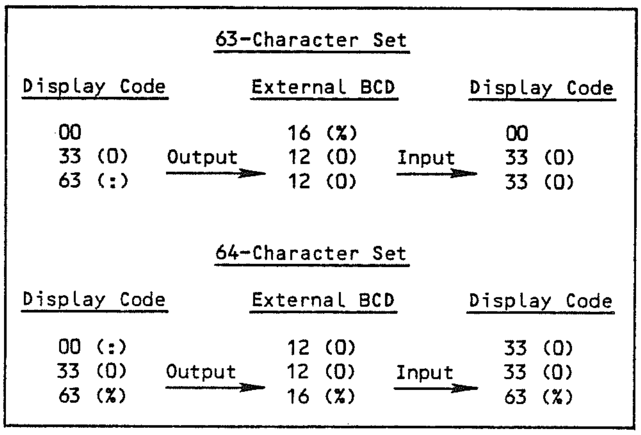
NOS/BE Usage

Remote batch terminal line printer, punched card, and plotter character set support is described in the INTERCOM reference manual.

Magnetic Tape Users

Coded character data to be copied from mass storage to magnetic tape is assumed to be represented in display code. NOS converts the data to external BCD code when writing a coded 7-track tape and to ASCII or EBCDIC code (as specified on the tape assignment statement) when writing a coded 9-track tape.

Because only 63 characters can be represented in 7-track even parity, one of the 64 display codes is lost in conversion to and from external BCD code. Figure A-1 shows the differences in conversion that depend on which character set (63 or 64) the system uses. The ASCII character for the specified character code is shown in parentheses. The output arrow shows how the display code changes when it is written on tape in external BCD. The input arrow shows how the external BCD code changes when the tape is read and converted to display code.



63-Character Set

Display Code		External BCD		Display Code
00		16 (%)		00
33 (0)	Output →	12 (0)	Input →	33 (0)
63 (:)		12 (0)		33 (0)

64-Character Set

Display Code		External BCD		Display Code
00 (:)		12 (0)		33 (0)
33 (0)	Output →	12 (0)	Input →	33 (0)
63 (%)		16 (%)		63 (%)

Figure A-1. Magnetic Tape Code Conversions

Tables A-3 and A-4 show the character set conversions for 9-track tapes. Table A-3 lists the conversions to and from 7-bit ASCII character code and 6-bit display code. Table A-4 lists the conversions between 8-bit EBCDIC character code and 6-bit display code. Table A-5 shows the character set conversions between 6-bit external BCD and 6-bit display code for 7-track tapes.

If a lowercase ASCII or EBCDIC code is read from a 9-track coded tape, it is converted to its uppercase 6-bit display code equivalent. To read and write lowercase ASCII or EBCDIC characters, the user must assign the tape in binary mode and then convert the binary character data.

During binary character data transfers to or from 9-track magnetic tape, the 7-bit ASCII codes shown in table A-6 are read or written unchanged; the 8-bit hexadecimal EBCDIC codes shown in table A-7 also can be read or written unchanged. ASCII and EBCDIC codes cannot be read or written to 7-track magnetic tape as binary character data.

Two CDC utility products, FORM and the 8-Bit Subroutines, can be used to convert to and from EBCDIC data. Table A-7 contains the octal values of each EBCDIC code right-justified in a 12-bit byte with zero fill. This 12-bit EBCDIC code can also be produced using FORM and the 8-Bit Subroutines.

TERMINAL TRANSMISSION CODE SETS

There are two modes in which coded character data can be exchanged with a network terminal console. These two modes, character mode and transparent mode, correspond to the type of character code editing and translation performed by the network software during input and output operations.

Under NOS, the terminal operator can select the network software input transmission mode by using a Terminal Interface Program command (sometimes referred to as a terminal definition command). The network software output transmission mode can be selected by the application program providing the terminal facility service.

Character Mode Transmissions

Character mode is the initial and default mode used for both input and output transmissions. When the network software services the terminal in character mode, it translates input characters from the transmission code used by the terminal into the ASCII code shown in table A-6. The translation of a specific transmission code to a specific ASCII code depends on the terminal class that the network software associates with the terminal. In character mode input, the parity of the terminal transmission code is not preserved in the corresponding ASCII code; the ASCII code received by the terminal-servicing facility program always has its eighth bit set to zero.

TABLE A-3. ASCII 9-TRACK CODED TAPE CONVERSION

ASCII				Display Code†††		ASCII				Display Code†††	
Code Conversion†		Character and Code Conversion††				Code Conversion†		Character and Code Conversion††			
Code (Hex)	Char	Code (Hex)	Char	ASCII Char	Code (Octal)	Code (Hex)	Char	Code (Hex)	Char	ASCII Char	Code (Octal)
20	space	00	NUL	space	55	40	@	60	`	@	74
21	!	7D	}	!	66	41	A	61	a	A	01
22	"	02	STX	"	64	42	B	62	b	B	02
23	#	03	ETX	#	60	43	C	63	c	C	03
24	$	04	EOT	$	53	44	D	64	d	D	04
25	%	05	ENQ	%	63	45	E	65	e	E	05
25	%	05	ENQ	space	55	46	F	66	f	F	06
26	&	06	ACK	&	67	47	G	67	g	G	07
27	'	07	BEL	'	70	48	H	68	h	H	10
28	(	08	BS	(	51	49	I	69	i	I	11
29	)	09	HT	)	52	4A	J	6A	j	J	12
2A	*	0A	LF	*	47	4B	K	6B	k	K	13
2B	+	0B	VT	+	45	4C	L	6C	l	L	14
2C	,	0C	FF	,	56	4D	M	6D	m	M	15
2D	-	0D	CR	-	46	4E	N	6E	n	N	16
2E	.	0E	SO	.	57	4F	O	6F	o	O	17
2F	/	0F	SI	/	50	50	P	70	p	P	20
30	0	10	DLE	0	33	51	Q	71	q	Q	21
31	1	11	DC1	1	34	52	R	72	r	R	22
32	2	12	DC2	2	35	53	S	73	s	S	23
33	3	13	DC3	3	36	54	T	74	t	T	24
34	4	14	DC4	4	37	55	U	75	u	U	25
35	5	15	NAK	5	40	56	V	76	v	V	26
36	6	16	SYN	6	41	57	W	77	w	W	27
37	7	17	ETB	7	42	58	X	78	x	X	30
38	8	18	CAN	8	43	59	Y	79	y	Y	31
39	9	19	EM	9	44	5A	Z	7A	z	Z	32
3A	:	1A	SUB	:	00	5B	[	1C	FS	[	61
Display code 00 is undefined at sites using the						5C	\	7C	|	\	75
63-character set.						5D	[	01	SOH	]	62
3A	:	1A	SUB	:	63	5E	^	7E	~	^	76
3B	;	1B	ESC	;	77	5F	_	7F	DEL	_	65
3C	<	7B	{	<	72						
3D	=	1D	GS	=	54						
3E	>	1E	RS	>	73						
3F	?	1F	US	?	71						

†When these characters are copied from or to a tape, the characters remain the same and the code changes from/to ASCII to/from display code.

††These characters do not exist in display code. When the characters are copied from a tape, each ASCII character is changed to an alternate display code character. The corresponding codes are also changed. Example: When the system copies a lowercase a, 61_{16}, from tape, it writes an uppercase A, 01_8.

†††A display code space always translates to an ASCII space.

TABLE A-4. EBCDIC 9-TRACK CODED TAPE CONVERSION

EBCDIC				Display Code†††		EBCDIC				Display Code†††	
Code Conversion†		Character and Code Conversion††				Code Conversion†		Character and Code Conversion††			
Code (Hex)	Char	Code (Hex)	Char	ASCII Char	Code (Octal)	Code (Hex)	Char	Code (Hex)	Char	ASCII Char	Code (Octal)
40	space	00	NUL	space	55	C6	F	86	f	F	06
4A	¢	1C	IFS	[	61	C7	G	87	g	G	07
4B	.	0E	SO	.	57	C8	H	88	h	H	10
4C	<	C0	{	<	72	C9	I	89	i	I	11
4D	(	16	BS	(	51	D1	J	91	j	J	12
4E	+	0B	VT	+	45	D2	K	92	k	K	13
4F	|	D0	}	!	66	D3	L	93	l	L	14
50	&	2E	ACK	&	67	D4	M	94	m	M	15
5A	!	01	SOH	]	62	D5	N	95	n	N	16
5B	$	37	EOT	$	53	D6	O	96	o	O	17
5C	*	25	LF	*	47	D7	P	97	p	P	20
5D	)	05	HT	)	52	D8	Q	98	q	Q	21
5E	;	27	ESC	;	77	D9	R	99	r	R	22
5F	–	A1	~	/	76	E0	\	6A	|	\	75
60	–	0D	CR	–	46	E2	S	A2	s	S	23
61	′	0F	SI	′	50	E3	T	A3	t	T	24
6B	,	0C	FF	,	56	E4	U	A4	u	U	25
6C	%	2D	ENQ	%	63	E5	V	A5	v	V	26
6C	%	2D	ENQ	space	55	E6	W	A6	w	W	27
6D	_	07	DEL	_	65	E7	X	A7	x	X	30
6E	>	1E	IRS	>	73	E8	Y	A8	y	Y	31
6F	?	1F	IUS	?	71	E9	Z	A9	z	Z	32
7A	:	3F	SUB	:	00	F0	0	10	DLE	0	33
Display code 00 is undefined at sites using the 63-character set.						F1	1	11	DC1	1	34
						F2	2	12	DC2	2	35
7A	:	3F	SUB	:	63	F3	3	13	TM	3	36
7B	#	03	ETX	#	60	F4	4	3C	DC4	4	37
7C	@	79	\	@	74	F5	5	3D	NAK	5	40
7D	′	2F	BEL	′	70	F6	6	32	SYN	6	41
7E	=	1D	IGS	=	54	F7	7	26	ETB	7	42
7F	"	02	STX	"	64	F8	8	18	CAN	8	43
C1	A	81	a	A	01	F9	9	19	EM	9	44
C2	B	82	b	B	02						
C3	C	83	c	C	03						
C4	D	84	d	D	04						
C5	E	85	e	E	05						

†All EBCDIC codes not listed translate to display code 55_8 (space). A display code space always translates to an EBCDIC space.

††These characters do not exist in display code. When the characters are copied from a tape, each EBCDIC character is changed to an alternate display code character. The corresponding codes are also changed. Example: When the system copies a lowercase a, 81_{16}, from tape, it writes an uppercase A, 01_8.

†††When these characters are copied from or to a tape, the characters remain the same (except EBCDIC codes $4A_{16}$, $4F_{16}$, $5A_{16}$, and $5F_{16}$) and the code changes from/to EBCDIC to/from display code.

TABLE A-5. 7-TRACK CODED TAPE CONVERSIONS

External BCD	ASCII Character	Octal Display Code	External BCD	ASCII Character	Octal Display Code
01	1	34	40	-	46
02	2	35	41	J	12
03	3	36	42	K	13
04	4	37	43	L	14
05	5	40	44	M	15
06	6	41	45	N	16
07	7	42	46	O	17
10	8	43	47	P	20
11	9	44	50	Q	21
12†	0	33	51	R	22
13	=	54	52	!	66
14	"	64	53	$	53
15	@	74	54	*	47
16	%	63	55	'	70
17	[	61	56	?	71
20	space	55	57	>	73
21	/	50	60	+	45
22	S	23	61	A	01
23	T	24	62	B	02
24	U	25	63	C	03
25	V	26	64	D	04
26	W	27	65	E	05
27	X	30	66	F	06
30	Y	31	67	G	07
31	Z	32	70	H	10
32	]	62	71	I	11
33	,	56	72	<	72
34	(	51	73	.	57
35	_	65	74	)	52
36	#	60	75	\	75
37	&	67	76	^	76
			77	;	77

†As explained in the text of this appendix, conversion of these codes depends on whether the tape is being read or written.

Character mode output is translated in a similar manner. The network software provides the parity bit setting appropriate for the terminal being serviced, even though translating from ASCII characters with zero parity bit settings.

The general case for code translations of character mode data is summarized in the following paragraphs. This generalized description permits use of only table A-6 to explain all specific cases. The reader can logically extend this generalized description to allow use of tables A-1 through A-5 as descriptions of character set mapping for various functions initiated from a terminal. Tables A-1 through A-5 are provided for the reader's use while coding an application program to run under the operating system. They do not describe character transmissions between an application program and the network.

Table A-6 contains the ASCII 128-character set supported by the Network Access Method. A 96-character subset consists of the rightmost six columns and includes the 95-character graphic subset referenced previously in this appendix; the deletion character (DEL) is not a graphic character. A 64-character subset consists of the middle four columns. Note that 6-bit display code equivalents exist for the characters in this 64-character subset only.

