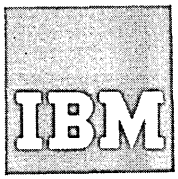
File Number S360-21
Form C24-3361-6

BOS



Systems Reference Library

IBM System/360
Basic Operating System
Assembler with Input/Output Macros
Specifications

This reference publication describes the assembler language and the input/output (I/O) macros supplied by IBM for use in writing programs for 8K disk-oriented System/360 installations. The general features of the assembler language are described first, followed by a description of each of three types of assembler language statements: machine-instruction, assembler-instruction, and macro-instruction statements. The description of the macro instructions consists of a description of each of the IBM-supplied I/O macros.

The reader should be familiar with the information presented in the publications:

IBM System/360 Principles of Operation, Form A22-6821;

IBM System/360 Basic Operating System, Programmer's Guide, Form C24-3372;

IBM System/360 Basic Operating System and IBM System/360 Basic Programming Support, Macro Definition Language, Form C24-3364.

For a list of other associated publications, refer to the IBM System/360 Bibliography, Form A22-6822.

PREFACE

This publication is intended as a guide for the programmer using the assembler language and its features. It contains all the information needed by the programmer to code an assembler-language program on the coding form. The information needed by the programmer to code user macro definitions (for inclusion into the macro library) is presented in the macro-definition language publication as listed on the cover of this publication.

The material in this publication is presented assuming that the reader has had experience with computer systems and has background in the basic programming concepts and techniques (or has completed basic courses of instruction in these areas). The publication IBM System/360 Principles of Operation (Form A22-6821) supplies the necessary background information about IBM System/360 operations (particularly storage addressing, data formats, and machine-instruction formats and functions). The publication IBM System/360 Basic Operating System, Programmer's Guide (Form C24-3372) supplies the necessary background information about IBM System/360 programming using Basic Operating System.

Seventh Edition (July 1968)

This is a major revision of, and obsoletes, C24-3361-5, and Technical Newsletters N24-5314, N24-5335, N24-5341, and N33-8534. The changes reflect the availability of BOS Release 17. The changes to the text, and small changes to illustrations, are indicated by a vertical line to the left of the change; changed or added illustrations are denoted by the symbol • to the left of the caption.

Significant changes or additions to the specifications contained in this publication are continually being made. When using this publication in connection with the operation of IBM equipment, check the latest SRL Newsletter for revisions or contact the local IBM Branch office. Requests for copies of IBM publications should be made to your IBM representative or to the local IBM branch office.

A form has been provided at the back of this publication for readers' comments. If the form has been detached, comments may be directed to IBM Laboratory, Publications Dept., P.O. Box 24, Uithoorn, Netherlands.

Computer programs may be expressed in machine language, i.e., language directly interpreted by the computer, or in a symbolic language, which is much more meaningful to the programmer. The symbolic language, however must be translated into machine language before the computer can execute the program. This function is accomplished by an associated processing program.

Of the various symbolic programming languages, assembler languages are closest to machine language in form and content.

The assembler language discussed in this manual is a symbolic programming language for the IBM System/360. It enables the programmer to use all IBM System/360 machine functions, as if he were coding in System/360 machine language.

The Assembler translates or processes assembler-language programs into machine language for execution by the computer. The program written in the assembler language used as input to the Assembler is called the source program; the machine-language program produced as output from the Assembler is called the object program. The translation or processing procedure performed by the Assembler to produce the object program is called assembling. Often, as in this publication, the object program produced is also referred to as an assembly.

Compatibility

The System/360 Basic Operating System Assembler can assemble programs written in the Basic Programming Support (8K Card) assembler language or the Basic Programming Support (8K Tape) assembler language. It can also assemble programs written for the Assembler of the 7090/7094 Support Package for the IBM System/360. The System/360 Basic Operating System Assembler requires the EBCDIC punches for the + ' ( ) and = signs.

Any program written in the IBM System/360 Basic Operating System assembler language can be assembled by the System/360 Operating System Assembler (provided the necessary macros are in the Macro library) and the System/360 Basic Programming Support Assembler with the following exceptions:

1. The XFR assembler instruction is considered an invalid mnemonic operation code by System/360 Operating System Assemblers and System/360 Disk and Tape Operating System Assemblers.
2. The assignment, size, and ordering of literal pools may differ among the assemblers.

ASSEMBLER-LANGUAGE STATEMENTS

Program statements (source statements) written in assembler language may consist of: a name to identify the statement; a symbolic operation code (mnemonic) to identify the function the statement represents; and an operand, consisting of one or more items called operands, to designate the data or storage locations used in the operation, and space for comments.

Assembler-language programs may consist of up to four types of statements: machine-instruction, macro-instruction, and assembler-instruction, and comments statements.

Machine-instruction statements are one-for-one symbolic representations of System/360 machine instructions. The Assembler produces an equivalent machine instruction in the object program from each machine-instruction statement in the source program.

Macro-instruction statements cause the Assembler to retrieve a specially-coded symbolic routine from the macro library, modify the routine according to the information in the macro instruction, and insert the modified routine into the source program for translation into machine language. IBM supplies specially-coded input/output (I/O) routines as part of the macro library. The assembler language includes a set of I/O macro-instruction statements through which these routines can be retrieved and modified to suit particular needs for input or output. Also, the user can define his own library routines, and reference them through macro-instruction statements he defines himself. These routines and statements are defined according to a special language, the macro-definition language, and are processed by the Assembler in the same manner as the IBM I/O routines and macro-instruction statements. The macro-definition language is presented in the macro-definition language publication, as listed on the front cover of this publication.

The Assembler program, in addition to its translation function, provides auxiliary functions that assist the programmer in checking and documenting programs, in controlling storage-address assignment, in program sectioning and linking, in data and storage-field definition, and in controlling the Assembler program itself. Assembler-instruction statements specify these auxiliary functions to be performed by the Assembler, and, with a few exceptions, do not result in the generation of any machine-language code by the Assembler.

Predefined mnemonic codes are provided in the assembler language for all machine-instruction, assembler-instruction, and IBM-supplied I/O macro-instruction statements. Additional extended mnemonics are provided for the various forms of the Branch-on-Condition machine instruction.

The assembler language provides for the symbolic representation of any addresses, machine components (such as registers), and actual values needed in source statements. Also provided is a variety of forms of data representation.

PROGRAMMER AIDS

The assembler program provides auxiliary functions that assist the programmer in checking and documenting programs, in controlling address assignment, in segmenting a program, in data and symbol definition, in generating macro-instructions, and in controlling the assembly program itself. Mnemonic codes, specifying these functions are provided in the language.

Variety in Data Representation: Decimal, binary, hexadecimal, or character representation of machine-language binary values may be employed by the programmer in writing source statements. The programmer selects the representation best suited to his purpose.

Base-Register Address Calculation: As discussed in the IBM System/360 Principles of Operation manual, the System/360 addressing scheme requires the designation of a base register (containing a base address value) and a displacement value in specifying a storage location. The assembler assumes the clerical burden of calculating storage addresses in these terms for the symbolic addresses used by the programmer. The programmer retains control of base register usage and the values entered therein.

Relocatability: The object programs produced by the assembler are in a format enabling relocation from the originally assigned storage area to any other suitable area.

Sectioning and Linking: The assembler language and program provide facilities for partitioning an assembly into one or more parts called control sections. Control sections may be added or deleted when loading the object program. Because control sections do not have to be loaded contiguously in storage, a sectioned program may be loaded and executed even though a continuous block of storage large enough to accommodate the entire program may not be available.

The linking facilities of the assembler language and program allow symbols to be defined in one assembly and referred to in another, thus effecting a link between separately assembled programs This permits reference to data and/or transfer of control between programs A discussion of sectioning and linking is contained in Program Sectioning and Linking.

Program Listings: A listing of the source program statements and the resulting object program statements is produced by the assembler for each source program it assembles. The programmer can partly control the form and content of the listing.

Error Indications: As a source program is assembled, it is analyzed for actual or potential errors in the use of the assembler language. Detected errors are indicated in the program listing.

IBM BASIC OPERATING SYSTEM/360 RELATIONSHIPS

The Assembler program operates as a part of the 8K disk resident version of the IBM System/360 Basic Operating System. The Assembler runs under control of the Supervisor, which provides the Assembler with all input/output and interruption services needed in assembling a source program. The object programs produced also operate as part of the disk resident system and make use of the functions provided by the resident control programs. For special applications not requiring disk, an independent Supervisor can be produced to control object programs outside of the Basic Operating System environment. Such independent programs do not include disk storage processing capability.

MACHINE REQUIREMENTS

To perform an assembly, the Assembler program requires a System/360 with at least the following features and units:

8,192 bytes of main storage. Additional main storage would be used by the Assembler to allocate area for input/output buffers and Assembler tables whenever they are needed.

The Assembler requires a 16K system if the size of the Supervisor exceeds 4,096 bytes of main storage.

Standard instruction set.

Either one multiplexor or one selector channel.

One IBM 2311 Disk Storage Drive. The following system programs must be present in the resident disk pack: IPL Loader, Supervisor, Job Control, Assembler, and Macro Library routines for all macros issued. At least one disk workarea must be provided. For faster processing, an additional disk drive can be used to split the Assembler workarea. See AWORK (Assembler Workfile) Cards.

One reader IBM 1442, 2501, 2520 (Model A1 or B1), 2540 or 2400-series tape unit.* This device may be the same I/O unit used for punching the output deck.

One punch IBM 1442, 2520, 2540, or 2400-series tape unit* (if the Assembler output deck is to be punched). This device may be the same I/O unit used for reading the source deck.

One printer IBM 1403, 1404, 1443, or 2400-series tape unit* (if the program listing is to be printed).

*At least one IBM 2400-series magnetic tape unit is required for any of the following conditions:

1. If the source program is to be read from tape.
2. If the output deck from the Assembler is to be written on tape.
3. If the program listing is to be written on tape.

A second tape unit is required if both the output deck and the program listing are to be written on tape. (See Figure 55 for 7-track tape requirements.)

An IBM 2400-series magnetic tape unit may be used for source input or object program output, only if sufficient main storage is available. See the Basic Operating System Programmer's Guide for information relating to main storage requirements when tape is used.

If an IBM 1052 Printer-Keyboard is available, it may be used for the output of special diagnostic messages.

To execute object programs, the minimum machine requirements are a System/360 with at least the following features and units.

8,192 bytes of main storage (except BSC applications, which require 16,384 bytes of main storage).

Standard instruction set.

Either one multiplexor or one selector channel.

One IBM 2311 Disk Storage Drive.

One Card Read-Punch (1442 or 2540 or 2501 and 2520).

Data Conversion special feature, if variable-length records are written on 7-track tape.

Additional units as required by the object program.

There are two basic approaches to a minimum resident system:

1. Problem program(s) that reside on the disk resident pack with the data to be processed must have:
   a. IPL Loader.
   b. Supervisor.
   c. Job Control.
   d. Appropriate problem program(s).
2. Problem program(s) that are loaded by means of a card reader must have:
   a. Items a-c above.
   b. Disk Linkage Editor.

A full description of the disk residence functions and the operating system environment is explained in detail in IBM System/360 Basic Operating System Programmer's Guide, C24-3372.

For the single drive system, these residence requirements must reside on each pack.

GENERAL INFORMATION

This section presents information about assembler language coding conventions, and assembler source statement structure.

ASSEMBLER LANGUAGE CODING CONVENTIONS

This subsection discusses the general coding conventions associated with use of the assembler language.

Coding Form

A source program is a sequence of source statements that are punched into cards. These statements may be written on the standard coding form, X28-6509 (Figure 1), provided by IBM. One line of coding on the form is punched into one card. The vertical columns on the form correspond to card columns.

Space is provided at the top of the form for program identification. Instructions to the keypunch operator can also be given; any character code that does not have a corresponding printer graphic can be assigned any special graphic to identify the code to the keypunch operator, who can then punch the corresponding card punch code wherever he encounters the special graphic. (See Character Set for the presentation of the valid character codes that can be used in a source program.) Neither the program information (Program, Programmer, Date, etc.) nor the instructions to the keypunch operator are punched into a card; they are for the user's own use.

The body of the form (Figure 1) is composed of two fields: the statement field, columns 1-71, and the identification-sequence field, columns 73-80. The identification-sequence field is not part of a statement and is discussed following the subsection Statement Format.

The entries (i.e., coding) composing a statement occupy columns 1-71 of a statement line and, if needed, columns 16-71 of a single continuation line.

Continuation Lines: When it is necessary to continue a statement on another line the following rules apply. Note that only one continuation line is permitted per statement.

1. Enter any nonblank character in column 72 of the statement line. This character must not be part of the statement coding. For a positional macro, there should be no blanks in the operand to the left of column 72.

2. Continue the statement on the next line, starting in column 16. All columns to the left of column 16 will be ignored.

Statement Boundaries

Source statements are normally contained in columns 1--71 of statement lines and columns 16--71 of any continuation lines. Therefore, columns 1, 71, and 16 are referred to as the "begin," "end," and "continue" columns, respectively This convention may be altered by the Input Format Control (ICTL) assembler instruction discussed later in this publication The continuation character, if used, always immediately follows the "end" column.