Although the network supports the 128-character set, some terminals restrict output to a smaller subset. This restriction is supported by replacing the control characters in columns 0 and 1 of table A-6 with blanks to produce the 96-character subset, and, additionally, replacing the characters in columns 6 and 7 with the corresponding characters from columns 4 and 5, respectively, to produce the 64-character subset.

Similarly, input from a device may be limited to a smaller subset by the device itself because the device cannot produce the full 128-character set. A character input from a device using a character set other than ASCII is converted to an equivalent ASCII character; characters without ASCII character equivalents are replaced by the ASCII space.

An application can also cause character replacement (as described previously for output) as well as character conversion, by requesting display-coded input from the network.

TABLE A-6. FULL ASCII CHARACTER SET

← 128-Character Set →

← 96-Character Subset →

← 64-Character Subset →

b7 →					0	0	0	0	1	1	1	1
b6 →					0	0	1	1	0	0	1	1
b5 →					0	1	0	1	0	1	0	1
Bits b4 ↓	b3 ↓	b2 ↓	b1 ↓	Row ↓ \ Column →	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL 000	DLE 020	SP 040	0 060	@ 100	P 120	` 140	p 160
0	0	0	1	1	SOH 001	DC1 021	! 041	1 061	A 101	Q 121	a 141	q 161
0	0	1	0	2	STX 002	DC2 022	" 042	2 062	B 102	R 122	b 142	r 162
0	0	1	1	3	ETX 003	DC3 023	# 043	3 063	C 103	S 123	c 143	s 163
0	1	0	0	4	EOT 004	DC4 024	$ 044	4 064	D 104	T 124	d 144	t 164
0	1	0	1	5	ENQ 005	NAK 025	% 045	5 065	E 105	U 125	e 145	u 165
0	1	1	0	6	ACK 006	SYN 026	& 046	6 066	F 106	V 126	f 146	v 166
0	1	1	1	7	BEL 007	ETB 027	' 047	7 067	G 107	W 127	g 147	w 167
1	0	0	0	8	BS 010	CAN 030	(050	8 070	H 110	X 130	h 150	x 170
1	0	0	1	9	HT 011	EM 031	) 051	9 071	I 111	Y 131	i 151	y 171
1	0	1	0	A	LF 012	SUB 032	* 052	: 072	J 112	Z 132	j 152	z 172
1	0	1	1	B	VT 013	ESC 033	+ 053	; 073	K 113	[133	k 153	{ 173
1	1	0	0	C	FF 014	FS 034	, 054	< 074	L 114	\ 134	l 154	174
1	1	0	1	D	CR 015	GS 035	- 055	= 075	M 115	] 135	m 155	} 175
1	1	1	0	E	SO 016	RS 036	. 056	> 076	N 116	^ 136	n 156	~ 176
1	1	1	1	F	SI 017	US 037	/ 057	? 077	O 117	_ 137	o 157	DEL 177

LEGEND:

Numbers under characters are the octal values for the 7-bit character codes used within the network.

TABLE A-7. FULL EBCDIC CHARACTER SET

Hexa-decimal EBCDIC Code	Octal 12-Bit EBCDIC Code	EBCDIC Graphic Character	EBCDIC Control Character†	Hexa-decimal EBCDIC Code	Octal 12-Bit EBCDIC Code	EBCDIC Graphic Character†	EBCDIC Control Character
00	0000		NUL	41	0101	undefined	undefined
01	0001		SOH	thru	thru		
02	0002		STX	49	0111		
03	0003		ETX	4A	0112	¢	
04	0004		PF	4B	0113	.	
05	0005		HT	4C	0114	<	
06	0006		LC	4D	0115	(	
07	0007		DEL	4E	0116	+	
08	0010	undefined	undefined	4F	0117	|	
09	0011	undefined	undefined	50	0120	&	
0A	0012		SMM	51	0121	undefined	undefined
0B	0013		VT	thru	thru		
0C	0014		FF	59	0131		
0D	0015		CR	5A	0132	!	
0E	0016		SO	5B	0133	$	
0F	0017		SI	5C	0134	*	
10	0020		DLE	5D	0135	)	
11	0021		DC1	5E	0136	;	
12	0022		DC2	5F	0137	¬	
13	0023		TM	60	0140	-	
14	0024		RES	61	0141	/	
15	0025		NL	62	0142	undefined	undefined
16	0026		BS	thru	thru		
17	0027		IL	69	0151		
18	0030		CAN	6A	0152	¦	
19	0031		EM	6B	0153	,	
1A	0032		CC	6C	0154	%	
1B	0033		CU1	6D	0155	_	
1C	0034		IFS	6E	0156	>	
1D	0035		IGS	6F	0157	?	
1E	0036		IRS	70	0160	undefined	undefined
1F	0037		IUS	thru	thru		
20	0040		DS	78	0170		
21	0041		SOS	79	0171	`	
22	0042		FS	7A	0172	:	
23	0043	undefined	undefined	7B	0173	#	
24	0044		BYP	7C	0174	@	
25	0045		LF	7D	0175	'	
26	0046		ETB or EOB	7E	0176	=	
27	0047		ESC or PRE	7F	0177	"	
28	0050	undefined	undefined	80	0200	undefined	undefined
29	0051	undefined	undefined	81	0201	a	
2A	0052		SM	82	0202	b	
2B	0053		CU2	83	0203	c	
2C	0054	undefined	undefined	84	0204	d	
2D	0055		ENQ	85	0205	e	
2E	0056		ACK	86	0206	f	
2F	0057		BEL	87	0207	g	
30	0060	undefined	undefined	88	0210	h	
31	0061	undefined	undefined	89	0211	i	
32	0062		SYN	8A	0212	undefined	undefined
33	0063	undefined	undefined	thru	thru		
34	0064		PN	90	0220		
35	0065		RS	91	0221	j	
36	0066		UC	92	0222	k	
37	0067		EOT	93	0223	l	
38	0070	undefined	undefined	94	0224	m	
39	0071	undefined	undefined	95	0225	n	
3A	0072	undefined	undefined	96	0226	o	
3B	0073		CU3	97	0227	p	
3C	0074		DC4	98	0230	q	
3D	0075		NAK	99	0231	r	
3E	0076	undefined	undefined	9A	0232	undefined	undefined
3F	0077		SUB	thru	thru		
40	0100	space		A0	0240		

TABLE A-7. FULL EBCDIC CHARACTER SET (Contd)

Hexa-decimal EBCDIC Code	Octal 12-Bit EBCDIC Code	EBCDIC Graphic Character†	EBCDIC Control Character	Hexa-decimal EBCDIC Code	Octal 12-Bit EBCDIC Code	EBCDIC Graphic Character†	EBCDIC Control Character
A1	0241	~		D7	0327	P	
A2	0242	s		D8	0330	Q	
A3	0243	t		D9	0331	R	
A4	0244	u		DA	0332	undefined	undefined
A5	0245	v		thru	thru		
A6	0246	w		DF	0337		
A7	0247	x		E0	0340	\	
A8	0250	y		E1	0341	undefined	undefined
A9	0251	z		E2	0342	S	
AA	0252	undefined	undefined	E3	0343	T	
thru	thru			E4	0344	U	
BF	0277			E5	0345	V	
C0	0300	{		E6	0346	W	
C1	0301	A		E7	0347	X	
C2	0302	B		E8	0350	Y	
C3	0303	C		E9	0351	Z	
C4	0304	D		EA	0352	undefined	undefined
C5	0305	E		EB	0353	undefined	undefined
C6	0306	F		EC	0354		
C7	0307	G		ED	0355	undefined	undefined
C8	0310	H		thru	thru		
C9	0311	I		EF	0357		
CA	0312	undefined	undefined	F0	0360	0	
CB	0313	undefined	undefined	F1	0361	1	
CC	0314	⑀		F2	0362	2	
CD	0315	undefined	undefined	F3	0363	3	
CE	0316	⑂		F4	0364	4	
CF	0317	undefined	undefined	F5	0365	5	
D0	0320	}		F6	0366	6	
D1	0321	J		F7	0367	7	
D2	0322	K		F8	0370	8	
D3	0323	L		F9	0371	9	
D4	0324	M		FA	0372	|	
D5	0325	N		FB	0373	undefined	undefined
D6	0326	O		thru	thru		
				FF	0377		

†Graphic characters shown are those used on the IBM System/370 standard (PN) print train. Other devices support subsets or variations of this character graphic set.

The 7-bit hexadecimal code value for each character consists of the character's column number in the table, followed by its row number. For example, N is in row E of column 4, so its value is $4E_{16}$.

Transparent Mode Transmissions

Transparent mode is selected separately for input and output transmissions. During transparent mode input, the parity bit is stripped from each terminal transmission code (unless the N parity option has been selected by a Terminal Interface Program command), and the transmission code is placed in an 8-bit byte without translation to 7-bit ASCII code. Line transmission protocol characters are deleted from a mode 4C terminal input stream.

When the 8-bit bytes arrive in the host computer, a terminal servicing facility program such as the Interactive Facility can right-justify the bytes within a 12-bit byte. Upon transmission of 12-bit bytes from the host computer, the leftmost 4 bits (bits 11 through 8) are discarded.

During transparent mode output, processing similar to that performed for input occurs. The code in each 8-bit byte received by the network software from the terminal servicing facility program is not translated. The parity bit appropriate for the terminal class being used is altered as indicated by the parity option in effect for the terminal. The codes are then output in transmission bytes appropriate for the codes associated with the terminal class being used. Line transmission protocol characters are inserted into a mode 4C terminal output stream.

PRODUCT-DEPENDENT CHARACTER USAGE

The character set collating sequences (tables A-8, A-9, and A-10) are utilized by the XCOMP subroutine for data comparison. In the ASCII and EBCDIC collating sequences, ascending binary code values correspond to ascending collating values. Numbers collate low using the ASCII character subset sequence. Numbers collate high using the EBCDIC character subset collating sequence. The CDC collating sequence varies through installation option.

TABLE A-8. CDC CHARACTER SET COLLATING SEQUENCE

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

†In installations using the 63-graphic set, the % graphic does not exist. The : graphic is display code 63, external BCD code 16.

TABLE A-9. ASCII CHARACTER SUBSET COLLATING SEQUENCE

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

[†] In installations using a 63-graphic set, the % graphic does not exist. The : graphic is display code 63.

TABLE A-10. EBCDIC 64-CHARACTER SUBSET COLLATING SEQUENCE

Collating Sequence		EBCDIC (64-character Subset)	6-bit Display Code	EBCDIC Hexa-decimal Code	Collating Sequence		EBCDIC (64-character Subset)	6-bit Display Code	EBCDIC Hexa-decimal Code
Decimal	Octal				Decimal	Octal			
00	00	blank	55	40	32	40	G	07	C7
01	01	.	57	4B	33	41	H	10	C8
02	02	<	72	4C	34	42	I	11	C9
03	03	(	51	4D	35	43	!	62	5A
04	04	+	45	4E	36	44	J	12	D1
05	05	|	66	4F	37	45	K	13	D2
06	06	&	67	50	38	46	L	14	D3
07	07	$	53	5B	39	47	M	15	D4
08	10	*	47	5C	40	50	N	16	D5
09	11	)	52	5D	41	51	O	17	D6
10	12	;	77	5E	42	52	P	20	D7
11	13	¬	76	5F	43	53	Q	21	D8
12	14	-	46	60	44	54	R	22	D9
13	15	/	50	61	45	55	none	75	E0
14	16	,	56	6B	46	56	S	23	E2
15	17	%	63	6C	47	57	T	24	E3
16	20	_	65	6D	48	60	U	25	E4
17	21	>	73	6E	49	61	V	26	E5
18	22	?	71	6F	50	62	W	27	E6
19	23	:	00	7A	51	63	X	30	E7
20	24	#	60	7B	52	64	Y	31	E8
21	25	@	74	7C	53	65	Z	32	E9
22	26	'	70	7D	54	66	0	33	F0
23	27	=	54	7E	55	67	1	34	F1
24	30	"	64	7F	56	70	2	35	F2
25	31	¢	61	4A	57	71	3	36	F3
26	32	A	01	C1	58	72	4	37	F4
27	33	B	02	C2	59	73	5	40	F5
28	34	C	03	C3	60	74	6	41	F6
29	35	D	04	C4	61	75	7	42	F7
30	36	E	05	C5	62	76	8	43	F8
31	37	F	06	C6	63	77	9	44	F9

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 occurs in a FORTRAN Extended 4 or COBOL program, appropriate traceback information is attempted. The general format of a diagnostic message is shown in figure B-1.