Statement Format

Statements may consist of one to four entries in the statement field. They are, from left to right: a name entry, an operation entry, an operand entry, and a comments entry. These entries must be separated by one or more blanks, and must be written in the order stated.

The coding form (Figure 1) is ruled to provide an eight-character name field, a five-character operation field, and a 56-character operand and/or comments field.

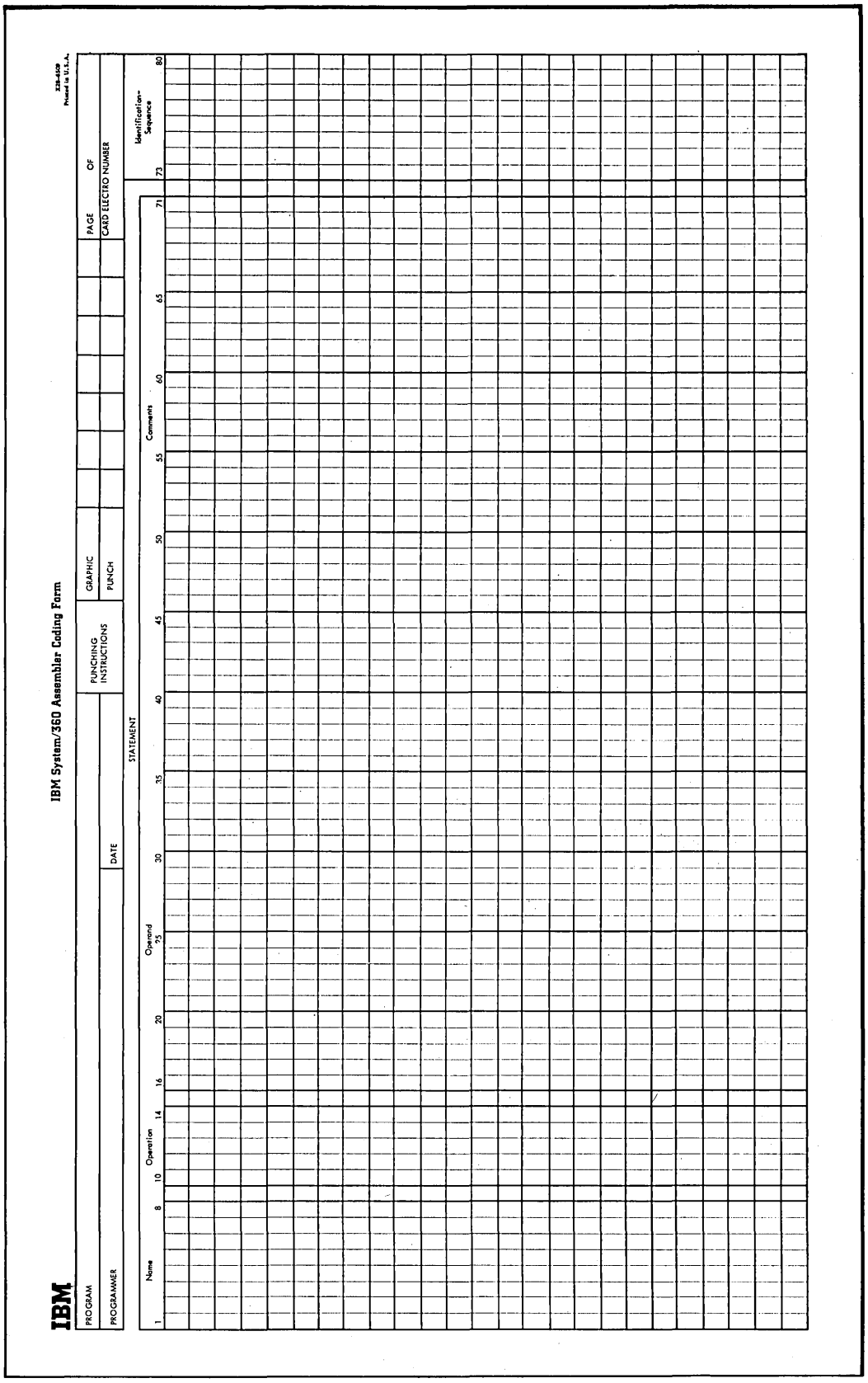


Figure 1. Coding Form

If desired, the programmer may disregard these boundaries and write the name, operation, operand, and comment entries in other positions, subject to the following rules.

1. The entries must not extend beyond statement boundaries (either the conventional boundaries or as designated by the programmer via the ICTL instruction).
2. The entries must be in proper sequence, as stated above.
3. The entries must be separated by one or more blanks.
4. If used, a name entry must be written starting in the begin column.
5. The name and operation entries must be completed in the first line of the statement, including at least one blank following the operation entry.

A description of the name, operation, operand, and comments entries follows:

Name Entries: The name entry is a symbol created by the programmer to identify a statement. A name entry is optional. The symbol must consist of eight characters or less, and be entered with the first character appearing in the begin column. If the begin column is blank, the assembler program assumes no name has been entered. No blanks may appear within the symbol.

Operation Entries: The operation entry is the mnemonic operation code specifying the machine operation or assembler functions desired. An operation entry is mandatory and must appear in the first statement line, starting at least one position to the right of the begin column. Valid mnemonic operation codes for machine and assembler operations are contained in Appendixes B and C of this publication. Valid operation codes consist of five characters or less. No blanks may appear within the operation entry.

Operand Entries: Operand entries are the coding that identifies and describes data to be acted upon by the instruction, by indicating such things as storage locations, masks, storage-area lengths, or types of data.

Depending on the needs of the instruction, one or more operands may be written. Operands are required for all machine instructions.

Operands must be separated by commas and no blanks must intervene between operands and the commas that separate them.

Symbols can appear in the operand field of a statement. Symbols that appear in the operand field must be defined. A symbol is considered to be defined when it appears in the name field of a statement.

The operands may not contain embedded blanks except as follows:

> If character representation is used to specify a constant, a literal, or immediate data in an operand, the character string may contain blanks, e.g., C'AB D'.

Comments Entries: Comments are descriptive items of information about the program that are to be inserted in the program listing. All valid characters (see Character Set in this section), including blanks may be used in writing a comment. The entry cannot extend beyond the end column (normally column 71), and a blank must separate it from the operand.

An entire line may be used for a comment by placing an asterisk in the begin column. Extensive comments entries may be written by using a series of lines with an asterisk in the begin column of each line or by using the aforementioned continuation line.

In statements where an optional operand entry is omitted but a comments entry is desired, the absence of the operand entry must be indicated by a comma preceded and followed by one or more blanks, as follows:

```
|Name   |Operation |Operand                  |
|-------|----------|-------------------------|
|       |END       |   COMMENT               |
```

Statement Example: The following example illustrates the use of name, operation, operand, and comments entries. A compare instruction has been named by the symbol COMP; the operation entry (CR) is the mnemonic operation code for a register-to-register compare operation, and the two operands (5,6) designate the two general registers whose contents are to be compared. The comments entry reminds the programmer that he is comparing "new sum" to "old" with this instruction.

```
|Name   |Operation  |Operand                  |
|-------|-----------|-------------------------|
|COMP   |CR         |5,6  NEW SUM TO OLD      |
```

Identification-Sequence Field

The identification-sequence field of the coding form (columns 73-80) is used to enter program identification and/or statement sequence characters. The entry is optional. If the field, or a portion of it, is used for program identification, the identification is punched by the user in the statement cards, and reproduced by the Assembler in the printed listing of the source program.

To aid in keeping source statements in order, the programmer may code an ascending sequence of characters in this field or a portion of it. These characters are punched into their respective cards, and, during assembly, the programmer may request the assembler to verify this sequence by the Input Sequence Checking (ISEQ) assembler instruction, which is discussed under Program Control Instructions.

Summary of Statement Format

The entries in a statement must always be separated by at least one blank and must be in the following order: name, operation, operand(s), comment.

Every statement, with the exception of the comments statement, requires an operation entry. Name and comment entries are optional. Operand entries are required for all machine instructions and most assembler instructions.

The name and operation entries must be completed in the first statement line, including at least one blank following the operation entry.

The name and operation entries must not contain blanks. Operand entries must not have a blank preceding or following the commas that separate them.

A name entry must always start in the "begin" column.

If the column after the end column is blank, the next line must start a new statement. If the column after the end column is not blank, the following line will be treated as a continuation line.

All entries must be contained within the designated begin, end, and continue column boundaries.

Character Set

Assembler-Language statements may be written using the following letters, numeric digits, and special characters:

Letters: 29 characters are classified as letters. These include the characters @, #, and $ as well as the alphabetic characters A through Z. The three additional characters are included so that the category can accommodate certain non-English languages.

Numeric Digits: 0 through 9

Special Characters: + - , = . * ( ) ' / & blank

These letters, digits, and special characters are only 51 of the set of 256 code combinations defined as the Extended Binary Coded Decimal Interchange Code (EBCDIC). Each of the 256 codes (including the 51 characters above) has a unique card punch code. Most of the terms used in Assembler-Language statements are expressed by the letters, digits, and special characters shown above. However, such Assembler- language features as character self-defining terms and character constants permit the use of any of the 256 card codes. Appendix A shows the various forms of EBCDIC codes.

ASSEMBLER-LANGUAGE STRUCTURE

The basic structure of the language can be stated as follows.

A source statement is composed of:

- A name entry (optional).
- An operation entry (mandatory).
- An operand entry (usually required).

A name entry is:

- A symbol.

An operation entry is:

- A mnemonic operation code representing a machine-, assembler-, or macro-instruction.

An operand entry is:

- One or more operands composed of one or more expressions, which, in turn, are

composed of a term or an arithmetic combination of terms.

Operands of machine instructions generally represent such things as storage locations, general registers, immediate data, or constant values. Operands of assembler instructions provide the information needed by the assembler program in order to perform the designated operation.

Figure 2 depicts this structure. Terms shown in Figure 2 are classed as absolute or relocatable. Terms are absolute or relocatable due to the effect of program relocation upon them. Program relocation is the loading of the object program into storage locations other than those originally assigned by the assembler program. A term is absolute if its value does not change upon relocation. A term is relocatable if its value changes by n when the program is relocated n bytes away from its assembled location.

The following subsection, Terms and Expressions, discusses these items as outlined in Figure 2.

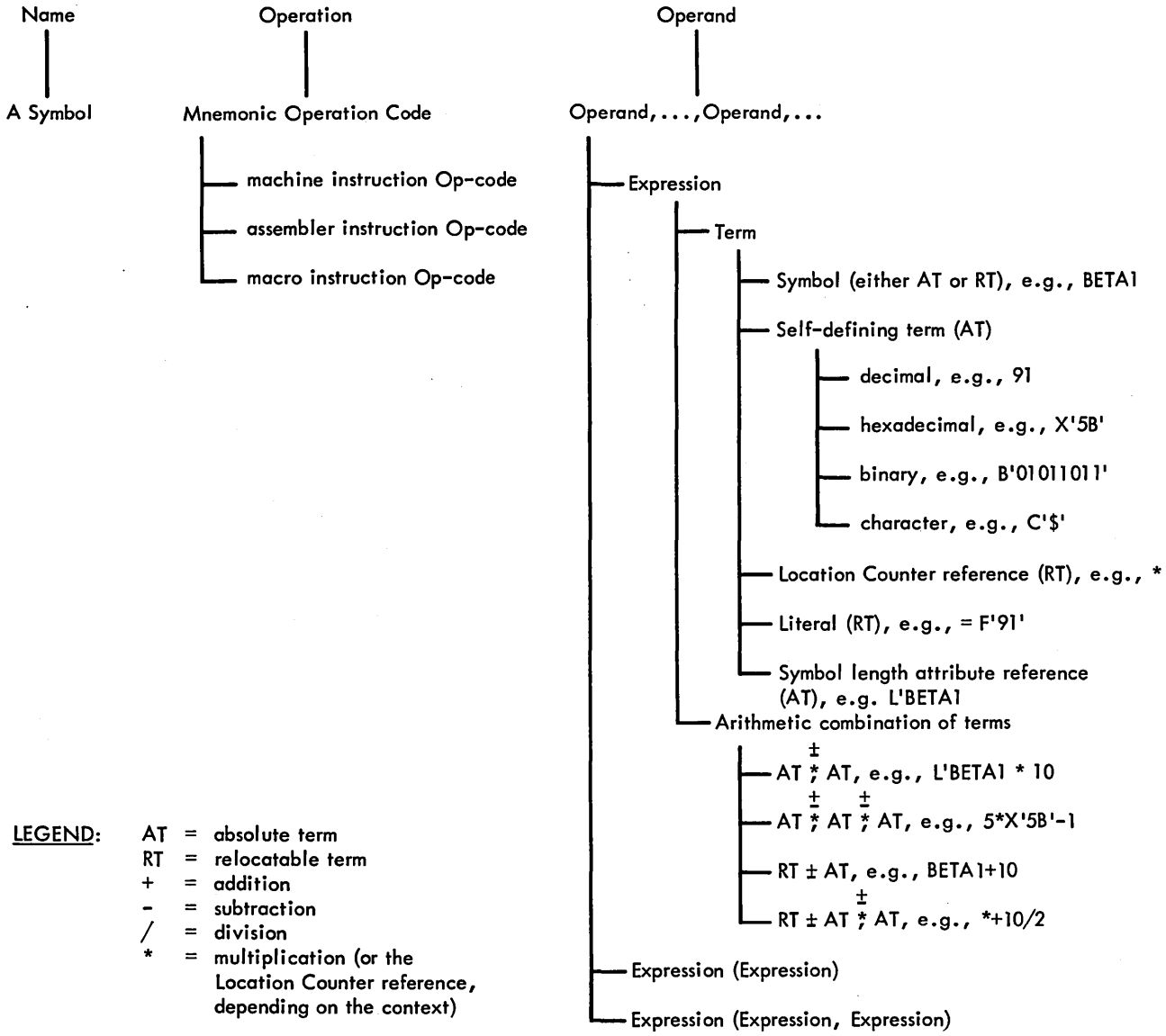


Figure 2. Assembler Language Structure--Machine and Assembler Instructions

TERMS AND EXPRESSIONS

TERMS

All terms represent a value. This value may be assigned by the assembler program (symbols, symbol length attribute location counter reference, literals) or may be inherent in the term itself (self-defining terms).

An arithmetic combination of terms is reduced to a single value by the assembler program.

The following material discusses each type of term and the rules for its use.

Symbols

A symbol is a character or combination of characters used to represent addresses or arbitrary values.

Symbols, through their use as names and in operands, provide the programmer with an efficient way to name and reference a program element. A symbol, created by the programmer for use as a name entry and in an operand, must conform to these rules:

1. The symbol must not consist of more than eight characters. The first character must be a letter. The other characters may be letters, digits, or a combination of the two.
2. No special characters may be included in a symbol.
3. No blanks are allowed in a symbol.
4. Symbols used by IOCS begin with the letter I. Therefore, user symbols in the problem program should not begin with the letter I. Also, a symbol or the first portion of a symbol (up to seven characters) in the problem program should not be the same as the file name in a DTF header entry, except when referring to that file in an IOCS macro instruction.
5. If a Supervisor is being assembled with a problem program, user symbols should not start with SYS because symbols used by the Supervisor start with SYS.

The following are valid symbols:

READER	LOOP2	$13
A23456	N	@PRICE
X4F2	S4	#LB1

The following symbols are invalid, for the reasons noted:

256B	(first character is not alphabetic)
RECORDAREA2	(more than eight characters)
BCD*34	(contains a special character - *)
IN AREA	(contains a blank)

Note: Any of several different combinations of characters can be installed in a printer. If the characters $, #, and @ are not included, either a blank space will occur or a different character will be printed when the code for one of these characters is sensed. This varies with the print arrangement that is used.

DEFINING SYMBOLS: The assembler program assigns a value to each symbol appearing as a name entry in a source statement. The value assigned to symbols naming storage areas, instructions, constants, and control sections represents the address of the leftmost byte of the storage field containing the named item. Since the addresses of these items may change upon program relocation, the symbols naming them are considered relocatable terms.

Symbols used as name entries in the Equate Symbol (EQU) assembler instruction are assigned the value designated in the operand entry of the instruction. Since the operand entry may represent a relocatable value or an absolute (i.e., nonchanging) value, the symbol is considered a relocatable term or an absolute term depending upon the value to which it is equated.

The value of a symbol may not be negative and may not exceed $2^{24}-1$.

A symbol is said to be defined when it appears as the name of a source statement. (A special case of symbol definition is discussed in Program Sectioning and Linking.)

Symbol definition also involves the assignment of a length attribute to the symbol. (The assembler program maintains an internal table -- the symbol table -- in which the values and attributes of symbols are kept. When the assembler program encounters a symbol in an operand, it refers to the table for the values associated with the symbol.) The length attribute of a symbol is the size, in bytes, of the storage field whose address is represented by the symbol. For example, a symbol naming an instruction that occupies four bytes of storage has a length attribute of four.

PREVIOUSLY DEFINED SYMBOLS: Some instructions require that a symbol appearing in the operand entry be previously defined. This simply means that the symbol, before it is used in an operand, must have appeared as a name entry in a prior statement.

GENERAL RESTRICTIONS ON SYMBOLS: A symbol may be defined only once in an assembly. That is, each symbol used as the name of a statement must be unique to that assembly. However, a symbol may be used in the name field more than once as a control section name (i.e., defined in the START, CSECT, or DSECT assembler statements described in Addressing -- Program Sectioning and Linking) because the coding of a control section may be suspended and then resumed at any subsequent point. The CSECT or DSECT statement that resumes the section must be named by the same symbol that initially named the section; thus, the symbol that names the section must be repeated. Such usage is not considered to be duplication of a symbol definition.

Self-Defining Terms

A self-defining term is one whose value is inherent in the term. It is not assigned a value by the assembler program. For example, the decimal self-defining term -- 15 -- represents a value of fifteen.

There are four types of self-defining terms: decimal, hexadecimal, binary, and character. Use of these terms is spoken of as decimal, hexadecimal, binary, or character representation of the machine language binary value or bit configuration they represent.

Self-defining terms are classed as absolute terms since the value they represent does not change upon program relocation.

USING SELF-DEFINING TERMS: Self-defining terms are the means of specifying machine values or bit configurations without equating the value to a symbol and using the symbol.

Self-defining terms may be used to specify such program elements as immediate data, masks, registers, addresses, and address increments. The type of term selected (decimal, hexadecimal, binary, or character) will depend on what is being specified.

The use of a self-defining term is quite distinct from the use of data constants or literals. When a self-defining term is used in a machine-instruction statement, its value is assembled into the instruction. When a data constant or literal is specified in the operand of an instruction, its address is assembled into the instruction

Decimal Self-Defining Term: A decimal term is simply an unsigned decimal number written as a sequence of decimal digits. High-order zeros may be used (e.g.,007). Limitations on the value of the term depend on its use. For example, a decimal term that designates a general register should have a value between 0 and 15 inclusively; one that represents an address should not exceed the size of storage. In any case, a decimal term may not consist of more than eight digits, or exceed 16,777,215($2^{24}-1$). A decimal term is assembled as its binary equivalent Some examples of decimal self-defining terms are: 8, 147, 4092, 00021.

Hexadecimal Self-Defining Term: A hexadecimal self-defining term is an unsigned hexadecimal number written as a sequence of hexadecimal digits. The digits must be enclosed in single quotation marks and preceded by the letter X; for example, X'C49'.

Each hexadecimal digit is assembled as its four-bit binary equivalent. Thus, a hexadecimal term used to represent an eight-bit mask would consist of two hexadecimal digits. The maximum value of a hexadecimal term is X'FFFFFF'.

The hexadecimal digits and their bit patterns are as follows:

0- 0000	4- 0100	8- 1000	C- 1100
1- 0001	5- 0101	9- 1001	D- 1101
2- 0010	6- 0110	A- 1010	E- 1110
3- 0011	7- 0111	B- 1011	F- 1111

A table for converting from hexadecimal representation to decimal representation is provided in Appendix E.

Binary Self-Defining Term: A binary self-defining term is written as an unsigned sequence of 1s and 0s enclosed in single quotation marks and preceded by the letter B, as follows: B'10001101'. This term would appear in storage as shown, occupying one byte. A binary term may have up to 24 bits represented.

Binary representation is used primarily in designating bit patterns of masks or in logical operations.

The following example illustrates a binary term used as a mask in a Test Under Mask (TM) instruction. The contents of GAMMA are to be tested, bit by bit, against the pattern of bits represented by the binary term.

Name	Operation	Operand
ALPHA	TM	GAMMA,B'¶0¶0¶1¶0¶'

Character Self-Defining Term: A character self-defining term consists of one to three characters enclosed by single quotation marks. It may be preceded by the letter C (this is not mandatory). All letters, decimal digits, and special characters may be used in a character term. In addition, any of the 256 punch combinations (shown in Appendix A) may be designated in a character self-defining term. Examples of character self-defining terms are as follows (the letter C preceding the quotation mark is optional):

```
C'/'          C' ' (blank)
C'ABC'        C'¶3'
```

Because of the use of quotes in the assembler language and ampersands in the macro language as syntactic characters, the following rule must be observed when using these characters in a character term.

For each single quotation mark or ampersand desired in a character term, two single quotation marks or ampersands must be written. For example, the character value A'# would be written as 'A''#', while a single quotation mark followed by a blank and another single quotation mark would be written as ''' '''.

Each character in the character sequence is assembled as its eight-bit code equivalent (see Appendix A). The two quotation marks or ampersands that must be used to represent a single quotation mark or ampersand within the character sequence are assembled as a single quotation mark or ampersand.

Location Counter Reference

The programmer may refer to the current value of the Location Counter at any place in a program, by using an asterisk in an operand. The asterisk represents the location of the first byte of currently available storage (i.e., after any required boundary adjustment). Using an asterisk in a machine-instruction statement is the same as placing a symbol in the name field of the statement and then using that symbol as an operand of the statement Because a Location Counter is maintained for each control section, a Location Counter reference designates the Location Counter for the section in which the reference appears.

A reference to the Location Counter may be made in a literal address constant (i.e., the asterisk may be used in an address constant specified in literal form). The address of the instruction containing the literal is used for the value of the Location Counter. A Location Counter reference may not be used in a statement which requires the use of a predefined symbol, with the exception of the EQU and ORG assembler instructions.

The Location Counter: A Location Counter is used to assign storage addresses to program statements. It is the assembler program's equivalent of the instruction counter in the computer. As each machine instruction or data area is assembled, the Location Counter is first adjusted to the proper boundary for the item, if adjustment is necessary, and then incremented by the length of the assembled item. Thus, it always points to the next available location. If the statement is named by a symbol, the value attribute of the symbol is the value of the Location Counter after boundary adjustment, but before addition of the length.

The assembler maintains a Location Counter starting at a double-word boundary for each control section of the program and manipulates each Location Counter independently as previously described. Source statements for each section are assigned addresses from the Location Counter for that section The Location Counter for each successively declared control section assigns locations in consecutively higher areas of storage. Thus, if a program has multiple control sections, all statements identified as belonging to the first control section will be assigned from the Location Counter for section 1, the statements for the second control section will be assigned from the Location Counter for section 2, etc. This procedure is followed whether the statements from different control sections are interspersed or written in control section sequence.

The Location Counter setting can be controlled by using the START and ORG assembler instructions, which are described in Addressing -- Program Sectioning and Linking. The counter affected by either of these assembler instructions is the counter for the control section in which they appear. Maximum value for the Location Counter is $2^{24}-1$.

Literals

A literal term is one of three basic ways to introduce data into a program. It is simply a constant preceded by an equal sign (=).

A literal represents data rather than a reference to data. The appearance of a literal in a source statement directs the assembler program to assemble the value specified by the literal, store this value in a "literal pool", and place the address of the storage field containing the value in the operand field of the assembled source statement.

Literals provide a means of entering constants (such as numbers for calculation addresses, indexing factors, or words or phrases for printing out a message) into a program by specifying the constant in the operand of the instruction in which it is used. This is in contrast to using the DC assembler instruction to enter the data into the program, and then using the name of the DC instruction in the operand. Only one literal is allowed in a machine-instruction statement.

A literal term may not be combined with any other terms.

A literal may not be changed in storage. That is, it may not be used as the receiving field of an instruction that modifies storage.

A literal may not be specified in a constant (see DC--Define Constant) or any other Assembler instruction.

The instruction coded below shows one use of a literal.

Name	Operation	Operand
GAMMA	L	10,=F'274'

The statement GAMMA is a load instruction using a literal as the second operand. When assembled, the second operand of the instruction will be the address at which the binary value represented by F'274' is stored.

In general, literals may be used wherever a storage address is permitted as an operand. They may not, however, be used in any assembler instruction that requires the use of a previously defined symbol. Literals are considered relocatable because the address of the literal rather than the literal itself, will be assembled in the statement that employs a literal. The assembler generates the literals, collects them, and places them in a specific area of storage as explained in the subsection The Literal Pool. A literal is not to be confused with the immediate data in an SI instruction. Immediate data is assembled into the instruction.

Literal Format: The assembler requires a description of the type of literal being specified as well as the literal itself. This descriptive information assists the assembler in assembling the literal correctly The descriptive portion of the literal must indicate the format in which the constant is to be assembled. It may also specify the length the constant is to occupy.

The method of describing and specifying a constant as a literal is nearly identical to the method of specifying it in the operand of a DC assembler instruction. The major difference is that the literal must start with an equal sign (=), which indicates to the assembler that a literal follows. The reader is referred to the discussion of the DC assembler instruction operand format Addressing -- Program Sectioning and Linking for the means of specifying a literal. All types of address constants, except S-type address constants, can be expressed as literals. Some examples of literals are:

```
=A(BETA)    --   address constant literal.
=F'1234'    --   a fixed-point number with
                 a length of four bytes.
=C'ABC'     --   a character literal.
=CL7'PAGE'  --   an explicit length literal.
```

The Literal Pool: The literals processed by the assembler are collected and placed in a special area called the literal pool, and the location of the literal, rather than the literal itself, is assembled in the statement employing a literal. The positioning of the literal pool may be controlled by the programmer, if he so desires. Unless otherwise specified, the literal pool is placed at the end of the first control section.