A typical set of error messages occuring in a FORTRAN Extended 4 program, including appropriate traceback information giving the calling routine name and the source program line number, is shown in figure B-2.

Dayfile diagnostic messages generated by the 8-bit subroutines are given in table B-1. Message significance and appropriate user action are also shown.

```
ERROR DETECTED BY xname - cause - message

xname      Name of the 8-bit subroutine that
           detected the error

cause      Probable cause of the error

message    Diagnostic message as listed in this
           appendix
```

Figure B-1. Diagnostic Message Format

```
ERROR DETECTED BY XFILE    -   PARAMETER-FILE NOT DECLARED
    CALLED FROM BJC8SB         AT LINE       7

ERROR DETECTED BY XWRITE   -   PARAMETER-FILE NOT SPECIFIED AS WRITE MODE
    CALLED FROM BJC8SB         AT LINE      12

ERROR DETECTED BY XMOVE    -   PARAMETER-UNRECOGNIZED MOVE-COMPARE TYPE
    CALLED FROM BJC8SB         AT LINE      38

ERROR DETECTED BY XWRITE   -   I-O - UNRECOVERABLE ERROR ON WRITE FILE
    CALLED FROM BJC8SB         AT LINE      33
```

Figure B-2. Sample Error Messages Occurring in a FORTRAN Extended 4 Program

TABLE B-1. DAYFILE DIAGNOSTIC MESSAGES

Messages	Significance	Action	Issued By
BAD OFFSET IN PARAMETER DESCRIPTOR	An illegal bit or byte position was specified.	Specify the correct bit or byte position.	XREAD, XWRITE
BAD SYNTAX IN Z, S, N, OR P FIELD	The data being converted is in illegal format.	Correct data format and and rerun.	XREAD, XWRITE
BIT SPECIFICATION ILLEGAL FOR NON-BIT FIELD	The w in i/w item locator format in a selector expression can be specified only if the T field is B.	Change position specification to i form to indicate character position.	XREAD, XWRITE
BLKSIZE EXCEEDS 32760 BYTES	The BLKSIZE parameter in the XFILE call is too big.	Reduce the value of BLKSIZE.	XREAD, XWRITE
BLKSIZE NOT SPECIFIED	The BLKSIZE parameter in the XFILE call has been omitted.	Specify BLKSIZE parameter.	XFILE
BLOCK SHORTER THAN V-HEADER	The input IBM record is in the wrong format. Block descriptor word contents or record descriptor word contents do not agree with actual block/record size.	Check for tape error or incorrect RECFM specification.	XREAD

TABLE B-1. DAYFILE DIAGNOSTIC MESSAGES (Contd)

Messages	Significance	Action	Issued By
CONVERSION STRINGS NESTED TOO DEEPLY	Conversion strings can be nested only to a depth of seven levels.	Reduce the number of levels.	XREAD, XWRITE
DOUBLY SPECIFIED PARAMETER IN FILE STRING	A duplicate parameter was specified in the file string parameter of the XFILE call.	Delete or correct the parameter.	XFILE
EMPTY BLOCK IN VS-RECORD	No data is present in the input block.	Rewrite the file.	XREAD
FILE NOT DECLARED	The wrong file number was specified in the XFILE call or the file declaration is missing on a FORTRAN program statement.	Specify the correct file.	XFILE
FILE NOT SPECIFIED AS READ MODE	The USE parameter in the XFILE call has not been specified as R.	Specify USE=R in the XFILE call.	XREAD
FILE NOT SPECIFIED AS WRITE MODE	The USE parameter in the XFILE call has not been specified as W.	Specify USE=W in the XFILE call.	XWRITE
FILE PARAMETER IS NOT A FILE NAME	The file name is misspelled in an XFILE call or does not appear in the COBOL SELECT clause.	Check the file name.	XFILE
FILE STRING DOES NOT BEGIN WITH -(-	The file string parameter in the XFILE call must always begin with a left parenthesis.	Insert a left parenthesis.	XFILE
FILE STRING DOES NOT TERMINATE WITH -)-	The file string parameter in the XFILE call must always terminate with a right parenthesis.	Insert a right parenthesis.	XFILE
FILE TYPE NOT SPECIFIED	The FT specification in the file string parameter in an XFILE call has been omitted.	Specify file type.	XFILE
FILE USAGE NOT SPECIFIED	The USE specification in the file string parameter in an XFILE call has been omitted.	Specify file usage.	XFILE
FIRST CHARACTER OF CONVERSION STRING IS NOT -(-	Conversion strings must always begin with a left parenthesis.	Insert left parenthesis.	XREAD, XWRITE
FIRST ITEM IN SELECTOR - EXPRESSION NOT RECOGNIZED	The iTm field in the selector expression is specified incorrectly.	Correct the iTm field. Eliminate the m specification if the data type cannot be variable in length.	XREAD, XWRITE
ILLEGAL FIRST ITEM TYPE	The T in the Tml field is specified incorrectly.	Correct the type specification.	XREAD, XWRITE
ILLEGAL LENGTH PARAMETER DESCRIPTION	Length parameters in COBOL must be declared as COMPUTATIONAL-1.	Correct the length parameter.	XCOMP, XMOVE, XFILE
ILLEGAL SECOND ITEM TYPE	The T in the Tm2 field is specified incorrectly.	Correct the type specification.	XREAD, XWRITE
INCOMPLETE VS-RECORD AT END-OF-DATA	Data is missing from the input file. The length specified in the header information of the last record does not agree with actual record length.	Rewrite the file.	XREAD

TABLE B-1. DAYFILE DIAGNOSTIC MESSAGES (Contd)

Messages	Significance	Action	Issued By
INCONSISTENT PARAMETERS IN FILE STRING	The file string parameters are inconsistent in XFILE call.	Correct the file string parameters.	XFILE
INDEFINITE SOURCE VALUE NOT REPRESENTABLE	A floating-point source item has indefinite value.	Check the source record for bad data.	XWRITE
INDEFINITE VALUE FOR INTEGER DESTINATION FIELD	An item to be stored in integer destination field has indefinite value.	Check the source record for bad data.	XWRITE
INFINITE SOURCE VALUE NOT REPRESENTABLE	A floating-point source item has infinite value.	Check the source record for bad data.	XWRITE
INFINITE VALUE FOR INTEGER DESTINATION FIELD	An item to be stored in integer destination field has infinite value.	Check the source record for bad data.	XWRITE
INTEGER VALUE TOO LARGE FOR FIELD	The receiving field does not contain enough characters to represent all digits in the source field.	Increase the length specification of the receiving field descriptor.	XREAD, XWRITE
INVALID DATA TYPE	Legal data types are A for ASCII, C for EBCDIC, and X for display code.	Change T to the proper data type.	XREAD, XWRITE
INVALID PARAMETER VALUE IN FILE STRING	The parameter value in the file string of an XFILE call is incorrect.	Correct the parameter value in the file string. Check the punctuation.	XFILE
KEYWORD NOT FOLLOWED BY = IN FILE STRING	An equals sign (=) must follow KEYWORD in file-string.	Insert an = sign before the keyword in the file string.	XFILE
LITERAL STRING IS TOO LONG	Literal strings in a selector expression are limited to 80 characters.	Correct the literal string.	XREAD, XWRITE
LRECL NOT SPECIFIED	The necessary LRECL specification in a file-string parameter of an XFILE call was omitted.	Specify the LRECL parameter.	XFILE
LRECL TOO SMALL FOR V-RECORD HEADER	The actual record length does not agree with length specified in record descriptor word (RDW).	Rewrite the file.	XREAD
LRECL TOO LARGE FOR BLKSIZE	If the blocking type is not spanned, LRECL must be less than BLKSIZE.	Correct the BLKSIZE or the LRECL parameter.	XFILE
M SPECIFICATION ILLEGAL FOR DATA TYPE	The m in Tm1 or Tm2 is incorrectly specified. If the data cannot be variable in length, m must be omitted.	Eliminate the m specification.	XREAD, XWRITE
MISSING LENGTH PARAMETER	The length parameter is missing from a FORTRAN or COMPASS call to XPACK, XPAND, XMOVE, OR XCOMP.	Specify the length parameter.	XPACK, XPAND, XMOVE, XCOMP
MISSING PARAMETER LIST	The subroutine called requires a parameter list.	Specify the parameter list.	Any 8-bit subroutine
MISSING RELATIONAL OPERATOR IN SELECTOR-EXPRESSION	A relationship must be stated between two items in a selector-expression.	Specify the correct format for a selector expression.	XREAD, XWRITE

TABLE B-1. DAYFILE DIAGNOSTIC MESSAGES (Contd)

Messages	Significance	Action	Issued By
MISSING RIGHT PARENTHESIS	A string parameter must always end with a right parenthesis.	Insert right parenthesis.	XFILE, XREAD, XWRITE
MISSING RIGHT PARENTHESIS OR SEMICOLON	Punctuation is missing in a conversion string.	Insert the necessary punctuation.	XREAD, XWRITE
MISSING RIGHT STRING DELIMITER	The literal string terminator * or $ is missing.	Insert the required string delimiter.	XREAD, XWRITE
MISSING SEPARATOR AFTER CONVERSION ITEM	A conversion item must be followed by a comma, semicolon, or right parenthesis, depending upon circumstances.	Insert the necessary separator.	XREAD, XWRITE
MISSING SOURCE-1 PARAMETER	The calling sequence parameter list is incomplete.	Correct the parameter list.	XCOMP, XMOVE
MISSING SOURCE-2 OR DESTINATION	The parameter list in calling sequence is incomplete.	Correct the parameter list.	XCOMP, XMOVE
MISSING LEFT STRING DELIMITER	A literal string is missing the * or $ delimiter.	Insert the required string delimiter.	XREAD, XWRITE
MORE DATA AFTER RECORD IN V-UNBLOCKED FILE	The actual record length in input file exceeds that specified in record descriptor word (RDW).	Correct the record length.	XREAD
MORE DATA AFTER VS-RECORD SEGMENT	An actual segment record length in the input file exceeds that specified in the segment descriptor word (SDW).	A bad file copy might have occurred. Recopy the file.	XREAD
NO FILE STRING GIVEN	The file-string parameter is missing in an XFILE call.	Specify the file-string parameter.	XFILE
NO PARAMETERS	The necessary parameters were not specified.	Specify the required parameters.	Any 8-bit subroutine
NO PARAMETERS SUPPLIED TO SUBROUTINE	The called subroutine requires parameters.	Specify the parameters.	Any 8-bit subroutine
NO STATUS PARAMETER	The status parameter is required.	Specify the status parameter.	XCOMP
NO WORKING STORAGE AREA PROVIDED	A workspace parameter in XFILE calling sequence is required.	Specify the working storage area.	XFILE
NUMERIC LITERAL EXPONENT .GE. 512	The exponent value cannot exceed 511.	Decrease the exponent value and rerun.	XREAD, XWRITE
NUMERIC LITERAL OUT OF RANGE (INFINITE)	The numeric literal has infinite value.	Correct the literal.	XWRITE
PARAMETER IS NOT A DATA ITEM	A literal was supplied for a data-item in COBOL.	Replace the literal with a variable or array name.	Any 8-bit subroutine
RECFM NOT SPECIFIED	The RECFM parameter was omitted from the file-string of an XFILE calling sequence.	Specify the RECFM parameter.	XFILE

TABLE B-1. DAYFILE DIAGNOSTIC MESSAGES (Contd)

Messages	Significance	Action	Issued By
RELATIONAL OPERATOR NOT RECOGNIZED	An illegal relationship mnemonic has been specified in a selector expression. Legal mnemonics are LE, LT, EQ, NE, GT, and GE.	Specify a legal relational mnemonic.	XREAD, XWRITE
SECOND SELECTOR-EXPRESSION ITEM NOT RECOGNIZED	The second iTm field in a selector expression is in illegal format.	Correct the format of iTm.	XREAD, XWRITE
SELECTOR-EXPRESSION NOT TERMINATED BY COLON	The selector-expression must be terminated by a colon.	Insert the required colon.	XREAD, XWRITE
SIZE PARAMETER NOT NUMERIC TYPE	In a COBOL calling sequence to XFILE, the size parameter must be described as numeric.	Correct the size parameter.	XFILE
SOURCE CHARACTER NOT 0 OR 1, TO BIT STRING	In a character-to-bit conversion item, the source character can be only 0 or 1.	Specify a source character of 0 or 1.	XREAD, XWRITE
SOURCE EXPONENT TOO LARGE, NOT REPRESENTABLE	The exponent of the floating point number in the source field is too large, and the conversion cannot be performed.	Reduce the exponent value.	XREAD, XWRITE
STATUS RETURN NOT COMP-2	In a COBOL calling sequence, the status parameter must be declared as a COMPUTATIONAL-2 item.	Correct the status parameter.	Any 8-bit subroutine
STRING NOT IN NUMERIC SYNTACTIC FORM	The character string is not in the correct numeric literal format.	Correct the format.	XREAD, XWRITE
SYNTAX...NO DIGIT AFTER -E- IN NUMERIC LITERAL	Numeric literals in value fields of selector expressions must fit the numeric literal definition.	Correct the format of the numeric literal.	XREAD, XWRITE
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.	Reduce length specification of locator field descriptor.	XREAD, XWRITE
TEST FIELD NOT IN RECORD, ON LEFT	The locator field specified in a selector expression references a character preceding the first character in the logical record.	Decrease magnitude of l in -iTM.	XREAD, XWRITE
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.	Reduce the value of l in iTm.	XREAD, XWRITE
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.	Increase the length specification of the destination field descriptor.	XREAD, XWRITE
TOO MANY PARAMETERS	Extraneous parameters appear in the calling sequence.	Remove the unncessary parameters.	Any 8-bit subroutine
UNRECOGNIZED CODE SET SPECIFIED	The legal code sets are ASCII (A), EBCDIC (C), or display code (X).	Specify the correct code set.	XMOVE, XCOMP