The programmer may also specify that multiple literal pools be created. However, the sequence in which literals are ordered within the pool is controlled by the assembler. Further information on positioning the literal pool(s) is under LTORG--Begin Literal Pool.

Symbol Length Attribute Reference

The length attribute of a symbol may be used as a term. Reference to the attribute is made by coding L' followed by the symbol, as in:

```
L'BETA
```

The length attribute of BETA will be substituted for the term. The following example illustrates the use of L'symbol in moving a character constant into either the high-order or low-order end of a storage field.

For ease in following the example, the length attributes of A1 and B2 are mentioned. However, keep in mind that the L'symbol term makes coding such as this possible in situations where lengths are unknown.

```
|Name   |Operation |Operand                  |
|-------|----------|-------------------------|
|A1     |DS        |CL8                      |
|B2     |DC        |CL2'AB'                  |
|HIORD  |MVC       |A1(L'B2),B2              |
|LOORD  |MVC       |A1+L'A1-L'B2(L'B2),B2    |
```

A1 names a storage field eight bytes in length and is assigned a length attribute of eight. B2 names a character constant two bytes in length and is assigned a length attribute of two. The statement named HIORD moves the contents of B2 into the leftmost two bytes of A1. The term L'B2 in parentheses provides the length specification required by the instruction. When the instruction is assembled, the length is placed in the proper field of the machine instruction.

The statement named LOORD moves the contents of B2 into the right-most two bytes of A1. The combination of terms A1+L'A1-L'B2 results in the addition of the length of A1 to the beginning address of A1, and the subtraction of the length of B2 from this value. The result is the address of the seventh byte in field A1. The constant represented by B2 is moved into A1 starting at this address. L'B2 in parentheses provides length specification as in HIORD.

EXPRESSIONS

The preceding subsection dealt with the various types of terms that can be used, either singly or in combination, to form operand entries. This subsection now deals with the more general category of expressions, where an expression is an operand entry consisting of either a single term or an arithmetic combination of terms.

Up to three terms can be combined with the following arithmetic operators:

+	addition, e.g., ALPHA+2
-	subtraction, e.g., ALPHA-BETA
*	multiplication, e.g., 5*L'DATA
/	division, e.g., (ALPHA-BETA)/2

Two of the terms within a three term expression can be grouped within parentheses to indicate to the Assembler the order in which they are to be evaluated. When the Assembler program encounters terms in parentheses in combination with another term, it first reduces the combination of terms inside the parentheses to a single value. This value then is used in reducing the rest of the expression to another single value.

Certain fixed rules determine the ways in which terms can be combined. These rules are discussed under Absolute and Relocatable Expressions. In addition to these, the following general rules can be stated for coding any expressions:

1. An expression may not start with an arithmetic operator (+-/*). Therefore, the expression -A+BETA is invalid. However, the expression 0-A+BETA is valid.

2. An expression may not contain two terms or two operators in succession.

3. An expression may not consist of more than 3 terms.

4. An expression may not have more than one pair of parentheses.

5. A multiterm expression may not contain a literal.

The following are examples of valid expressions:

```
AREA1+X'2D'   (EXIT-ENTRY)/8 29
*+32          =F'1234'       L'FIELD
N-25          L'BETA*10      TEN/TWO
FIELD         B'101'         LAMBDA+GAMMA
FIELD+332     C'ABC'
```

Evaluation of Expressions

A single term expression, e.g., 29, BETA, *, L'SYMBOL, takes on the value of the term involved.

A multiterm expression, e.g., BETA+10,ENTRY-EXIT, 10+A/B, is reduced to a single value, as follows:

1. Each term is given its value.

2. Expressions within parentheses are evaluated first.

3. Arithmetic operations are performed left to right. Multiplication and division are done before addition and subtraction, e.g., A+B*C is evaluated as A+(B*C), not (A+B)*C. The computed result is the value of the expression.

4. Division yields an integer result; any fractional portion of the result will be dropped. For example, the expression 1/2*10 equals zero, but the expression 10*1/2 equals five.

5. Division by zero is valid and yields a zero result.

Final values of expressions representing storage addresses are never greater than $2^{24}-1$, however intermediate results may have a maximum value of $2^{31}-1$.

Absolute and Relocatable Expressions

An expression is called absolute if its value is unaffected by program relocation

An expression is called relocatable if its value changes upon program relocation

The two types of expressions, absolute and relocatable, take on these characteristics from the term or terms composing them. The following material discusses this relationship.

ABSOLUTE EXPRESSION: An absolute expression may be an absolute term or any arithmetic combination of absolute terms. An absolute term may be an absolute symbol, any of the self-defining terms, or the length attribute reference. As indicated in Figure 2, all arithmetic operations are permitted between absolute terms.

An absolute expression may contain two relocatable terms (RT) -- alone or in combination with an absolute term (AT) -- under the following conditions:

1. The relocatable terms must be paired, that is, they must appear in the same control section in this assembly (see Program Sectioning and Linking) and have opposite signs. The paired terms do not have to be contiguous, e.g., RT+AT-RT.

2. No relocatable term may enter into a multiply or divide operation. Thus, RT-RT*10 is invalid However, (RT-RT)*10 is valid.

The pairing of relocatable terms cancels the effect of relocation. Therefore the value represented by the paired terms remains constant, regardless of program relocation For example, in the absolute expression A-Y+X, A is an absolute term, and X and Y are relocatable terms from the same control section. If A equals 50, Y equals 25, and X equals 10, the value of the expression would be 35. If X and Y are relocated by a factor of 100 their values would then be 125 and 110. However, the expression would still evaluate as 35 (50-125+110=35).

An absolute expression reduces to a single absolute value.

The following examples illustrate absolute expressions. A is an absolute term; X and Y are relocatable terms from the same control section.

```
A-Y+X
A
A*A
X-Y+A
*-Y   (a reference to the Location Counter
       must be paired with another
       relocatable term from the same control
       section).
```

RELOCATABLE EXPRESSIONS: A relocatable expression is one whose value would change by n if the program in which it appears is relocated n bytes away from its originally assigned area of storage. A relocatable expression must not have a value below the starting address of the control section, except in a USING, CCW, or A- and Y-type address constant.

A relocatable expression may be a relocatable term. A relocatable expression may contain relocatable terms -- alone or in combination with absolute terms -- under the following conditions:

1. There must be an odd number of relocatable terms.

2. If a relocatable expression contains three relocatable terms, two of them must be paired. Pairing is described in Absolute Expression.

3. The unpaired term must not be directly preceded by a minus sign.

4. No relocatable term may enter into a multiply or divide operation.

A relocatable expression reduces to a single relocatable value. This value is the value of the odd relocatable term, adjusted by the values represented by the absolute terms and/or paired relocatable terms associated with it.

For example, in the expression W-X+W, W and X are relocatable terms from the same control section. If initially W equals 10 and X equals 5, the value of the expression is 15. However, upon relocation this value will change. If a relocation factor of 100 is applied, the value of the expression is 115. Note that the value of the paired terms, W-X, remains constant at 5 regardless of relocation. Thus, the new value of the expression, 115, is the result of the value of the odd term (W) adjusted by the values of W-X.

The following examples illustrate relocatable expressions. A is an absolute term, W and X are relocatable terms from the same control section. Y is a relocatable term from a different control section.

Y-32*A	W-X+*	=F'1234' (literal)
W-X+Y		A*A+W
* (reference to Location Counter)		W-X+W
		Y

ADDRESSING

The System/360 addressing technique requires the use of a base register, which contains the base address, and a displacement, which is added to the contents of the base register. The programmer may specify a symbolic address and request the assembler to determine its storage address in terms of a base register and a displacement. The programmer may rely on the assembler to perform this service for him by indicating which general registers are available for assignment and what values the assembler may assume each contains. The programmer may use as many or as few registers for this purpose as he desires. The only requirement is that, at the point of reference, a register containing an address from the same control section is available, and that this address is less than or equal to the address of the item to which the reference is being made. The difference between the two addresses may not exceed 4095 bytes.

ADDRESSES -- EXPLICIT AND IMPLIED

An address is composed of a displacement plus the contents of a base register. (In the case of RX instructions the contents of an index register are also used to derive the address.)

The programmer writes an explicit address by specifying the displacement and the base register. If, in an RX instruction, an explicit address is used, it is assembled without being checked for proper boundary alignment. The assembler assumes that the programmer has either used an aligned explicit address or programmed with a register to align the address.

The boundary alignment checked by the assembler is the alignment of the effective address, when an implied address is used. The programmer writes an implied address by specifying an absolute or relocatable address. The assembler can select a base register and compute a displacement, thereby generating an explicit address from an implied address, if the programmer has conveyed the availability and contents of the base registers. He can do this with the USING and DROP instructions.

BASE REGISTER INSTRUCTIONS

The USING and DROP assembler instructions enable programmers to use expressions representing implied addresses as operands of machine-instruction statements, leaving the assignment of base registers and the calculation of displacements to the assembler.

In order to use symbols in the operand field of machine-instruction statements the programmer must (1) indicate to the assembler, by means of a USING statement, that one or more general registers are available for use as base registers, (2) specify by means of the USING statement, what value each base register contains, and (3) load each base register with the value he has specified for it.

A program must have at least one USING statement for each control section to be addressed.

Having the assembler determine base registers and displacements relieves the programmer of separating each address into a displacement value and a base address value. This feature of the assembler will eliminate a likely source of programming errors, thus reducing the time required to check out programs. To take advantage of this feature, the programmer uses the USING and DROP instructions described in this subsection. The principal discussion of this feature follows the description of both instructions.

REGISTER USAGE

Certain general registers have special uses and are available to the programmer under certain restrictions. These registers and the restrictions follow.

0-1 These registers are used by the routines generated from the IBM-supplied macros. Therefore, these registers may be used without restriction <u>if</u> no IBM macros appear in the program; otherwise they should be used only for immediate computations, where the content of the register is no longer needed after the computation. If the programmer uses them, he must either save their content himself (and reload them later) or be finished with them before IOCS uses them.

12-13 These registers are used by the Supervisor Interruption Routine. Since interruptions are unpredictable, these registers should not be used by the programmer, unless SUPVR SAVEREG=YES is specified when assembling the supervisor. (See Macro Instructions to Assemble a Supervisor.)

Note: Whenever Autotest is used, the programmer must not use registers 12 and 13 because these registers are used by the Autotest Master Control routine.

14-15 Logical IOCS uses these two registers for linkage. Register 14 contains the return address (to the problem program) from the DTF routine. Register 15 contains the entry point into the DTF routine. IOCS does not save the contents of these registers prior to using them; if the programmer uses them, he must either save their contents himself (and reload them later) or be finished with them before IOCS uses them. It should be noted that IOCS uses these registers only when the user has called an IOCS routine such as GET, PUT, OPEN, or CLOSE.

Registers 2-11 are available to the programmer. To avoid the possibility of errors, these registers should be the registers used by the programmer. However, if for any reason there is a shortage of registers, 0-1 and 14-15 are available under the restrictions previously stated.

Note: Whenever the Translate and Test (TRT) instruction is used, the contents of register 2 must be saved before this instruction is executed. After the TRT instruction has been executed, the contents of register 2 may be restored. See the System/360 Principles of Operation manual listed on the front cover of this publication for further information on the TRT instruction.

USING -- Use Base Address Registers

The USING instruction indicates that one or more general registers are available for use as base registers. This instruction also states the base address values that the assembler may assume will be in the registers at object time. Note that a USING instruction does not load the registers specified. It is the programmer's responsibility to see that the specified base address values are placed into the registers. Suggested loading methods are described in the subsection Programming with the USING Instruction.

The format of the USING instruction statement is:

Name	Operation	Operand
Blank	USING	From 2-6 expressions of the form v,r1, r2,r3,r4,r5

Operand v must be an absolute or relocatable expression. No literals are permitted. Operand v specifies a value that the assembler can use as a base address. The other operands must be absolute expressions. The operand r1 specifies the general register that can be assumed to contain the base address represented by operand v. Operands r2, r3, r4, and r5 specify registers that can be assumed to contain v+4096, v+8192, v+12288, and v+16384 respectively The values of the operands r1, r2, r3, r4, and r5 must be between 0 and 15. For example, the statement:

Name	Operation	Operand
	USING	*,8,9

tells the assembler it may assume that the current value of the Location Counter will be in general register 8 at object time, and that the current value of the Location Counter, incremented by 4096, will be in general register 9 at object time.

The operands r1, r2, r3, r4, and r5 can be a symbol or an expression provided that the value of the symbol or expression is between 0 and 15.

If the programmer changes the value in a base register currently being used and wishes the assembler to compute displacements from this value, the assembler must be told the new value by means of another USING statement. In the following sequence the assembler first assumes that the value of ALPHA is in register 9. The second statement then causes the assembler to assume that ALPHA+1000 is the value in register 9.