TABLE B-1. DAYFILE DIAGNOSTIC MESSAGES (Contd)

Messages	Significance	Action	Issued By
UNRECOGNIZED KEYWORD IN FILE STRING	A file string parameter in an XFILE call is misspelled.	Correct the file string parameter.	XFILE
UNRECOGNIZED MOVE-COMPARE TYPE	The xy parameter is incorrectly specified.	Correct the xy parameter.	XMOVE, XCOMP
UNRECOVERABLE ERROR ON WRITE FILE	A parity error has occurred.	Follow site-defined procedure for reporting software failure or operational problems.	XWRITE
V-BLOCK HAS SHORT RECORD FRAGMENT	The actual record size is less than that specified in the record descriptor word (RDW).	Rewrite the file, a file error has occurred.	XREAD
V-BLOCK LENGTH EXCEEDS BLOCK SIZE	The actual record size exceeds the specified block size.	Specify larger block size.	XREAD
V-RECORD LENGTH LESS THAN 4 BYTES	Variable records must contain 4 bytes for the record descriptor word (RDW).	Rewrite the tape.	XREAD
VALUE TOO LARGE FOR FIELD WIDTH	A numeric value contains too many digits and/or symbols for the receiving field.	Increase the length specification of the receiving field descriptor.	XREAD, XWRITE
VS-RECORD FINAL SEGMENT MISSING	The final segment of a variable spanned logical record is missing from the input file.	Rewrite the tape.	XREAD
VS-RECORD FOUND IN TYPE V FILE	A segment descriptor word (SDW) was found in a file that was not spanned.	Rewrite the tape.	XREAD
VS-RECORD INITIAL SEGMENT MISSING	The first segment of a spanned logical record is missing from the input file.	Rewrite the tape.	XREAD
WORKING STORAGE AREA TOO SMALL	The size of the workspace buffer specified in the XFILE call is too small.	Correct the workspace buffer size.	XFILE
WSA NOT ALIGNED ON WORD BOUNDARY	The workspace parameter in a COBOL calling sequence to XFILE must have a beginning character position of 0.	Declare the working storage area as level of the item or use the SYNCHRONIZE clause for proper alignment.	XFILE

GLOSSARY C

ASCII -
The American Standard Code for Information Interchange, used under NOS as the ASCII 128-character set with either 6- or 12-bit characters and under NOS and NOS/BE as the ASCII 95-character set with 8-bit character codes contained in 12-bit bytes.

ASCII Graphic 63-Character Set -
A subset of the ASCII 128-character graphic and control set. The % character and related card code do not exist.

ASCII Graphic 64-Character Set -
A subset of the ASCII 128-character graphic and control set.

ASCII Graphic 95-Character Set -
Consists of all the characters in the ASCII 128-character set that can be printed at a CDC line printer. Only 12-bit code ASCII files can be printed using the ASCII graphic 95-character set. The 95-character set is represented by the 12-bit codes 0040_8 through 0176_8.

ASCII Graphic 128-Character Set -
Consists of all letters (uppercase and lowercase), digits, special symbols, and device control characters.

ASCII 8-Bit Code -
Eight bits stored in an 8-bit byte. IBM data is ASCII 8/8.

ASCII 12-Bit Code -
The ASCII 7-bit code (as defined by ANSI Standard X3.4-1977) right-justified in a 12-bit byte. Assuming that the bits are numbered from the right, bits 0 through 6 contain the ASCII code, bits 7 through 10 contain zeros, and bit 11 distinguishes the 12-bit ASCII 0000_8 code from the end-of-line byte. The 12-bit codes are 0001_8 through 0177_8 and 4000_8.

Basic Access Methods (BAM) -
A file manager that processes sequential and word addressable file organizations. (See CYBER Record Manager.)

Beginning-of-Information (BOI) -
The start of the first user information in a file.

Block -
The term block has several meanings depending on context. On tape, a block is information between interrecord gaps on tape. CYBER Record Manager defines blocks depending on organization. Valid block types are I, C, K, and E.

Boundary -
A file boundary is a physical indication that marks a logical division within a sequential file. The start of the first user record and individual user records is always recognized; other boundaries are affected by the record and blocking type and the file storage device. A word boundary is the first character position in a central memory word.

Byte -
A group of bits. When used for encoding character data, a byte represents a single character.

Character -
A letter, digit, punctuation mark, or mathematical symbol forming part of one or more of the standard character sets.

Character Set -
A set of graphic and/or control characters that is specified at the time the operating system is installed. (See Code Set.)

Checksum -
A value used to verify that the content of a record (excluding prefix tables) was copied correctly.

Code Set -
A set of codes used to represent each character within a character set. (See Character Set.)

Collating Sequence -
The sequence in which the character codes that are acceptable to a computer is ordered for purposes of sorting, merging, and comparing.

Conversion Item -
A conversion string component that causes a data item in a CDC or IBM input record to be converted to an IBM or CDC format and transmitted to an output record. Also refers to a nested conversion string.

Conversion Specification -
A conversion string component consisting of an optional selector expression and a list of associated conversion items.

Conversion String -
A string of one or more conversion specifications; used as input parameters for the XREAD, XREREAD, and XWRITE subroutines.

COPY8P Utility -
A utility program used to convert IBM print files to CDC compatible print files.

CYBER Record Manager (CRM) -
A generic term relating to the common products BAM and AAM that run under NOS and NOS/BE operating systems and that allow a variety of record types, blocking types, and file organizations to be created and accessed. (See Basic Access Methods.) The COPY8P utility does not make use of the CYBER Record Manager.

Default -
A value assumed in the absence of a user-specified value declaration for the parameter involved. Values for many defaults are defined by the installation.

Display Code -
A 6-bit code representing a 63-character or 64-character computer character set.

EBCDIC -
The Extended Binary Coded Decimal Information Code representing a set of 256 characters as 8-bit codes.

End-of-Information (EOI) -
CRM defines end-of-information in terms of the file organization and file residence, as shown in table C-1.

TABLE C-1. END-OF-INFORMATION BOUNDARIES

File Organization	File Residence	Physical Position
Sequential	Mass storage	After the last user record
	Labeled tape in SI, I, S, or L format	After the last user record and before any file trailer labels
	Unlabeled tape in SI or I format	After the last user record and before any file trailer labels
	Unlabeled tape in S or L format	Undefined

Entry Point -
A location within a program to which control can be transferred from another program. Each entry point has a unique name.

Field -
A portion of a word or record; a subdivision of information with a record; also a generic entry in a file information table identified by a mnemonic.

Field Length -
The area in central memory allocated to a particular job; the only part of central memory that a job can directly access. Contrasts with mass storage space or tapes allocated for a job and on which user's files reside.

File -
A logically related set of information; the largest collection of information that can be addressed by a file name. A file starts at beginning-of-information and ends at end-of-information. Every file in use by a job must have a logical file name.

FILE Control Statement -
A control statement that supplies file information table values. Basic file characteristics such as organization, record type, and description can be specified in the FILE control statement.

File Information Table (FIT) -
A table through which a user program communicates with BAM. For direct processing through BAM, you must initiate establishment of this table. All file processing executes on the basis of information in this table.

File Organizer and Record Manager (FORM) -
A file organization and record management utility callable by control statements.

Folding -
The process of mapping more than one source character to a single destination character.

Item Locator -
A conversion string component that specifies which data fields in the current source record are to be used in a relational test. (See Selector Expression.)

Keyword -
A word that has special meaning to the 8-bit subroutines when used in a specific context.

Literal -
A character string, the value of which is implied by the ordered set of characters that make up the string.

Logical File Name (lfn) -
The name given to a file being used by a job. The name must be unique for the job and must consist of one to seven letters or digits.

Mass Storage -
A disk pack that can be accessed randomly. Extended memory is not considered mass storage.

Module -
A specific function of the 8-bit subroutines package. Unneeded modules can be eliminated in order to conserve space in the field length.

Next Bit -
The bit succeeding the last bit read or written.

Next Record -
The record succeeding the last record read or written.

Noise -
The number of characters the tape drivers discard as being extraneous noise rather than a valid record.

Partition -

CRM defines a partition as a division within a file with sequential organization. Generally, a partition contains several records or sections. Implementation of a partition boundary is affected by file structure and residence, as shown in table C-2.

Physical Record -

On magnetic tape, information between inter-record gaps. A physical record need not contain a fixed amount of data.

Record -

The largest collection of information passed between CYBER Record Manager and a user program in a single read or write operation. You define the structure and characteristics of records within a file by declaring a record format. The beginning and ending points of a record are implicit in each format.

Record Type -

CYBER Record Manager defines eight record types established by the RT field in the file information table.

Section -

CRM defines a section as a division within a file with sequential organization. Generally, a section contains more than one record and is a division within a partition of a file. A section terminates with a physical representation of a section boundary, as shown in table C-3.

The NOS/BE operating system equates a section with a system-logical-record of level 0 through 16_8.

Selector Expression -

A conversion specification component that indicates 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.

Sequential File -

A file with records stored in the physical order in which they were written. No logical order exists other than the relative physical record position.

Simple Item Conversion -

A conversion string component that performs data conversions to or from IBM format. (See Conversion Item.)

Working Storage Area (WSA) -

An area within the user's field length intended for receipt of data from a file or transmission of data to a file.

XCOMP -

A utility subroutine that compares two character strings not necessarily of the same character set.

XFILE -

An input/output subroutine that defines the input/output file. No operation is performed on the file by XFILE; however, workspace and other specific information is defined.

TABLE C-2. PARTITION BOUNDARIES

Device	Record Type (RT)	Block Type (BT)	Physical Boundary
PRU device	W	I	A short PRU of level 0 containing a one-word deleted record pointing back to the last I block boundary, followed by a control word with a flag indicating a partition boundary
	W	C	A short PRU of level 0 containing a control word with a flag indicating a partition boundary
	D,F,R, T,U,Z	C	A short PRU of level 0 followed by a zero-length PRU of level 17_8
	S	-	A zero-length PRU of level number 17_8
S or L format tape	W	I	A separate tape block containing as many deleted records of record length 0 as required to exceed noise record size, followed by a deleted one-word record pointing back to the last I block boundary, followed by a control word with a flag indicating a partition boundary
-	W	C	A separate tape block containing as many deleted records of record length 0 as required to exceed noise record size, followed by a control word with a flag indicating a partition boundary
	D,F,T, R,U,Z	C,K,E	A tapemark
	S	-	A tapemark
Any other tape format	-	-	Undefined

TABLE C-3. SECTION BOUNDARIES

Device	Record Type (RT)	Block Type (BT)	Physical Representation
PRU device	W	I	A deleted one-word record pointing back to the last I block boundary followed by a control word with flags indicating a section boundary At least the control word is in a short PRU of level 0
	W	C	A control word with flags indicating a section boundary. The control word is in a short PRU of level 0
	D,F,R, T,U,Z	C	A short PRU with a level less than 17_8
	S	-	Undefined
S or L format tape	W	I	A separate tape block containing as many deleted records of record length 0 as required to exceed noise record size, followed by a deleted one-word record pointing back to the last I block boundary, followed by a control word with flags indicating a section boundary
	W	C	A separate tape block containing as many deleted records of record length 0 as required to exceed noise record size, followed by a control word with flags indicating a section boundary
	D,F,R, T,U,Z	C,K,E	Undefined
	S	-	Undefined
Any other tape format	-	-	Undefined

XMOVE -
A utility subroutine that moves a designated character string from a source location to a destination location, translating from one character code to another during the move.