Name	Operation	Operand
	USING	ALPHA,9
	.	
	.	
	USING	ALPHA+1000,9

A USING statement may specify general register 0 as a base register if operand v is a relocatable expression or zero. If general register 0 is specified, it must be operand r1. In this case, the assembler assumes that register 0 contains the value zero. Subsequent registers specified in the same statement are assumed to have the values 4096, 8192, etc. The assembler therefore places all subsequent effective addresses less than 4096 in the displacement field and uses zero for the base register field.

Note: If register 0 is made available by a USING instruction, the program is not relocatable, despite the fact that the value specified by operand v must be relocatable. However, the programmer is able to make the program relocatable at some future time by:

1. Replacing register 0 in the USING statement.
2. Loading the register with a relocatable value.
3. Reassembling the program.

DROP -- Drop Base Register

The DROP instruction specifies a previously available register that may no longer be used as a base register. The format of the DROP instruction statement is as follows:

Name	Operation	Operand
Blank	DROP	Up to 5 absolute expressions of the form r1,r2, r3,r4,r5

The expressions indicate general registers previously named in a USING statement that are now unavailable for base addressing. The following statement for example, prevents the assembler from using registers 7 and 11:

Name	Operation	Operand
	DROP	7,11

It is not necessary to use a DROP statement when the base address in a register is changed by a USING statement; nor are DROP statements needed at the end of the source program.

A register made unavailable by a DROP instruction can be made available again by a subsequent USING instruction.

PROGRAMMING WITH THE USING INSTRUCTION

The USING (and DROP) instructions may be used anywhere in a program, as often as needed, to indicate the general registers that are available for use as base registers and the base address values the assembler may assume each contains at execution time. Whenever an address is specified in a machine-instruction statement, the assembler determines whether there is an available register containing a suitable base address. A register is considered available for a relocatable address if it was loaded with a relocatable value that is in the same control section as the address. A register with an absolute value is available only for absolute addresses. In either case, the base address is considered suitable only if it is less than or equal to the address of the item to which the reference is made. The difference between the two addresses may not exceed 4095 bytes.

In the following sequence, the BALR instruction loads register 2 with the address of the first storage location immediately following. In this case, it is the instruction named FIRST. The USING instruction indicates to the assembler that register 2 contains this location. When employing this method, the USING instruction must immediately follow the BALR instruction. No other USING or load instructions are required if the location named LAST is within 4095 bytes of FIRST.

Name	Operation	Operand
BEGIN	BALR	2,0
	USING	*,2
FIRST	.	
	.	
	.	
LAST	.	
	END	BEGIN

In the following example, the BALR and LM instructions load registers 2-5. The USING instruction indicates to the assembler that these registers are available as base registers for addressing a maximum of 16,384 consecutive bytes of storage, beginning with the location named

HERE. The number of addressable bytes may be increased or decreased by altering the number of registers designated by the USING and LM instructions and the number of address constants specified in the DC instructions.

```
|Name     | Operation      |Operand          |
|---------|----------------|-----------------|
|BEGIN    | BALR           |2,0              |
|         | USING          |HERE,2,3,4,5     |
|HERE     | LM             |3,5,BASEADDR     |
|         | B              |FIRST            |
|BASEADDR | DC             |A (HERE+4096)    |
|         | DC             |A (HERE+8192)    |
|         | DC             |A (HERE+12288)   |
|FIRST    | .              |                 |
|         | .              |                 |
|         | .              |                 |
|LAST     | .              |                 |
|         | END            |BEGIN            |
```

RELATIVE ADDRESSING

Relative addressing is the technique of addressing instructions and data areas by designating their location in relation to the Location Counter or to some symbolic location. This type of addressing is always in bytes, never in bits, words, or instructions. Thus, the expression *+4 specifies an address that is four bytes greater than the current value of the Location Counter. In the sequence of instructions shown in the following example, the location of the CR machine instruction can be expressed in two ways, ALPHA+2 or BETA-4, because all of the mnemonics in the example are for 2-byte instructions in the RR format.

```
|Name  |Operation |Operand          |
|------|----------|-----------------|
|ALPHA |LR        |3,4              |
|      |CR        |4,6              |
|      |BCR       |1,14             |
|BETA  |AR        |2,3              |
```

PROGRAM SECTIONING AND LINKING

It is often convenient, or necessary, to write a large program in sections. The sections may be assembled separately, then combined subsequently into one object program. The assembler provides facilities for creating multisectioned programs and symbolically linking separately assembled programs or program sections. The combined number of control sections and dummy sections may not exceed 32. The combined number of control sections and dummy sections plus the number of unique symbols in EXTRN statements and V-type address constants may not exceed 255. (EXTRN statements are discussed in this section; V-type constants under DC -- Define Constant assembler instruction.) If the same symbol appears in a V-type address constant and in the name field of any other statement, it is counted as two symbols.

Sectioning a program is optional, and many programs can best be written without sectioning them. The programmer writing an unsectioned program need not concern himself with the subsequent discussion of program sections, which are called control sections. He need not employ the CSECT instruction, which is used to identify the control sections of a multisection program. Similarly, he need not concern himself with the discussion of symbolic linkages if his program neither requires a linkage to nor receives a linkage from another program. He may, however, wish to identify the program and/or specify a tentative starting location for it, both of which may be done by using the START instruction. He may also want to employ the dummy section feature obtained by using the DSECT instruction.

Note: Program sectioning and linking is closely related to the specification of base registers for each control section. Sectioning and linking examples are provided under the heading Addressing External Control Sections.

CONTROL SECTIONS

The concept of program sectioning is a consideration at coding time, assembly time, and load time. To the programmer, a program is a logical unit. He may want to divide it into sections called control sections; if so, he writes it in such a way that control passes properly from one section to another regardless of the relative physical position of the sections in storage. A control section is a block of coding that can be relocated, independently of other coding, at load time without altering or impairing the operating logic of the program. It is normally identified by the CSECT instruction. However, if it is desired to specify a tentative starting location, the START instruction may be used to identify the first control section

To the assembler, there is no such thing as a program; instead, there is an assembly, which consists of one or more control sections. (However, the terms assembly and program are often used interchangeably An unsectioned program is treated as a single control section. To the linkage editor, there are no programs,

only control sections that must be fashioned into an object program.

The output of the assembler consists of the assembled control sections and a control dictionary. The control dictionary contains information the linkage editor needs in order to complete cross-referencing between control sections, as it combines them into an object program. The linkage editor can take control sections from various assemblies and combine them properly with the help of the corresponding control dictionaries. Successful combination of separately assembled control sections depends on the techniques used to provide symbolic linkages between the control sections.

Whether the programmer writes an unsectioned program, a multisection program or part of a multisection program, he still knows what eventually will be entered into storage, because he has described storage symbolically. He may not know where each section appears in storage, but he does know what storage contains. There is no constant relationship between control sections. Thus, knowing the location of one control section does not make another control section addressable by relative addressing techniques.

Control Section Location Assignment

Control section contents can be intermixed because the assembler provides a Location Counter for each control section. Locations are assigned to control sections in such a way that the sections are placed in storage consecutively, in the same order as they first occur in the program. Each control section subsequent to the first begins at the next available double-word boundary.

FIRST CONTROL SECTION

The first control section of a program has the following special properties

1. Its tentative loading location may be specified as an absolute value.

2. It normally contains the literals requested in the program, although their positioning can be altered. This is further explained under the discussion of the LTORG assembler instruction.

START -- Start Assembly

The START instruction may be used to give a name to the first (or only) control section of a program. There may be only one START instruction in an assembly. It may also be used to specify a tentative starting location for the program. The format of the START instruction statement is as follows:

Name	Operation	Operand
A symbol or blank	START	A self-defining term or blank

If a symbol names the START instruction the symbol is established as the name of the control section. If not, the control section is considered to be unnamed. All subsequent statements are assembled as part of that control section This continues until a CSECT instruction identifying a different control section or a DSECT instruction is encountered. A CSECT instruction named by the same symbol that names a START instruction is considered to identify the continuation of the control section first identified by the START. Similarly an unnamed CSECT that occurs in a program initiated by an unnamed START is considered to identify the continuation of the unnamed control section

The symbol in the name field is a valid relocatable symbol whose value represents the address of the first byte of the control section. It has a length attribute of one.

The assembler uses the self-defining value specified by the operand as the tentative starting location of the program. This value must be divisible by eight. For example, either of the following statements could be used to assign the name PROG2 to the first control section and to indicate an initial assembly location of 2040:

Name	Operation	Operand
PROG2	START	2040
PROG2	START	X'7F8'

If the operand is omitted, the assembler sets the tentative starting location of the program at zero.

Note: The START instruction may be preceded only by ICTL, ISEQ, REPRO, PUNCH, EJECT, SPACE, TITLE, PRINT, and comments statements, and by macro instructions that generate only these statements.

If the user plans to write his own macro-instruction routines, the START instruction may not be used as an instruction within his macro routine.

CSECT -- Identify Control Section

The CSECT instruction identifies the beginning or the continuation of a control section. The format of the CSECT instruction statement is as follows:

Name	Operation	Operand
A symbol or blank	CSECT	Not used; any operand is treated as a comment

If a symbol names the CSECT instruction the symbol is established as the name of the control section; otherwise the section is considered to be unnamed. All statements following the CSECT are assembled as part of that control section until a statement identifying a different control section is encountered (i.e., another CSECT or a DSECT instruction).

The symbol in the name field is a valid relocatable symbol whose value represents the address of the first byte of the control section. It has a length attribute of one.

Several CSECT statements with the same name may appear within a program. The first is considered to identify the beginning of the control section; the rest identify the resumption of the section. Thus, statements from different control sections may be interspersed. They are properly assembled (assigned contiguous storage locations) as long as the statements from the various control sections are identified by the appropriate CSECT instructions

Unnamed Control Section

If neither a named CSECT instruction nor START instruction appears at the beginning of the program, the assembler determines that it is to assemble an unnamed control section as the first (or only) control section. There may be only one unnamed control section in a program. If one is initiated and is then followed by a named control section, any subsequent unnamed CSECT statements are considered to resume the unnamed control section. If it is desired to write a small program that is unsectioned, the program does not need to contain a CSECT instruction.

DSECT -- Identify Dummy Section

A dummy section represents a control section that is assembled but is not part of the object program. A dummy section is a convenient means of describing the layout of an area of storage without actually reserving the storage. (It is assumed that the storage is reserved either by some other part of this assembly or else by another assembly The DSECT instruction identifies the beginning or resumption of a dummy section. More than one dummy section may be defined per assembly, but each must be named. The format of the DSECT instruction statement is as follows:

Name	Operation	Operand
A symbol	DSECT	Not used; any operand is treated as a comment

The symbol in the name field is a valid relocatable symbol whose value represents the first byte of the dummy section. It has a length attribute of one.

Program statements belonging to dummy sections may be interspersed throughout the program or may be written as a unit. In either case, the appropriate DSECT instruction should precede each set of statements. Whenever the assembler instructions EJECT, SPACE, PRINT, PUNCH, REPRO, XFR, or TITLE are used within a DSECT, they are treated as comments and not executed. When multiple DSECT instructions with the same name are encountered the first is considered to initiate the dummy section and the rest to continue it.

Symbols that appear in the name field of a DSECT statement or in the name field of statements in a dummy section may be used in USING instructions. Therefore, they may be used in program elements (e.g., machine-instructions and data definitions) that specify storage addresses. An example illustrating the use of a dummy section appears subsequently under Addressing Dummy Sections.

A symbol that names a statement in a dummy section may be used in an A-type address constant only if it is paired with another symbol (with the opposite sign) from the same dummy section.

DUMMY SECTION LOCATION ASSIGNMENT: A Location Counter is used to determine the relative locations of named program elements in a dummy section. The Location Counter is always set to zero at the beginning of the dummy section, and the location values assigned to symbols that name statements in the dummy section are relative to the initial statement in the section.

ADDRESSING DUMMY SECTIONS: The programmer may wish to describe the format of an area whose storage location will not be

determined until the program is executed. He can describe the format of the area in a dummy section, and he can use symbols defined in the dummy section as the operands of machine instructions. To effect references to the storage area, he does the following:

1. Provides a USING statement specifying both a general register that the assembler can assign to the machine-instructions as a base register and a value from the dummy section that the assembler may assume the register contains

2. Ensures that the same register is loaded with the actual address of the storage area.

The values assigned to symbols defined in a dummy section are relative to the initial statement of the section. Thus, all machine-instructions which refer to names defined in the dummy section will, at execution time, refer to storage locations relative to the address loaded into the register.

An example is shown in the following coding. Assume that two independent assemblies (assembly 1 and assembly 2) have been loaded and are to be executed as a single overall program. Assembly 1 is an input routine that places a record in a specified area of storage, places the address of the input area containing the record in general register 3, and branches to assembly 2. Assembly 2 processes the record. Coding shown in the example is from assembly 2.