XPACK -
A utility subroutine that compresses 8-bit byte character data from a 5-character, 12-bit byte per word internal format containing 8-bit ASCII or EBCDIC characters. When the data is packed, seven 8-bit character codes are placed right-justified in a word; and the leftmost 4 bits, as well as any unused character positions are set to zero.

XPAND -
A utility subroutine that reverses the process performed by XPACK and unpacks the 8-bit byte compressed string into words containing 12-bit character bytes.

XREAD -
An input/output subroutine used to read a next record from the file workspace area, convert the record (if specified in the calling sequence), and place the record in a destination area in memory.

XREREAD -
An input/output subroutine used to reread the current record, possibly with a different conversion specification.

XWRITE -
An input/output subroutine that takes data from a record area in memory, converts the data (if so specified), and writes the data in the file workspace area.

IBM TAPE FILE RECORD AND BLOCK FORMATS D

You can use the XREAD/XREREAD and XWRITE subroutines to read and write files in IBM compatible format. Parameters are required for the XFILE routine to describe the record and blocking format to be used. These parameters correspond to the data control block subparameters RECFM, BLKSIZE and LRECL on a data definition statement in IBM Job Control Language.

BLKSIZE PARAMETER

The BLKSIZE parameter specifies length, in number of 8-bit bytes, of the longest block expected, regardless of format. For variable format records, the specification must include space for a 4-byte block header. The maximum value for BLKSIZE is 32760.

LRECL PARAMETER

The LRECL parameter specifies length, in number of 8-bit bytes, of the longest logical record expected. For variable format records, the specification must include space for a 4-byte record header. The maximum value for LRECL is 32756 for variable format records and 32760 for fixed and unspecified format records. LRECL must not exceed BLKSIZE for all formats except variable spanned and variable blocked spanned.

RECFM PARAMETER

The RECFM parameter describes the physical arrangement of records in blocks and in files. The record/block formats are fixed, variable, and unspecified.

FIXED RECORD/BLOCK FORMAT

The fixed record/block formats include fixed (F), and fixed blocked (FB) formats.

Fixed Format

With F format, every record is exactly LRECL 8-bit bytes long. Each block contains exactly one record. The F format is shown in figure D-1.

Fixed Blocked Format

With FB format every record is exactly LRECL bytes long. Each block contains an integral number of records. The total block length must not exceed BLKSIZE. Figure D-2 shows the FB format.

VARIABLE RECORD/BLOCK FORMAT

The variable record/block formats include variable (V), variable blocked (VB), variable spanned (VS), and variable spanned blocked (VSB) formats.

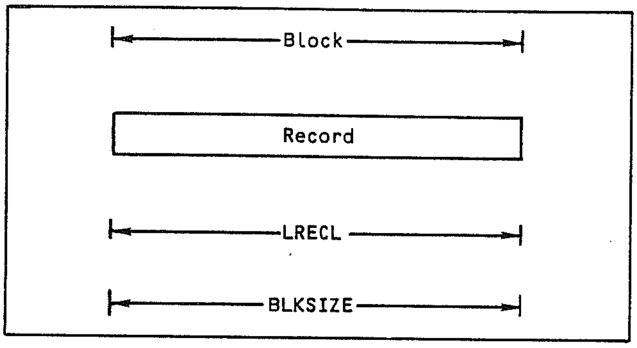


Figure D-1. Fixed Record/Block Format

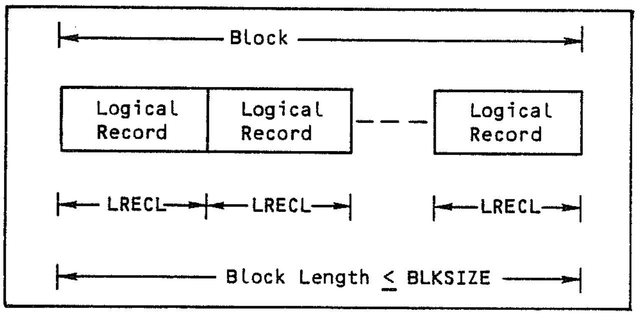


Figure D-2. Fixed Blocked Format

Variable Format

With V record and block format, the maximum record length is LRECL-4 bytes (32756), because variable length records are prefixed by a 4-byte record descriptor word (RDW) as shown in figure D-3. Each block contains exactly one record, prefixed by a block descriptor word (BDW) as shown in figure D-4.

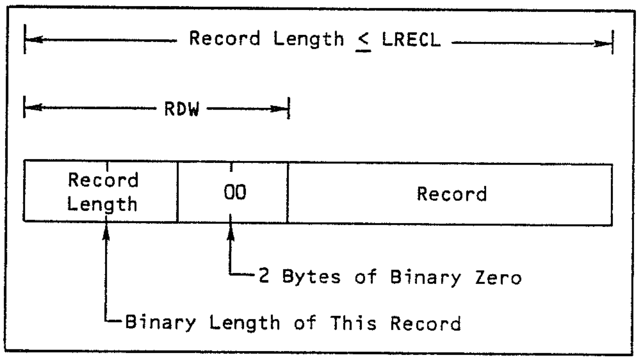


Figure D-3. Variable Record Format

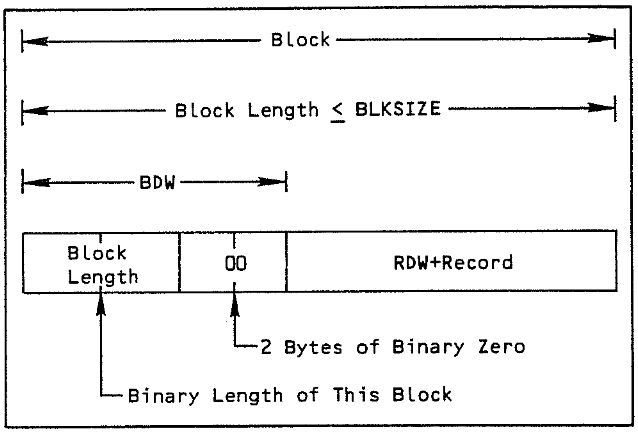


Figure D-4. Variable Block Format

Variable Blocked Format

The record format for VB type records is the same as the record format for V type records. The maximum record length is LRECL-4 bytes (32756), because the record is prefixed by a 4-byte RDW as shown in figure D-3.

Each VB block can contain several records. The total block length cannot exceed BLKSIZE. Figure D-5 shows the VB record and block format.

Variable Spanned Format

The variable spanned format makes the best use of blocks for variable length records and allows writing of logical records exceeding BLKSIZE. The record length cannot exceed LRECL.

The maximum value for LRECL for VS format is 32752, because both a block descriptor word and a segment descriptor word (SDW) are required. A record larger than the remaining block size (less than BLKSIZE-8) can be split into two or more blocks. SDW prefixes each record segment as shown in figure D-6.

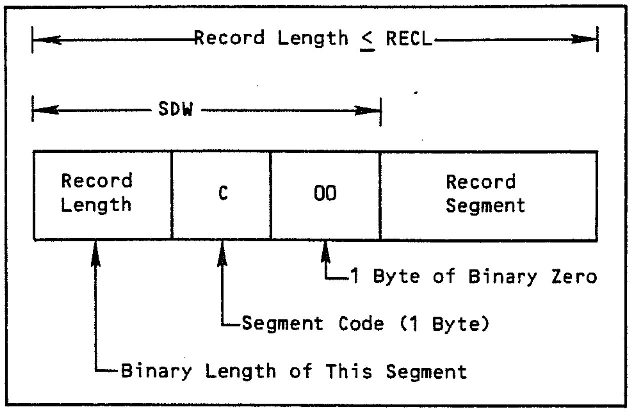


Figure D-6. Variable Spanned Format

Included in SDW is a 1-byte segment code with the following values:

0 Complete logical record; for code 0, the SDW is exactly one RDW

1 First segment of a multiple-segment record

2 Last segment of a multiple-segment logical record

3 Middle segment of a multiple-segment logical record

Each VS block contains exactly one record segment.

Variable Spanned Blocked Format

The VSB record and block format allows writing of logical records exceeding BLKSIZE. The record length must not exceed LRECL. LRECL for VSB format is 32752, because a record in VSB format must be prefixed by both a block size descriptor word and a segment descriptor word.

Several record segments can occupy a block. An attempt is made to fill the block to BLKSIZE; the attempt fails if the remaining space is less than or equal to 4 bytes, because at least one data byte must be written. If the attempt fails, the current block is ended and a new block is started. The VSB format is shown in figure D-7.

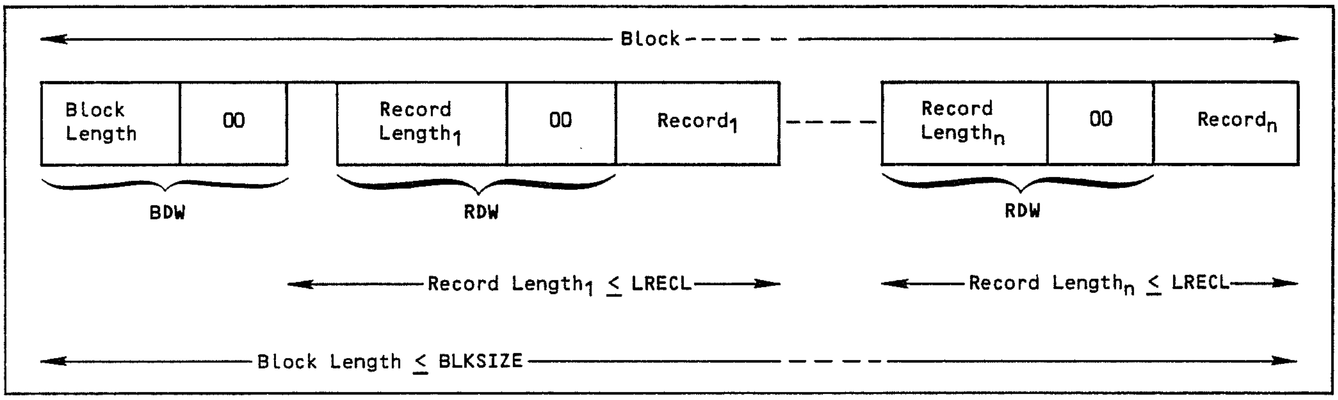


Figure D-5. VB Record and Block Format

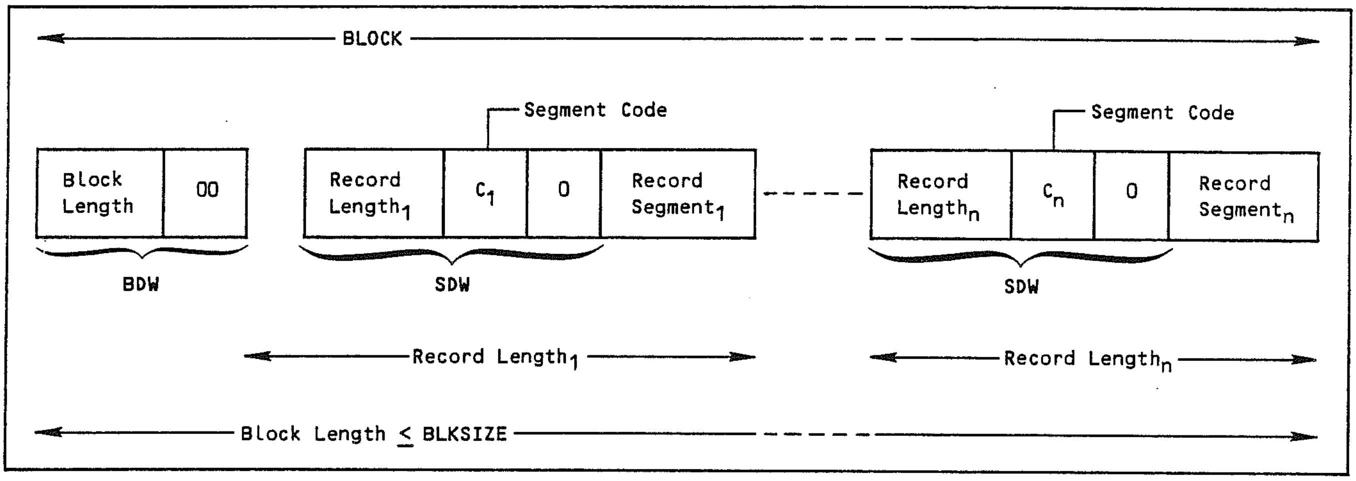


Figure D-7. Variable Spanned Blocked Format

UNSPECIFIED FORMAT

The unspecified format includes undefined length records (U), and undefined length blocked records (UB).