The input area is described in assembly 2 by the DSECT control section named INAREA. Portions of the input area (i.e., record) that the programmer wishes to work with are named in the DSECT control section as shown. The assembler instruction USING INAREA,3 designates general register 3 as the base register to be used in addressing the DSECT control section, and that general register 3 is assumed to contain the address of INAREA.

Assembly 1, during execution, loads the actual beginning address of the input area in general register 3. Because the symbols used in the DSECT section are defined relative to the initial statement in the section, the address values they represent, will, at the time of program execution, be the actual storage locations of the input area.

```
|Name     |Operation  |Operand              |
|---------|-----------|---------------------|
|ASMBLY2  |CSECT      |                     |
|BEGIN    |BALR       |2,0                  |
|         |USING      |*,2                  |
|         |  .        |                     |
|         |  .        |                     |
|         |USING      |INAREA,3             |
|         |CLI        |INCODE,C'A'          |
|         |BE         |ATYPE                |
|         |  .        |                     |
|         |  .        |                     |
|ATYPE    |MVC        |WORKA,INPUTA         |
|         |MVC        |WORKB,INPUTB         |
|         |  .        |                     |
|         |  .        |                     |
|WORKA    |DS         |CL20                 |
|WORKB    |DS         |CL18                 |
|         |  .        |                     |
|         |  .        |                     |
|INAREA   |DSECT      |                     |
|INCODE   |DS         |CL1                  |
|INPUTA   |DS         |CL20                 |
|INPUTB   |DS         |CL18                 |
|         |  .        |                     |
|         |END        |                     |
```

SYMBOLIC LINKAGES

Symbols may be defined in one program and referred to in another, thus effecting symbolic linkages between independently assembled programs. The linkages can be effected only if the assembler is able to provide information about the linkage symbols to the linkage editor, which resolves these linkage references at load time. The assembler places the necessary information in the control dictionary on the basis of the linkage symbols identified by the ENTRY and EXTRN instructions. Note that these symbolic linkages are described as linkages between independent assemblies; more specifically, they are linkages between independently assembled control sections.

In the program where the linkage symbol is defined (i.e., used as a name), it must also be identified to the assembler by means of the ENTRY assembler instruction. It is identified as a symbol that names an entry point, which means that another program will use that symbol in order to effect a branch operation or a data reference. The assembler places this information in the control dictionary.

Similarly, the program that uses a symbol defined in some other program must identify it by the EXTRN assembler instruction. It is identified as an externally defined symbol (i.e., defined in another program) that is used to effect linkage to the point of definition The

assembler places this information in the control dictionary.

There is another way to obtain symbolic linkage, namely by using the V-type address constant. The subsection Data Definition Instructions contains the details pertinent to writing a V-type address constant. It is sufficient here to note that this constant may be considered an indirect linkage point. It is created from an externally defined symbol, but that symbol does not have to be identified by an EXTRN statement The V-type address constant may be used for external branch references (i.e., for effecting branches to other programs). It should not be used for external data references (i.e., for referring to data in other programs).

ENTRY -- IDENTIFY ENTRY-POINT SYMBOL

The ENTRY instruction identifies a linkage symbol that is defined in this program but may be used by some other program. The format of the ENTRY instruction statement is as follows:

Name	Operation	Operand
Blank	ENTRY	A relocatable symbol that also appears as a statement name

The symbol in the ENTRY operand field may be used as an operand by other programs. An ENTRY statement operand may not contain a symbol defined in an unnamed control section or a dummy section. The following example identifies the statements named SINE and COSINE as entry points to the program.

Name	Operation	Operand
	ENTRY	SINE
	ENTRY	COSINE

The name of a control section does not have to be identified by an ENTRY instruction when another program uses it as an entry point. The assembler automatically places information on control section names in the control dictionary. A maximum of 100 ENTRY statements will be processed in a single assembly.

EXTRN -- IDENTIFY EXTERNAL SYMBOL

The EXTRN instruction identifies a linkage symbol that is used by this program but defined in some other program Each external symbol must be identified; this includes symbols that name control sections. The format of the EXTRN instruction statement is as follows:

Name	Operation	Operand
Blank	EXTRN	A relocatable symbol

The symbol in the operand field may not appear as the name of a statement in this program. The following example identifies three external symbols that have been used as operands in this program but are defined in some other program.

Name	Operation	Operand
	EXTRN	RATEBI
	EXTRN	PAYCALC
	EXTRN	WITHCALC

An example that employs the EXTRN instruction appears subsequently under Addressing External Control Sections.

Note: A V-type address constant does not have to be defined by an EXTRN statement.

Note: When external symbols are used in an expression they may not be paired. The assembler processes them as though they originated from different control sections.

Addressing External Control Sections

A common way for a program to link to an external control section is to:

1. Create a V-type address constant with the name of the external symbol.
2. Load the constant into a general register and branch to the control section via the register.

```
Name      Operation  Operand
MAINPROG  CSECT
BEGIN     BALR       2,0
          USING      *,2
            .
            .
          L          3,VCON
          BALR       1,3
            .
            .
VCON      DC         V(SINE)
          END        BEGIN
```

For example, to link to the control section named SINE, the preceding coding might be used.

An external symbol naming data may be referred to as follows:

1. Identify the external symbol with the EXTRN instruction, and create an address constant from the symbol.

2. Load the constant into a general register, and use the register for base addressing.

For example, to use an area named RATETBL, which is in another control section, the following coding might be used:

```
Name        Operation   Operand
MAINPROG    CSECT
BEGIN       BALR        2,0
            USING       *,2
              .
              .
            EXTRN       RATETBL
              .
              .
            L           4,RATEADDR
            USING       RATETBL,4
            A           3,RATETBL
              .
              .
RATEADDR    DC          A(RATETBL)
            END         BEGIN
```

This section discusses the coding of the machine-instructions represented in the assembler language. The reader is reminded that the functions of each machine-instruction are discussed in the Principles of Operation manual (see Preface).

MACHINE-INSTRUCTION STATEMENTS

Machine-instructions may be represented symbolically as assembler language statements. The symbolic format of each varies according to the actual machine-instruction format, of which there are five: RR, RX, RS, SI, and SS. Within each basic format, further variations are possible.

The symbolic format of a machine-instruction is similar to, but does not duplicate, its actual format. Appendix D illustrates machine format for the five classes of instructions. A mnemonic operation code is written in the operation field, and one or more operands are written in the operand field. Comments may be appended to a machine-instruction statement as previously explained in the Introduction.

Any machine-instruction statement may be named by a symbol, which other assembler statements can use as an operand. The value attribute of the symbol is the address of the leftmost byte assigned to the assembled instruction. The length attribute of the symbol depends on the basic instruction format, as follows:

Basic Format	Length Attribute
RR	2
RX	4
RS	4
SI	4
SS	6

Instruction Alignment and Checking

All machine-instructions are aligned automatically by the assembler on half-word boundaries. If any statement that causes information to be assembled requires alignment, the bytes skipped are filled with hexadecimal zeros. All expressions that specify storage addresses are checked to insure that they refer to appropriate boundaries for the instructions in which they are used. Register numbers are also checked to make sure that they specify the proper registers as follows:

1. Floating-point instructions must specify floating-point registers 0, 2, 4, or 6.

2. Double-shift, full-word multiply, and full-word divide instructions must specify an even-numbered general register in the first operand.

OPERAND FIELDS AND SUBFIELDS

Some symbolic operands are written as a single field and other operands are written as a field followed by one or two subfields. For example, addresses consist of the contents of a base register and a displacement. An operand that specifies a base and displacement is written as a displacement field followed by a base register subfield, as follows: 40(5). In the RX format, both an index register subfield and a base register subfield are written as follows: 40(3,5). In the SS format, both a length subfield and a base register subfield are written as follows: 40(21,5).

Appendix D shows two types of addressing formats for RX, RS, SI, and SS instructions In each case, the first type shows the method of specifying an address explicitly, as a base register and displacement. The second type indicates how to specify an implied address as an expression.

For example, a load multiple instruction (RS format) may have either of the following symbolic operands:

```
R1,R3,D2(B2)  - -  explicit address
R1,R3,S2      - -  implied address
```

Whereas D2 and B2 must be represented by absolute expressions, S2 may be represented either by a relocatable or an absolute expression.

In order to use implied addresses, the following rules must be observed:

1. The base register assembler instructions (USING and DROP) must be used.

2. An explicit base register designation must not accompany the implied address.

For example, assume that FIELD is a relocatable symbol, which has been assigned a value of 7400. Assume also that the

assembler has been notified (by a USING instruction) that general register 8 currently contains a relocatable value of 4096 and is available as a base register. The following example shows a machine-instruction statement as it would be written in assembler language and as it would be assembled. Note that the value of D2 is the difference between 7400 and 4096 and that X2 is assembled as zero, since it was omitted. The assembled instruction is presented in decimal:

Assembler statement:

```
ST      4,FIELD
```

Assembled instruction:

```
Op.Code  R1  X2  B2   D2
50       4   0    8   3304
```

An address may be specified explicitly as a base register and displacement (and index register for RX instructions) by the formats shown in the first column of the following table. The address may be specified as an implied address by the formats shown in the second column. Observe that the two storage addresses required by the SS instructions are presented separately; an implied address may be used for one while an explicit address is used for the other.

Type	Explicit Address	Implied Address
RX	D2(X2,B2)	S2(X2)
	D2(0,B2)*	S2
RS	D2(B2)	S2
SI	D1(B1)	S1
SS	D1(L1,B1)	S1(L1)
	D1(L,B1)	S1(L)
	D2(L2,B2)	S2(L2)

*A zero must be supplied in an RX explicit address when it is desired to omit an index register specification but include a base register specification.

A comma must be written to separate operands. Parentheses must be written to enclose either one or two subfields of an operand, and two subfields must be separated by a comma. If the format of an operand includes one subfield and if the subfield is omitted, the parentheses must also be omitted. If the format includes two subfields, the following rules apply:

1. If both subfields are omitted, the separating comma and the parentheses must also be omitted.

```
L     2,48(4,5)
L     2,FIELD       (implied address)
```

2. If the first subfield in the sequence is omitted, the comma that separates it from the second subfield is written. The parentheses must also be written.

```
MVC   32(16,5),FIELD2
MVC   BETA(,5),FIELD2 (implied length)
```
(Beta must be an absolute expression)

3. In the RX class of instructions if the index register subfield is not used, but the base register is specified, the first subfield (index register) must be specified as zero. It may not be omitted.

```
L     2,48(4,5)
L     2,48(0,5)
```

4. If the second subfield in the sequence is omitted, the comma that separates it from the first subfield must be omitted. The parentheses must be written

```
MVC   32(16,5),FIELD2
MVC   FIELD1(16),FIELD2       (implied
                               address)
```

Fields and subfields in a symbolic operand may be represented either by absolute or by relocatable expressions, depending on what the field requires. (An expression has been defined as consisting of one term or a series of arithmetically combined terms.) Refer to Appendix D for a detailed description of field requirements.

Note: Blanks may not appear in an operand unless provided by a character self-defining term or a character literal Thus, blanks may not intervene between fields and the comma separators, between parentheses and fields, etc.

LENGTHS -- EXPLICIT AND IMPLIED

The length field in SS instructions can be explicit or implied. To imply a length, the programmer omits a length field from the operand. The omission indicates that the length field is either of the following:

1. The length attribute of the expression specifying the displacement, if an explicit base and displacement have been written.

2. The length attribute of the expression specifying the effective address, if the base and displacement have been implied.

In either case, the length attribute for an expression is the length of the leftmost term in the expression.

By contrast, an explicit length is written by the programmer in the operand as an absolute expression. The explicit length overrides any implied length.

Whether the length is explicit or implied, it is always an effective length. The value inserted into the length field of the assembled instruction is one less than the effective length in the machine-instruction statement

Note: If a length field of zero is desired, the length may be stated either as a one or as a zero. (This is useful when the subject instruction is to be executed by the Execute (EX) machine instruction.)

To summarize, the length required in an SS instruction may be specified explicitly by the formats shown in the first column of the following table or may be implied by the formats shown in the second column. Observe that the two lengths required in one of the SS instruction formats are presented separately. An implied length may be used for one while an explicit length is used for the other.

Explicit Length	Implied Length
D1(L1,B1)	D1(,B1)
S1(L1)	S1
D1(L,B1)	D1(,B1)
S1(L)	S1
D2(L2,B2)	D2(,B2)
S2(L2)	S2

MACHINE-INSTRUCTION MNEMONIC CODES

The mnemonic operation codes (shown in Appendix B) are designed to be easily remembered codes that indicate the functions of the instructions The normal format of the code is shown below; the items in brackets are not necessarily present in all codes:

Verb [Modifier] [Data Type] [Machine Format]

The verb, which is usually one or two characters, specifies the function. For example, A represents Add , and MV represents Move. The function may be further defined by a modifier. For example, the modifier L indicates a logical function, as in AL for Add Logical.

Mnemonic codes for functions involving data usually indicate the data types, by letters that correspond to those for the data types in the DC assembler instruction (see Assembler Instruction Statements). Furthermore, letters U and W have been added to indicate short and long, unnormalized floating-point operations, respectively For example, AE indicates Add Normalized Short, whereas AU indicates Add Unnormalized Short. Where applicable, full-word fixed-point data is implied if the data type is omitted.