U format records can be any nonzero length up to LRECL. Each record occupies one block as shown in figure D-8.

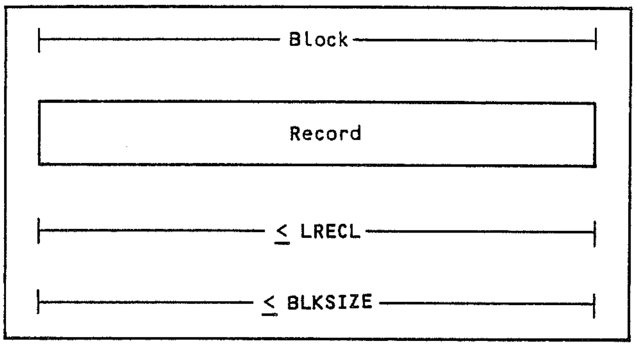


Figure D-8. Unspecified Format Records

UB records are not an IBM record format but are included to allow nonstandard record handling. Records can be any nonzero length up to LRECL, and LRECL can be smaller than BLKSIZE. Blocking and deblocking is handled as follows:

- Writing

  The length of a converted record is examined for fit in the current block. If space for the record exists, the record is appended to the data already in the block. If space does not exist, the block is written out and the record is used to start the next block.

- Reading

  If at least LRECL characters remain in the current block, LRECL characters are delivered to you. If more than zero but fewer than LRECL characters remain, the remaining characters are delivered to you.

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

CDC AND IBM DATA FORMATS E

This appendix discusses the CDC and IBM data formats that can be processed by the 8-bit subroutines.

IBM DATA FORMATS

The data formats found in IBM format files include character data and numeric data.

CHARACTER DATA

EBCDIC character codes are stored in IBM systems in 8-bit bytes. Each bit position within a byte is assigned a bit number. Numbers are assigned low to high, left to right. IBM EBCDIC bit numbers are E0 through E7 as shown in figure E-1.

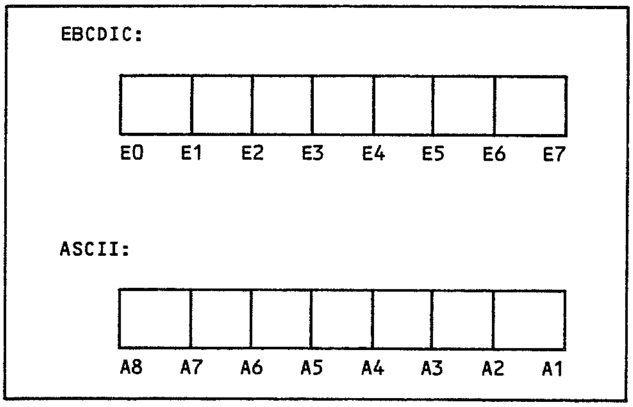


Figure E-1. IBM Storage of 8-bit EBCDIC and ASCII Codes

ASCII character codes are stored in IBM systems in 8-bit bytes. Bit numbers are assigned within a byte low to high, left to right. IBM ASCII bit numbers are A1 through A8 as shown in figure E-1.

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; four bytes comprise a whole-word (full-word); eight bytes are a double-word.

IBM systems use four forms of numeric data: fixed-point binary, floating-point hexadecimal, packed decimal, and decimal signed numeric. The numeric data formats are shown in figure E-2.

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. This format can also be referred to as signed integer format. Negative values are represented in two's complement form.

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 seven 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.

The greatest precision is achieved when a floating-point number is normalized. The fraction part of the floating-point number has a nonzero, high-order, hexadecimal digit produced by shifting the fraction left until the high-order, hexadecimal digit is nonzero 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 can be zero.

Packed Decimal

Because decimal numbers can be expressed by four bits, two 1-digit decimal values can be packed into one 8-bit byte. Variable length fields are used to contain packed decimal values; the rightmost four bits of the low-order byte of the field contains the value.

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 format, this representation is required for character set sensitive input/output devices. A zoned format number carries the sign in the leftmost four bits of the low order byte. The zoned format is not used in decimal arithmetic operations.

CDC DATA FORMATS

CDC data can be formatted in internal 6-bit byte display code, internal 12-bit byte code, and arithmetic data codes.

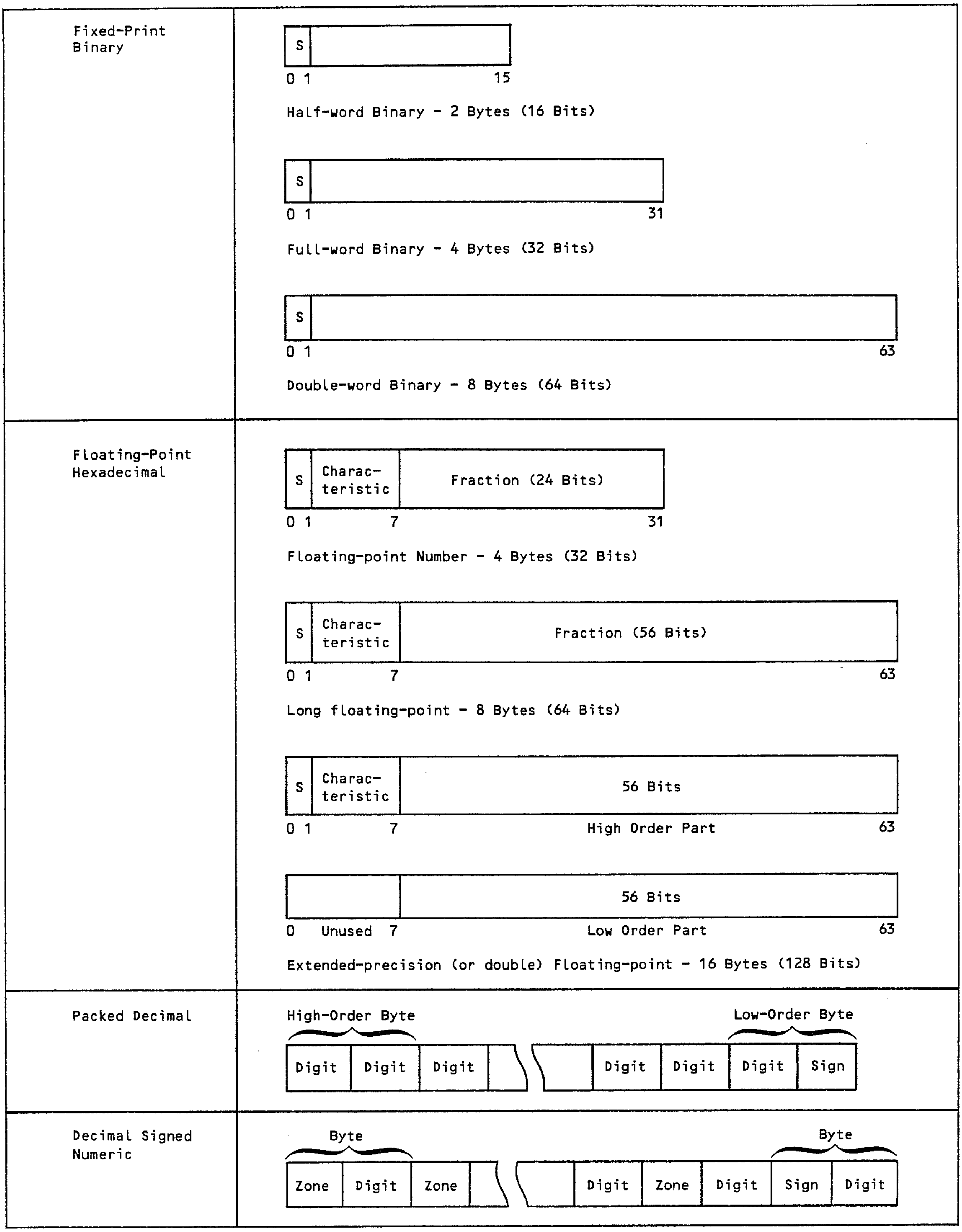


Figure E-2. IBM Numeric Data Formats

INTERNAL STORAGE OF 6-BIT DISPLAY CODE

Six-bit coded information is represented in the central memory unit of the CDC system in 6-bit bytes, ten bytes to a 60-bit word. Information is stored within a byte as shown in figure E-3. Within a word 6-bit bytes are stored as shown in figure E-4.

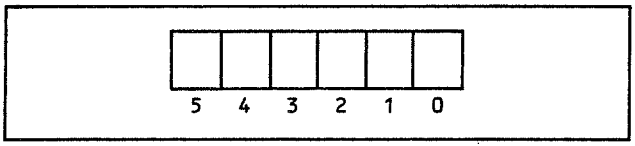


Figure E-3. CDC Display Code Bit Numbers

INTERNAL STORAGE OF 8-BIT BYTE CODES

IBM 8-bit data that has been copied from tape to disk is stored internally in CDC systems as binary data. The 8-bit subroutines can process this data as 8-bit data since the data format is identical to that of IBM tapes read by CYBER Record Manager.

INTERNAL STORAGE OF 12-BIT BYTE CODES

Under both NOS and NOS/BE, the CDC 12-bit ASCII code is the ASCII 7-bit code right-justified and stored internally in a 12-bit byte. Assuming that the bits are numbered from the right starting with 0, bits 0 through 6 contain the ASCII code, bits 7 through 10 contain zeros, and bit 11 (the flag bit) distinguishes the 12-bit ASCII 0000_8 code from the end-of-line byte. The leftmost bits are set to zero when the byte is stored in a word and are ignored when the character code is used. (See figure E-5.)

If EBCDIC codes are created by the 8-bit subroutines, the octal value of each EBCDIC code is stored right-justified in a 12-bit byte. The four remaining bits are zero-filled. (See figure E-5.)

Character data composed of either 12-bit ASCII codes or 12-bit EBCDIC codes must be aligned on the byte boundary. (See figure E-6.)

The NOS operating system supports a code set composed of 6/12 display code. This code set can not be processed by the 8-bit subroutines.

ARITHMETIC DATA

The CDC arithmetic data types are integer and floating point. The arithmetic data formats are shown in figure E-7. Display code signed overpunch numeric is CDC COBOL defined.

Integer

Integer data is stored in a 60-bit central memory word. The binary representation of the integer is right-justified 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. 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 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_8). 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.

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, eight 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.

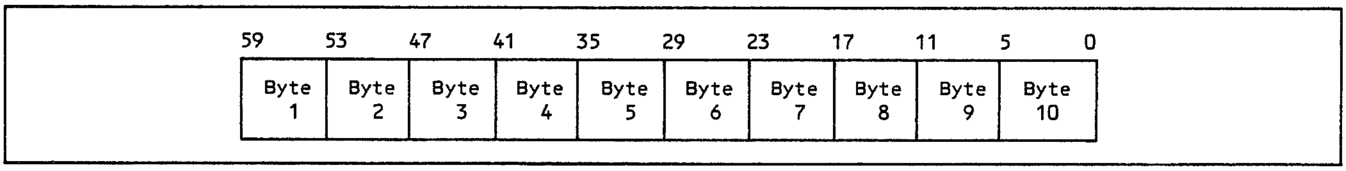
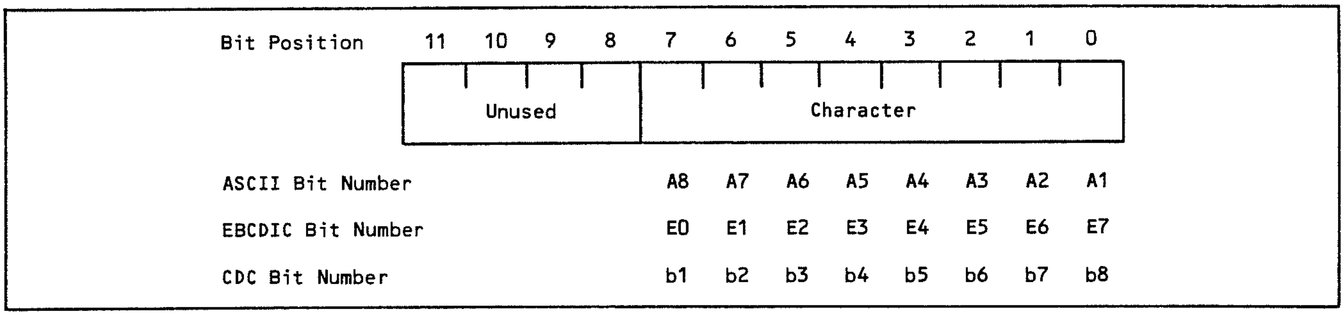


Figure E-4. CDC Display Code Byte Numbers



Bit Position	11	10	9	8	7	6	5	4	3	2	1	0
	Unused				Character							
ASCII Bit Number					A8	A7	A6	A5	A4	A3	A2	A1
EBCDIC Bit Number					E0	E1	E2	E3	E4	E5	E6	E7
CDC Bit Number					b1	b2	b3	b4	b5	b6	b7	b8

Figure E-5. CDC Format Used to Store 12-Bit Byte Codes

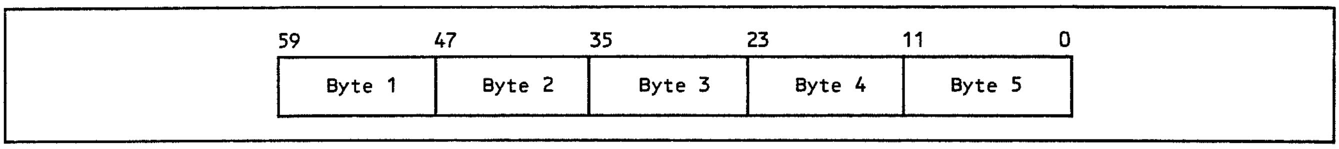


Figure E-6. CDC 5-Byte Word Format

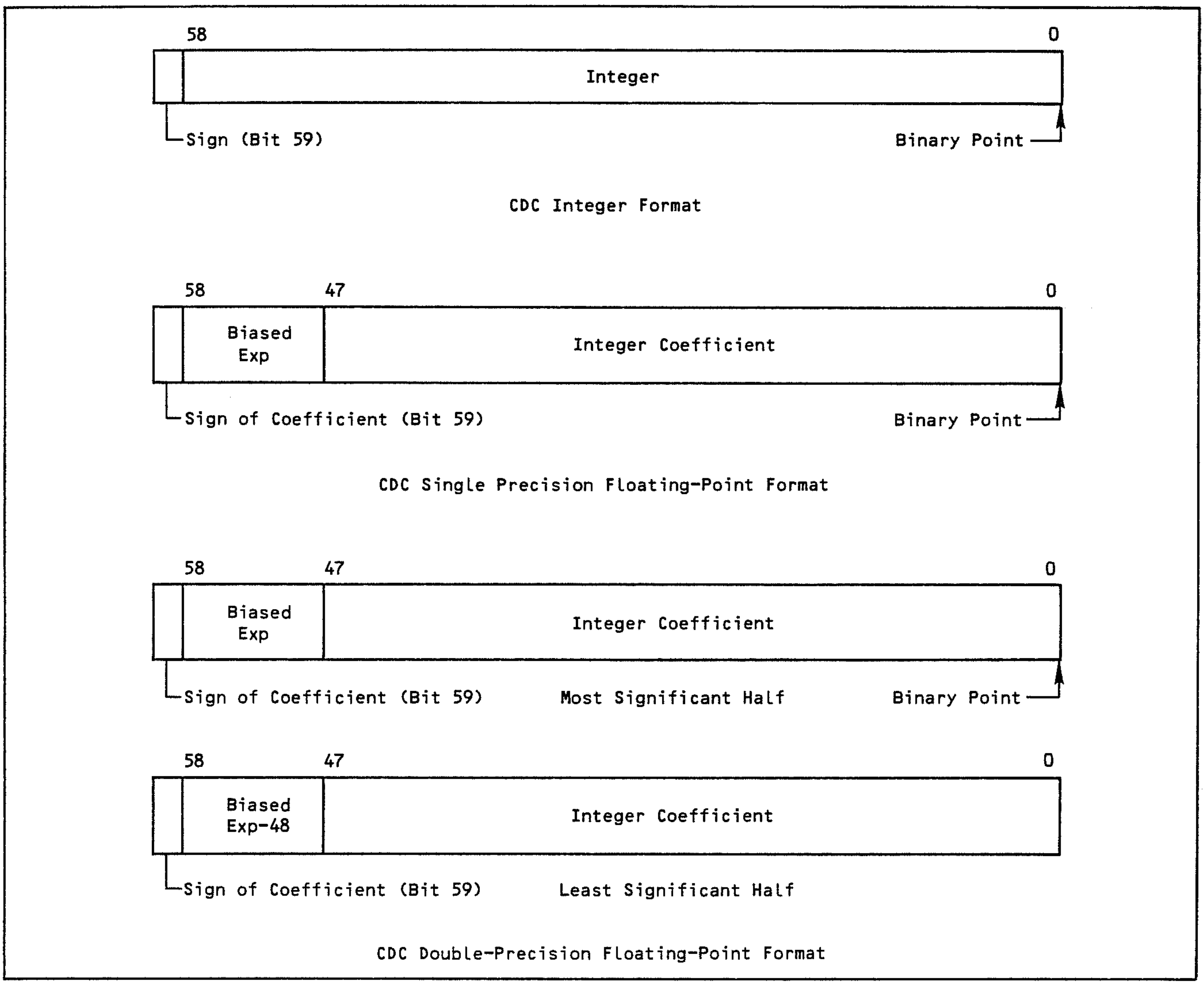


Figure E-7. CDC Numeric Data Formats

Display Code Numeric Signed Overpunch

A string of display code decimal digits form a display code signed overpunch number. The sign of the number is indicated by transforming the units digit (low order) of the number as shown in table E-1.

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 insert the plus sign into a positive value and transform the units digit.

TABLE E-1. VALUE OF DISPLAY CODE NUMERIC SIGN

Value	Units Digit	Transformed to:	Corresponds to:
Positive	0 1 through 9	> A through I	12-0 card punch† 12-1 through 12-9 punch
Negative	0 1 through 9	∧ J through R	11-0 card punch† 11-1 through 11-9 punch

†This character translation is not supported under NOS for 029 keypunch codes.

PRINT FORMAT-CDC 596-6 PRINT TRAIN

Any available printer can print information coded in 6-bit display code; however, 8-bit data coded in internal 12-bit format (leftmost four bits zero) must be converted to the CDC 12-bit ASCII code format and output to a CDC 580 Line Printer by using a CDC 596-6 Print Train having the ASCII graphic subset of 95 characters.

Only 12-bit ASCII code files can be printed by using the ASCII graphic 95-character set. For printing uppercase and lowercase data using the CDC 596-6 Print Train, the following conventions apply:

- All characters must be 7-bit ASCII code stored in 12-bit bytes.
- The end of a print line must be indicated by a zero byte in the lower twelve bits of the last central memory word of the line. Any other unused characters in the last word should be zero filled.
- Each line must start in the upper twelve bits of a central memory word.
- Each print line consists of up to 137 12-bit character bytes of 7-bit ASCII code.
- A maximum of 137 characters can be specified for a line, but no more than 136 are printed.
- The character in position 1 is interpreted as a carriage control character and that character is not printed.
- Legal hexadecimal values for print characters range from 20 through 7E, space through }. Values outside this range cause an error condition.
- If an unprintable character exists in a line (that is, a 12-bit byte containing ASCII code outside of the range 0040_8 through 0176_8), the number sign (#) appears in the first printable column of a print line and a space replaces the unprintable character.

When the FMT parameter of the XFILE subroutine file string equals 1, 2, or 3, XWRITE presets column 1 to blank, zero, or minus, respectively. The record starts with column 2.

Table F-1 shows the carriage control characters supported by the NOS operating system for a 580 line printer. The carriage control characters supported by the NOS/BE operating system are shown in table F-2.

The Q, R, S, T, and V format controls remain in effect until changed; all other carriage control characters must be supplied for each line controlled.

Refer to the appropriate system reference manual for information about carriage control tape or programmable format control (PFC).

TABLE F-1. NOS CARRIAGE CONTROL CHARACTERS

Character	Action
space	Single space
1	Eject page before printing
0	Skip one line before print {double space}
-	Slot two lines before print {triple space}
+	Suppress space before print
/	Suppress space after print
2	Skip to last line of form before print
Q†	Clear auto reject, remainder of line not printed
R	Select auto page eject
S	Select 6 lines per inch
T	Select 8 lines per inch
V	Eject page before print on a 580 printer; V loads a user supplied PFC array {validated users only}
8 ††	Skip to next punch in format channel 1 before print.
7	Skip to next punch in format channel 2 before print.
6	Skip to next punch in format channel 3 before print.
5	Skip to next punch in format channel 4 before print.
4	Skip to next punch in format channel 5 before print.
3	Skip to next punch in format channel 6 before print.
H	Skip to next punch in format channel 1 after print.
G	Skip to next punch in format channel 2 after print.
F	Skip to next punch in format channel 3 after print.
E	Skip to next punch in format channel 4 after print.
D	Skip to next punch in format channel 5 after print.
C	Skip to next punch in format channel 6 after print.

† The deselection of auto eject mode on a 580 line printer results in the deselection of 8 lines per inch, if previously selected.

†† No space after print. For all other control characters a line feed is issued after print.

TABLE F-2. NOS/BE CARRIAGE CONTROL CHARACTERS

Character	Action
A [†]	Space 1, skip to top of next page after print
B	Space 1, skip to last line of page after print
C	Space 1,skip to channel 6 after print
D	Space 1, skip to channel 5 after print
E	Space 1, skip to channel 4 after print
F	Space 1, skip to channel 3 after print
G	Space 1, skip to channel 2 after print
H	Space 1, skip to channel 11 after print
I	Space 1, skip to channel 7 after print
J	Space 1, skip to channel 8 after print
K	Space 1, skip to channel 9 after print
L	Space 1, skip to channel 10 after print
Q [††]	Clear auto page eject (Janus default)
R	Select auto page eject
S	Clear eight vertical lines per inch
T	Select eight vertical lines per inch
PM	Output remainder of line (up to 30characters) on the B display and the dayfile and wait for the Janus entry /OKxx
V	Specifies a new carriage control array to be loaded for a 580 printer
1	Skip to the top of the next page, no space
2	Skip to the last line on page, no space
3	Skip to channel 6, no space after print
4	Skip to channel 5, no space after print
5	Skip to channel 4, no space after print
6	Skip to channel 3, no space after print
7	Skip to channel 2, no space after print
8	Skip to channel 7, no space after print
X	Skip to channel 8, no space after print
Y	Skip to channel 9, no space after print
Z	Skip to channel 10, no space after print
+	No space, no space after print
0(zero)	Space 2, no space after print
-(minus)	Space 3, no space after print
blank	Space 1, no space after print

[†] The top of a page is indicated by a punch in channel 1 of the carriage control tape. The bottom of the page is indicated by a punch in channel 7.

[††] No printing takes place. The remainder of the line is ignored.

CONVERSION RULES G

Rules pertaining to all possible conversions are described in this appendix. Table G-1 shows conversion rules to be used when converting from IBM format to CDC format. The conversion rules for converting from CDC format to IBM format are shown in table G-2. The numbers in the tables refer to notes in the following subsections.

TABLE G-1. CONVERSION RULES: IBM TO CDC FORMATS

IBM Format	CDC Format										
	B	X	A	C	I	U	E	D	S	N	Z
B	1	2	2	2	3,4	3,5	3,6	3,7	3,8	3,9	3,10
X	11	12	12	12	13,4	13,5	13,6	13,7	13,8	13,9	13,10
H	14	15	15	15	4	5	6	7	8	9	10
W	14	15	15	15	4	5	6	7	8	9	10
G	14	15	15	15	4	5	6	7	8	9	10
F	14	15	15	15	4	5	6	7	8	9	10
I	14	15	15	15	4	5	6	7	8	9	10
E	14	15	15	15	4	5	6	7	8	9	10
P	14,25	15,25	15,25	15,25	4,25	5,25	6,25	7,25	8,25	9,25	10,25
S	14,26	15,26	15,26	15,26	4,26	5,26	6,26	7,26	8,26	9,26	10,26

TABLE G-2. CONVERSION RULES: CDC TO IBM FORMATS

CDC Format	IBM Format									
	B	X	H	W	G	F	L	E	P	S
B	1	2	3,16	3,17	3,18	3,19,20	3,19,21	3,19,22	3,23,25	3,24,26
X	11	12	13,16	13,17	13,18	13,19,20	13,19,21	13,19,22	13,23,25	13,24,26
A	11	12	13,16	13,17	13,18	13,19,20	13,19,21	13,19,22	13,23,25	13,24,26
C	11	12	13,16	13,17	13,18	13,19,20	13,19,21	13,19,22	13,23,25	13,24,26
I	14	15	16	17	18	19,20	19,21	19,22	23,25	24,26
U	14	15	16	17	18	19,20	19,21	19,22	23,25	24,26
E	14	15	16	17	18	19,20	19,21	19,22	23,25	24,26
D	14	15	16	17	18	19,20	19,21	19,22	23,25	24,26
S	14	15	16	17	18	19,20	19,21	19,22	23,25	24,26
N	14	15	16	17	18	19,20	19,21	19,22	23,25	24,26
Z	14	15	16	17	18	19,20	19,21	19,22	23,25	24,26

NOTES FOR CONVERSION RULES, TABLES G-1 AND G-2

1. Applies Bm to Bn. The leftmost [min,(m,n)] bits are copied 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 (m-n) bits of the destination field are set to zero.