The letters R and I are added to the codes to indicate, respectively, RR and SI machine instruction formats. Thus, AER indicates Add Normalized Short in the RR format. Functions involving character and decimal data types imply the SS format.

MACHINE-INSTRUCTION EXAMPLES

The examples that follow are grouped according to machine-instruction format. They illustrate the various symbolic operand formats. All symbols employed in the examples must be assumed to be defined elsewhere in the same assembly. All symbols that specify register numbers and lengths must be assumed to be equated elsewhere to absolute values.

Implied addressing, control section addressing, and the function of the USING assembler instruction are not considered here. For discussion of these considerations and for examples of coding sequences that illustrate them, the reader is referred to Program Sectioning and Linking and Base Register Instructions.

RR Format

Name	Operation	Operand
ALPHA1	LR	1,2
ALPHA2	LR	REG1,REG2
BETA	SPM	6
GAMMA1	SVC	250
GAMMA2	SVC	TEN

The operands of ALPHA1, BETA, and GAMMA1 are decimal self-defining values, which are categorized as absolute expressions. The operands of ALPHA2 and GAMMA2 are symbols that are equated elsewhere to absolute values.

RX Format

```
|Name     |Operation |Operand              |
|---------|----------|---------------------|
|ALPHA1   |L         |1,39(4,10)           |
|ALPHA2   |L         |REG1,39(4,TEN)       |
|BETA1    |L         |2,ZETA(4)            |
|BETA2    |L         |REG2,ZETA(REG4)      |
|GAMMA1   |L         |2,ZETA               |
|GAMMA2   |L         |REG2,ZETA            |
|GAMMA3   |L         |2,=F'1000'           |
|LAMBDA1  |L         |3,20(0,5)            |
```

Both ALPHA instructions specify explicit addresses; REG1 and TEN are absolute symbols. Both BETA instructions specify implied addresses, and both use index registers. Indexing is omitted from the GAMMA instructions. GAMMA1 and GAMMA2 specify implied addresses. The second operand of GAMMA3 is a literal. LAMBDA1 specifies no indexing.

RS Format

```
|Name     |Operation |Operand              |
|---------|----------|---------------------|
|ALPHA1   |BXH       |1,2,20(4)            |
|ALPHA2   |BXH       |REG1,REG2,20(REGD)   |
|ALPHA3   |BXH       |REG1,REG2,ZETA       |
|ALPHA4   |SLL       |REG2,5               |
|ALPHA5   |SLL       |REG2,0(5)            |
```

Whereas ALPHA1 and ALPHA2 specify explicit addresses, ALPHA3 specifies an implied address. ALPHA4 is a shift instruction shifting the contents of REG2 left 5 bit positions. ALPHA5 is a shift instruction shifting the contents of REG2 left by the value contained in general register 5.

SI Format

```
|Name     |Operation |Operand              |
|---------|----------|---------------------|
|ALPHA1   |CLI       |40(9),X'40'          |
|ALPHA2   |CLI       |40(REG9),TEN         |
|BETA1    |CLI       |ZETA,TEN             |
|BETA2    |CLI       |ZETA,C'A'            |
|GAMMA1   |SIO       |40(9)                |
|GAMMA2   |SIO       |0(9)                 |
|GAMMA3   |SIO       |40(0)                |
|GAMMA4   |SIO       |ZETA                 |
```

The ALPHA instructions and GAMMA1-GAMMA3 specify explicit addresses, whereas the BETA instructions and GAMMA4 specify implied addresses. GAMMA2 specifies a displacement of zero. GAMMA3 does not specify a base register.

SS Format

```
|Name     |Operation |Operand                  |
|---------|----------|-------------------------|
|ALPHA1   |AP        |40(9,8),30(6,7)          |
|ALPHA2   |AP        |40(NINE,REG8),30(L6,7)   |
|ALPHA3   |AP        |FIELD2,FIELD1            |
|ALPHA4   |AP        |FIELD2(9),FIELD1(6)      |
|BETA     |AP        |FIELD2(9),FIELD1         |
|GAMMA1   |MVC       |40(9,8),30(7)            |
|GAMMA2   |MVC       |40(NINE,REG8),DEC(7)     |
|GAMMA3   |MVC       |FIELD2,FIELD1            |
|GAMMA4   |MVC       |FIELD2(9),FIELD1         |
```

ALPHA1, ALPHA2, GAMMA1, and GAMMA2 specify explicit lengths and addresses. ALPHA3 and GAMMA3 specify both implied length and implied addresses. ALPHA4 and GAMMA4 specify explicit length and implied addresses. BETA specifies an explicit length for FIELD2 and an implied length for FIELD1; both addresses are implied.

EXTENDED MNEMONIC CODES

For the convenience of the programmer, the assembler provides extended mnemonic codes, which allow conditional branches to be specified mnemonically as well as through the use of the BC machine-instruction. These extended mnemonic codes specify both the machine branch instruction and the condition on which the branch is to occur. The codes are not part of the universal set of machine-instructions, but are translated by the assembler into the corresponding operation and condition combinations.

The allowable extended mnemonic codes and their operand formats are shown in Figure 3, together with their machine-instruction equivalents. Unless otherwise noted, all extended mnemonics shown are for instructions in the RX format. Note that the only difference between the operand fields of the extended mnemonics and those of their machine-instruction equivalents is the absence of the R1 field and the comma that separates it from the rest of the operand field. The extended mnemonic list, like the machine-instruction list, shows explicit address formats only. Each address can also be specified as an implied address.

EXTENDED CODE	OPERAND	MEANING	MACHINE INSTRUCTION
B	D2 (X2, B2)	Branch Unconditional	BC 15, D2 (X2, B2)
BR	R2	Branch Unconditional (RR Format)	BCR 15, R2
NOP	D2 (X2, B2)	No Operation	BC 0, D2 (X2, B2)
NOPR	R2	No Operation (RR Format)	BCR 0, R2
		USED AFTER COMPARE INSTRUCTIONS	
BH	D2 (X2, B2)	Branch on High	BC 2, D2 (X2, B2)
BL	D2 (X2, B2)	Branch on Low	BC 4, D2 (X2, B2)
BE	D2 (X2, B2)	Branch on Equal	BC 8, D2 (X2, B2)
BNH	D2 (X2, B2)	Branch on Not High	BC 13, D2 (X2, B2)
BNL	D2 (X2, B2)	Branch on Not Low	BC 11, D2 (X2, B2)
BNE	D2 (X2, B2)	Branch on Not Equal	BC 7, D2 (X2, B2)
		USED AFTER ARITHMETIC INSTRUCTIONS	
BO	D2 (X2, B2)	Branch on Overflow	BC 1, D2 (X2, B2)
BP	D2 (X2, B2)	Branch on Plus	BC 2, D2 (X2, B2)
BM	D2 (X2, B2)	Branch on Minus	BC 4, D2 (X2, B2)
BZ	D2 (X2, B2)	Branch on Zero	BC 8, D2 (X2, B2)
		USED AFTER TEST UNDER MASK INSTRUCTION	
BO	D2 (X2, B2)	Branch if Ones	BC 1, D2 (X2, B2)
BM	D2 (X2, B2)	Branch if Mixed	BC 4, D2 (X2, B2)
BZ	D2 (X2, B2)	Branch if Zeros	BC 8, D2 (X2, B2)

Figure 3. Extended Mnemonic Codes

The following examples illustrate instructions using extended mnemcnic codes. In these examples it is assumed that the symbol GO is defined elsewhere in the program.

Name	Operation	Operand
	B	40(3,6)
	B	40(0,6)
	BL	GO(3)
	BL	GO
	BR	4

The first two instructions specify an unconditional branch to an explicit address The address in the first case is the sum of the contents of base register 6, the contents of index register 3, and the displacement 40; the address in the second instruction is not indexed. The third instruction specifies a branch on low to the address implied by GO as indexed by the contents of index register 3; the fourth instruction does not specify an index register. The last instruction is an unconditional branch to the address contained in register 4.

Just as machine instructions are used to request the computer to perform a sequence of operations during program execution time, so assembler instructions are requests to the assembler to perform certain operations during the assembly. Assembler-instruction statements, in contrast to machine-instruction statements do not always cause machine-instructions to be included in the assembled program. Some, such as DS and DC, generate no instructions but do cause storage areas to be set aside for constants and other data. Others, such as EQU and SPACE, are effective only at assembly time; they generate nothing in the assembled program and have no effect on the Location Counter.

The following is a list of all the assembler instructions.

Symbol Definition Instruction
EQU - Equate Symbol

Data Definition Instructions
DC - Define Constant
DS - Define Storage
CCW - Define Channel Command Word

* Program Sectioning and Linking Instructions
START - Start Assembly
CSECT - Identify Control Section
DSECT - Identify Dummy Section
ENTRY - Identify Entry-Point Symbol
EXTRN - Identify External Symbol

* Base Register Instructions
USING - Use Base Address Register
DROP - Drop Base Address Register

Listing Control Instructions
TITLE - Identify Assembly Output
EJECT - Start New Page
SPACE - Space Listing
PRINT - Print Optional Data

Program Control Instructions
ICTL - Input Format Control
ISEQ - Input Sequence Checking
ORG - Set Location Counter
LTORG - Begin Literal Pool
CNOP - Conditional No Operation
END - End Assembly
REPRO - Reproduce Following Card
PUNCH - Punch a Card
XFR - Generate a Transfer Card

* Discussed under Addressing -- Program Sectioning and Linking.

SYMBOL DEFINITION INSTRUCTION

EQU -- EQUATE SYMBOL

The EQU instruction is used to define a symbol by assigning to it the attributes of an expression in the operand field. The format of the EQU instruction statement is as follows:

Name	Operation	Operand
A symbol	EQU	An expression

The expression in the operand field must be absolute or relocatable. Any symbols appearing in the expression must be previously defined.

The symbol in the name field is given the same attributes as the expression in the operand field. The length attribute of the symbol is that of the leftmost (or only) term of the expression. The value attribute of the symbol is the value of the expression.

The EQU instruction is the means of equating symbols to register numbers, immediate data, and other arbitrary values The following examples illustrate how this might be done:

Name	Operation	Operand
REG2	EQU	2 (general register)
TEST	EQU	X'3F' (immediate data)

To reduce programming time, the programmer can equate symbols to frequently used expressions and then use the symbols as operands in place of the expressions Thus, in the statement

Name	Operation	Operand
FIELD	EQU	ALPHA-BETA+GAMMA

FIELD is defined as ALPHA-BETA+GAMMA and may be used in place of it. Note, however,

that ALPHA, BETA, and GAMMA must all be previously defined.

DATA DEFINITION INSTRUCTIONS

There are three data definition instruction statements: Define Constant (DC), Define Storage (DS), and Define Channel Command Word (CCW).

These statements are used to enter data constants into storage, to define and reserve areas of storage, and to specify the contents of channel command words. The statements may be named by symbols so that other program statements can refer to the fields generated from them. The DS instruction is written in the same format as the DC instruction and may specify some or all of the information that the DC instruction provides. Only the function and treatment of the statements vary. The DC instruction is presented first and discussed in more detail than the DS instruction.

DC -- DEFINE CONSTANT

The DC instruction is used to provide constant data in storage. It may specify one constant or a series of constants. Furthermore, a variety of constants may be specified: binary fixed-point, floating-point, decimal, hexadecimal, character, and storage addresses (Data constants are generally called constants unless they are created from storage addresses, in which case they are called address constants.)

The format of the DC instruction statement is as follows:

Name	Operation	Operand
A symbol or blank	DC	One operand in the following format

The operand consists of four subfields. They are written in the following sequence:

1	2	3	4
Dupli-cation Factor	Type	Modifiers	Constant(s)

The constant provided in the fourth subfield is described by subfields 1-3. Some or all of the three descriptive subfields may be omitted, depending on the constant. Note that more than one constant may be specified in the fourth subfield for most types of constants, so the programmer need not write a separate data definition for every constant desired. However, each constant so specified must be of the same type; the descriptive three subfields apply to all of them. No blanks may occur within any subfield, except in providing a character in a character constant. No blanks may occur between the subfields of an operand.

The symbol that names the DC instruction is the name of the constant, or, if the instruction specifies more than one, the first constant. In the case of multiple constants, relative addressing (for example, SYMBOL+2) may be used to reach the various values. The number of bytes allocated to each constant can readily be determined from the four subfields.

The value attributed to the symbol naming the DC instruction is the address of the leftmost byte (after alignment) of the constant, or the first constant where multiple constants are defined. The length attribute depends on two things: the type of constant being defined and the use of a length specification. If no length specification is present, the implied length of the constant is assumed. Should more than one constant be defined, the attributed length is the length in bytes (specified or implied) of the first constant.

Boundary alignment also varies according to the type of constant being specified and the presence of a length specification. Some constant types are only aligned to a byte boundary, but the DS instruction can be used to force any type of word boundary alignment for them. This is explained under DS -- Define Storage. Other constants are aligned at various word boundaries (half, full, or double) in the absence of a length specification. For these constants, no boundary alignment occurs if length is indicated.