   BmBO can be used to skip m bits of the source record. BOBn can be used to zero an n-bit field in the destination area.

2. Applies Bm to Xn, An, and Cn. The source field is copied 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.

   BOXn (or BOAn or BOCn) can be used to set an n-character field in the destination area to all zeros.

3. Applies Bm to numeric type. The source field is treated as an m-bit positive binary integer.

   A BO source field is treated as zero.

4. Applies when converting to I. The source field is rounded (if necessary) to an integer, and the low-order 59 bits are the value. If the magnitude is $>2^{59}$ (that is, more than 59 bits are required), an error condition is flagged.

5. Applies when converting to U. The source field is rounded (if necessary) to an integer, and the high-order 48 bits are the value. The result is kept as a single-precision floating-point number. However, this number is denormalized (COBOL COMP-1 definition), if necessary, to keep the biased exponent $\geq 2000_8$.

6. Applies when converting 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 when converting 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 when converting 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 shown in figure G-1. The 11-0 and 12-0 card punch is not supported on NOS in a manner that translates as signed numeric overpunch.

0→<	(corresponds to 12-0 card punch)
Positive:	
1-9→A-J	(corresponds to 12-1→12-9 card punch)
0→V	(corresponds to 11-0 card punch)
Negative:	
1-9→J-R	(corresponds to 11-1→11-9 card punch)

Figure G-1. Amending of Low-Order Digits

9. Applies when converting 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 when converting 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 Xm to 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 (string)m to string(n). The source field is copied to the destination field from the left. If the source field corresponds to card (Hollerith) input, and if the card punches from 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 can be skipped. Using an m value of 0 allows setting a destination field to all spaces.

13. Applies (string) to numeric type. A source character string which is to be converted to numeric type must have the general form shown in figure G-2. The format closely follows that used by FORTRAN where a decimal value is optionally followed by a power of ten.

$\begin{bmatrix}+\\-\end{bmatrix}$ d

$\begin{bmatrix}+\\-\end{bmatrix}$ d E $\begin{bmatrix}+\\-\end{bmatrix}$ exp

where:

d Is in the form of a floating-point or integer constant

exp Is an unsigned integer exponent

Figure G-2. Numeric Type Format

Numbers are kept to a precision of at least 96 bits. Spaces are ignored. Spaces can 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 numeric to 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: two′s complement for IBM format and one′s complement for CDC 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 IBM format).

15. Applies numeric to (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 a maximum precision of the source item. The receiving field format can be determined by the following subrules (headings refer to the source field format).

<u>All</u>

A. If the destination field width is zero, no conversion takes place.

B. If the source item is indefinite or infinite (internal items only), the destination field is filled as shown in table G-3.

C. The maximum precision, p, of the source item can be determined from table G-4.

D. The destination field width (d) is calculated as follows:

Set d to n. If the source value is negative, set d to n-1.

E. If the source value is an integer, the rules under Integer are followed; otherwise, the rules under Floating-Point are followed.

TABLE G-3. DESTINATION FIELD FOR INFINITE OR INDEFINITE ITEMS

Condition	1	2	3	4 or More (Right-Justified in Field)
+∞	F	NF	INF	INF
-∞	F	-F	-NF	-INF
+?	D	ND	IND	IND
-?	D	-D	-ND	-IND

TABLE G-4. MAXIMUM PRECISION OF SOURCE ITEMS

Source Computer	Source Item Type	p(digits)	Number of Accurate Guaranteed Decimal Digits
IBM	H	5	4
	W	10	9
	G	19	18
	F	8	7
	L	17	16
	E	34	33
	Pm	2m-1	2m-1
	Sm	m	m
CDC	I	18	17
	U	15	14
	E	15	14
	D	29	28
	Sm	m	m
	Nm	m	m
	Zm	m	m

Integer

A. If the magnitude of the value is $\geq 10^d$, the rules under Type E are followed.

B. The value is converted to a decimal integer and placed in the destination field, right-justified. All leading zeros are replaced (except in the units position) by spaces.

C. If the value is negative and n>1, a - is placed immediately to the left of the leftmost digit. Otherwise (must be -0 in a 1-character field), the 0 is replaced by a -.

Floating-Point

A. The minimum number of digit positions required to use this representation, r, is determined as follows:

   If |value|>1, then r=K,

   where $10^{K-1}<|value|<10^{K}$

   If |value|<1, then r=K-1+min(p,d-5),

   where $10^{-K}<|value|<10^{-K+1}$

B. If r<(d-1), the rules under step C are followed; otherwise, the rules under Type E are followed.

C. The value is converted according to one of the following formats and the result string is placed, right-justified, in the destination field. The value is rounded, if necessary, to the indicated number of places.

   (1) If (x>1) and [(r=d-1) or (r>p)]:

   d1d2....di   i=r

   (2) If (x>1) and (r<d-1) and (r<p):

   d1d2...di.di+1...dj i=r,j=min(p,d-1)

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

   0.d1d2...dj   j=K-1+p

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

   .d1d2...dj   j=d-1

D. If the value is negative, a - is placed immediately to the left of the leftmost nonblank character.

Type E

A. If d<6 and $|value|<.95\times10^{-99}$, 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 -. If d>6 and $|value|<.95\times10^{-99}$, the rule under step B is followed.

B. The value is converted according to the following format:

   d1.d2...djEeee   j=min(p,d-5)

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

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

   d1.d2...djE-nnn   j-min(p,d-6)

   The value is rounded to the indicated number of digits and placed, right-justified, in the destination field.

C. If the value is negative, a - is placed immediately to the left of the leftmost nonblank character.

16. Applies when converting to H. The source value is rounded, if necessary, to a two's complement integer, and the low-order 16 bits are taken as the value. If significance is lost, an error is flagged.

17. Applies when converting to W. The source value is rounded, if necessary, to a two's complement integer, and the low-order 32 bits are taken as the value. If significance is lost, an error is flagged.

18. Applies when converting to G. The source value is rounded, if necessary, to a two's complement integer, and the low-order 64-bits are taken as the value. If significance is lost, an error is flagged.

19. Applies when converting to F, L, and E. The source value is converted to an IBM format floating-point number rounded to the indicated number of bits. If the source magnitude is too large ($>\sim5\times10^{75}$), the largest possible number is supplied. If the source magnitude is too small ($<\sim5\times10^{-75}$) but not zero, the smallest nonzero number is supplied.

20. Applies when converting to F. The resultant value is a 4-byte field of 21- to 24-bit precision (IBM short floating-point).

21. Applies when converting to L. The resultant value is an 8-byte field of 53- to 56-bit precision (IBM long floating-point).

22. Applies when converting to E. The resultant value is an 16-byte field of 110- to 112-bit precision (IBM extended-precision floating-point). The low-order 14- to 16-bits might not be accurate, since only 96-bit precision is guaranteed.

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

24. Applies when converting to Sm. The source value is rounded, if necessary, to an integer and converted to a decimal string. If $|value| \geq 10^{m}$, overflow has occurred and an error is flagged.

25. The sign of the field and low-order (units) numeric place, for P fields, are contained in the low-order (rightmost) byte as shown in figure G-3.

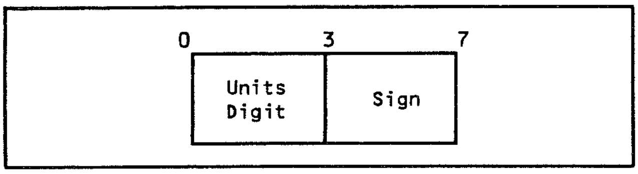


Figure G-3. Low-order Byte Format for P Fields

Valid sign codes used when a P field is read are as follows:

Bit Pattern	Sign
1010	+
1011	-
1100	+
1101	-
1110	+
1111	+

The EBCDIC sign code generated is 1101 for a negative sign and 1100 for a positive sign.

26. The sign of the field and low-order (units) numeric places for S fields in EBCDIC data are contained in the low-order (rightmost) byte as shown in table G-5.

The sign of the field and low-order (units) numeric digits for S fields in ASCII data are contained in the low-order byte as shown in table G-6.

TABLE G-5. SIGN POSITION FOR EBCDIC FIELDS

Low-Order Digit	Character Placed in That Position	
	+ (Sign)	-
0	{	}
1	A	J
2	B	K
3	C	L
4	D	M
5	E	N
6	F	O
7	G	P
8	H	Q
9	I	R

TABLE G-6. SIGN POSITION FOR ASCII FIELDS

Low-Order Digit	Character Placed in That Position	
	+ (Sign)	-
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

8-BIT SUBROUTINES/FORM COMPARISON

Because the 8-bit subroutines are utilized by FORM to perform IBM/CDC conversions, FORM has most of the capabilities of the 8-bit subroutines, including the capability of maintaining 8-bit significance in converted data. The 8-bit subroutines handle the same character sets as FORM. Differences include management of print files, ease of usage, and minor functional differences.

The 8-bit subroutines have the same conversion capabilities as FORM, including identical conversion string syntax. However, conversion strings can be changed from record to record when using the 8-bit subroutines, whereas FORM record specifications are set once per output file and can be varied from record to record only by comparing a record field to a literal quantity. The 8-bit subroutines only reformat data fields sequentially; FORM has an automatic data reformatting capability that allows you to reorder data fields and insert literals. When the 8-bit subroutines process a file, all input records are included in the output file; FORM has a record selection capability which allows you to select certain records for inclusion in the output file. The 8-bit subroutines are primarily record oriented and their use is restricted to sequential files; FORM is primarily file oriented and can be used to process CDC random access files.

The 8-bit subroutines handle print files differently than FORM. Print files processed by the 8-bit subroutines must be organized by you, but any character in the 95-graphic ASCII character set can be used. A utility program, COPY8P, is provided in the 8-bit subroutines to automatically print IBM print files by using the CDC 595-6 Print Train. FORM allows automatic organization of print file functions, such as paging, line spacing, and titling. Also, FORM print files can contain only CDC display code characters.

Other minor functional differences between the 8-bit subroutines and FORM include the following:

- FORM provides automatic sequencing of records; the 8-bit subroutines have no such provision.
- The 8-bit subroutines process IBM card files as free-form binary input and output; FORM does not.

FUTURE SYSTEM MIGRATION GUIDELINES

This appendix contains programming practices recommended by CDC for users of the software described in this manual. When possible, application programs based on this software should be designed and coded in conformance with these recommendations.

GENERAL GUIDELINES

Programmers should observe the following practices to avoid hardware dependency:

- Avoid programming with hardcoded constants. Manipulation of data should never depend on the occurance of a type of data in a fixed multiple such as 6, 10, or 60.
- Do not manipulate data based on the binary representation of that data. Characters should be manipulated as characters, rather than as octal diplay-coded values or as 6-bit binary digits. Numbers should be manipulated as numeric data of a known type, rather than as binary patterns within a central memory word.
- Do not identify or classify information based on the location of a specific value within a specific set of central memory word bits.
- Avoid using COMPASS in application programs. COMPASS and other machine-dependent languages can complicate migration to future hardware or software systems. Migration is restricted by continued use of COMPASS for stand-alone programs, by COMPASS subroutines embedded in programs using higher-level languages, and by COMPASS owncode routines used with CDC standard products. COMPASS should only be used to create part or all of an application program when the function cannot be performed in a higher-level language or when execution efficiency is more important than any other consideration.

INDEX

COMMENT SHEET

MANUAL TITLE: 8-Bit Subroutines Version 1 Reference Manual

PUBLICATION NO.: 60495500

REVISION: C

This form is not intended to be used as an order blank. Control Data Corporation welcomes your evaluation of this manual. Please indicate any errors, suggested additions or deletions, or general comments on the back (please include page number references).

_____ Please reply _____ No reply necessary

FOLD FOLI

CUT ALONG LINE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 8241 MINNEAPOLIS, MN.

POSTAGE WILL BE PAID BY ADDRESSEE

CONTROL DATA

Publications and Graphics Division
Mail Stop: SVL104
P.O. Box 3492
Sunnyvale, California 94088-3492

FOLD FOLD

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.
FOLD ON DOTTED LINES AND TAPE

NAME:

COMPANY:

STREET ADDRESS:

CITY/STATE/ZIP:

TAPE TAPE

CORPORATE HEADQUARTERS, P.O. BOX O, MINNEAPOLIS, MINN 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

LITHO IN U.S.A.

CONTROL DATA