Bytes that must be skipped in order to align the field at the proper boundary are not considered to be part of the constant. In other words, the Location Counter is incremented to reflect the proper boundary (if any incrementing is necessary) before the address value is established. Thus, the symbol naming the constant will not receive a value attribute that is the location of a skipped byte. The bytes skipped in aligning the constant defined by a DC instruction will be zeroed, because information is being assembled. This would occur, for example, in the alignment of the statement DC F'123'.

Appendix F summarizes, in chart form, the information concerning constants that is presented in this section.

LITERAL DEFINITIONS: The reader is reminded that the discussion of literals as machine-instruction operands referred him to the description of the DC operand for the method of writing a literal operand. All subsequent operand specifications are applicable to writing literals, the only differences being:

1. The literal is preceded by an = sign.
2. The duplication factor may not be zero.
3. S-type address constants may not be specified.

Examples of literals appear throughout the balance of the DC instruction discussion.

Operand Subfield 1: Duplication Factor

The duplication factor must be specified by an unsigned decimal value. It causes the constant(s) to be generated the number of times indicated by the factor. It is applied after the constant has been fully assembled, that is, after it has been developed into its proper format. Note that, except in a literal, a duplication factor of zero is permitted. It is used to force alignment to a doubleword, full-word, or half-word boundary, as desired. (See Forcing Alignment under DS--Define Storage.) The duplication factor may be omitted altogether.

Note: If duplication is specified for an address constant containing a Location Counter reference, the value of the Location Counter used in each duplication is incremented by the length of the constant.

Operand Subfield 2: Type

The type subfield defines the type of constant being specified. From the type specification, the assembler determines how it is to interpret the constant and translate it into the appropriate machine format. The type is specified by a letter code as shown in Figure 4.

Further information about these constants is provided in the discussion of the constants themselves under Operand Subfield 4: Constant.

Operand Subfield 3: Modifiers

Modifiers describe the length in bytes desired for a constant (in contrast to an implied length), and the scaling and exponent for the constant. If multiple modifiers are written, they must appear in

CODE	TYPE OF CONSTANT	MACHINE FORMAT
C	Character	8-bit code for each character
X	Hexadecimal	4-bit code for each hexadecimal digit
B	Binary	Binary format
F	Fixed-point	Signed, fixed-point binary format; normally a full word
H	Fixed-point	Signed, fixed-point binary format; normally a half word
E	Floating-point	Short floating-point format; normally a full word
D	Floating-point	Long floating-point format; normally a double word
P	Decimal	Packed decimal format
Z	Decimal	Zoned decimal format
A	Address	Value of address; normally a full word
Y	Address	Value of address; normally a half word
S	Address	Base register and displacement value; a half word
V	Address	Space reserved for external symbol address; each address normally a full word

Note: The type subfield for a character constant may be omitted, C is assumed.

Figure 4. Type Codes for Constants

this sequence: length, scale, exponent. Each is written and used as described in the following text.

LENGTH MODIFIER: A length modifier may be specified for any type of constant. It is written as Ln, where n is an unsigned decimal value. The value of n represents the number of bytes of storage that are assembled for the constant. No boundary alignment is provided when a length modifier is given.

The maximum value permitted for a length modifier varies with the type of constant. A character may have a length up to 256 specified, whereas the range of a fixed-point constant is 1 to 8. Should the specified length be greater or less than the constant actually given, padding or truncation occurs as necessary. If no length modifier is present, the implied length of that type of constant is used. Limits on the modifiers and implied lengths are found in Appendix F, a summary of constants.

SCALE MODIFIER: This modifier is written as Sn, where n is a decimal value. The decimal value may be preceded by a sign; if none is present, a plus sign is assumed. The maximum values for scale modifiers are summarized in Appendix F.

A scale modifier may be used with fixed-point (F, H) and floating-point (E, D) constants only. It is used to specify the amount of internal scaling that is desired, as follows.

Scale Modifier for Fixed-Point Constant: This scale modifier specifies the power of two by which the constant must be multiplied after it has been converted to its binary representation. It must fall within the range -187 to +346. Multiplication of a binary number by a power of two causes the binary point to move. It has the effect of shifting the binary point away from its assumed position in the binary field, the assumed position being to the right of the rightmost position. The process is comparable to the movement of a decimal point in the multiplication of a decimal number by a power of ten.

The scale modifier, then, indicates either of the following:

1. the number of binary positions to be occupied by the fractional portion of the binary number, or

2. the number of binary positions to be deleted from the integral portion of the binary number.

A positive scale of x shifts the integral portion of the number x binary positions to the left, thereby reserving the rightmost x binary positions for the fractional portion. A negative scale shifts the integral portion of the number right, thereby deleting rightmost integral positions. If a scale modifier does not accompany a fixed-point constant containing a fractional part, the fractional part is lost. For example, if the decimal portion of the number 987.65 is to be retained, the statement DC FS8'987.65' is required. The statement DC F'987.65' without the scale factor S8 retains only the integral portion, 987, of the number.

In all cases where positions are lost because of scaling (or the lack of scaling), rounding occurs in the leftmost bit of the lost portion. The rounding is reflected in the rightmost position saved.

Scale Modifier for Floating-Point Constant: Only a positive scale modifier may be used with a floating-point constant. It may be any value from 0 to 13 and indicates the number of hexadecimal positions that the fraction is to be shifted to the right. It is specified in this way because a floating-point constant is always converted to a fraction. This fraction is hexadecimally normalized, that is, with a high-order hexadecimal digit that is not zero. Because the point is then assumed to be to the left of the leftmost position in the field, and cannot be moved left, the fraction is shifted right. Note that this shift amount is in terms of hexadecimal positions.

Thus, scaling that is specified for a floating-point constant provides an assembled fraction that is unnormalized, i.e., contains hexadecimal zeros in the leftmost positions of the fraction. When the fraction is shifted, the exponent is adjusted accordingly to retain the correct magnitude. When hexadecimal positions are lost, rounding occurs in the leftmost hexadecimal position of the lost portion. The rounding is reflected in the rightmost hexadecimal position saved.

EXPONENT MODIFIER: This modifier is written as En, where n is a decimal value. The decimal value may be preceded by a sign; if none is present, a plus sign is assumed. The maximum values for exponent modifiers are summarized in Appendix F.

An exponent modifier may be used with fixed-point (F, H) and floating-point (E, D) constants only. The modifier denotes the power of 10 by which the constant is to be multiplied before its conversion to the proper internal format.

This modifier is not to be confused with the exponent of the constant itself, which is specified as part of the constant and is explained under Operand Subfield 4: Constant. Both are denoted in the same fashion, as En. The exponent modifier affects each constant in the operand, whereas the exponent written as part of the constant only pertains to that constant. Thus, a constant may be specified with an exponent of +2, and an exponent modifier of +5 may precede the constant. In effect, the constant has an exponent of +7.

Note that there is a maximum value, both positive and negative, listed in Appendix F for exponents. This applies both to exponent modifier and exponents specified as part of the constant, or to their sum if both are specified.

Operand Subfield 4: Constant

This subfield supplies the constant (or constants) described by the subfields that precede it. A data constant (all types except A,Y,S,and V) is enclosed by single quotation marks. An address constant (types A, Y, S, and V) is enclosed by parentheses. For types F,H,E,D,P, and Z two or more constants may be specified in the subfield. The constants must be separated by commas and the entire sequence of constants must be enclosed by single quotation marks. Thus, the format for specifying the constant(s) is one of the following:

Single Constant	Multiple Constants*
'constant'	'constant,...,constant'
(constant)	

*Permitted for F,H,E,D,P, and Z type constants only.

All constant types except character (C), hexadecimal (X), binary (B), packed decimal (P), and zoned decimal (Z), are aligned on the proper boundary, as shown in Appendix F, unless a length modifier is specified. In the presence of a length modifier, no boundary alignment is performed. If an operand specifies more than one constant, any necessary alignment applies to the first constant only. Thus, for an operand that provides five full-word constants, the first would be aligned on a full-word boundary. The rest, however, would automatically fall on full-word boundaries as well.

The total storage requirement of an operand is the product of the length times the number of constants in the operand times the duplication factor (if present) plus any bytes skipped for boundary alignment of the first constant .

If an address constant contains a Location Counter reference, the Location Counter value that is used is the storage address of the first byte the constant will occupy. If an address constant is specified (and it is a Location Counter reference) with a duplication factor, the constant is duplicated with a varying Location Counter value.

The subsequent text describes each of the constant types and provides examples.

Character Constant --- C: Any of the valid 256 punch combinations may be designated in a character constant. Only one character constant may be specified per operand. Since multiple constants within an operand are separated by commas, an attempt to specify two character constants would result in interpreting the comma separating them as a character.

The maximum length of a character constant is 256 bytes. No boundary alignment is performed. Each character is translated into one byte. If no length modifier is given, the size in bytes of the character constant is equal to the number of characters in the constant. If a length modifier is provided the result varies as follows:

1. If the number of characters in the constant exceeds the specified length, as many rightmost bytes as necessary are dropped.

2. If the number of characters is less than the specified length, the excess rightmost bytes are filled with blanks.

Special consideration must be given to representing quotation marks and ampersands as characters. Each single quotation mark or ampersand desired as a character in the constant must be represented by a pair of single quotation marks or ampersands. The double quotation marks and ampersands count as one character. Only one single quotation mark or ampersand appears in storage.

In the following example, the length attribute of FIELD is 12:

```
|Name   |Operation  |Operand                |
|-------|-----------|-----------------------|
|FIELD  |DC         |C'TOTAL IS 110'        |
```

However, in this next example, the length attribute is 15, and three blanks appear in storage to the right of the zero:

```
|Name   |Operation  |Operand                  |
|-------|-----------|-------------------------|
|FIELD  |DC         |CL15'TOTAL IS 110'       |
```

In the next example, the length attribute of FIELD is 12, although 13 characters appear in the operand. The two ampersands count as only one byte.

```
|Name   |Operation  |Operand                  |
|-------|-----------|-------------------------|
|FIELD  |DC         |C'TOTAL is &&10'         |
|       |           |                         |
```

Note that in the next example, a length of four has been specified, but there are five characters in the constant

```
|Name   |Operation  |Operand                  |
|-------|-----------|-------------------------|
|FIELD  |DC         |3CL4'ABCDE'              |
```

The generated constant would be:

ABCDABCDABCD

On the other hand, if the length had been specified as six instead of four, the generated constant would have been:

ABCDE ABCDE ABCDE

Note that the same constant could be specified as a literal.

```
|Name   |Operation  |Operand                  |
|-------|-----------|-------------------------|
|       |MVC        |AREA(12),=3CL4'ABCDE'    |
```

Hexadecimal Constant -- X: A hexadecimal constant is comprised of one or more of the hexadecimal digits, which are 0-9 and A-F. Only one hexadecimal constant may be specified per statement. The maximum length of a hexadecimal constant is 256 bytes (512 hexadecimal digits). No word boundary alignment is performed.

Constants that contain an even number of hexadecimal digits are translated as one byte per pair of digits. If an odd number of digits is specified, a hexadecimal zero is paired with the leftmost digit to make up another byte.

If no length modifier is given, the implied length of the constant is half the number of hexadecimal digits in the constant (assuming that a hexadecimal zero is added to an odd number of digits). If a length modifier is given, the constant is handled as follows:

1. If the number of hexadecimal digit pairs is greater than the specified length, the extra leftmost bits (and/or bytes) are dropped.

2. If the number of hexadecimal digit pairs is less than the specified length, the necessary bits (and/or bytes) are added to the left and filled with hexadecimal zeros.

An eight-digit hexadecimal constant provides a convenient way to set the bit pattern of a full binary word. The constant in the following example would set the first and third bytes of a word to 1s:

```
|Name   |Operation  |Operand                  |
|-------|-----------|-------------------------|
|       |DS         |0F                       |
|TEST   |DC         |X'FF00FF00'              |
```

The DS instruction sets the location counter to a full-word boundary.

The next example uses a hexadecimal constant as a literal and inserts 1s into bits 24 through 31 of register 5.

```
|Name   |Operation  |Operand                  |
|-------|-----------|-------------------------|
|       |IC         |5,=X'FF'                 |
```

In the following example, the digit A would be dropped, because five hexadecimal digits are specified for a length of two bytes:

```
|Name       |Operation  |Operand              |
|-----------|-----------|---------------------|
|ALPHACON   |DC         |3XL2'A6F4E'          |
|           |           |                     |
```

The resulting constant would be 6F4E, which would occupy the two bytes specified by the length modifier (L2). It would be generated three times, as requested by the duplication factor. Had it been specified as X'A6F4E', the resulting constant would have contained a hexadecimal zero in the leftmost position:

0A6F4E

Binary Constant -- B: A binary constant is written using 1s and 0s enclosed in quotation marks. Only one binary constant may be specified in a statement. Duplication and length may be specified.

The maximum length of a binary constant is 256 bytes.

The implied length of a binary constant is the number of bytes occupied by the constant including the padding necessary to complete a byte. The padding bit used is a 0. Padding or truncation takes place on the left.

The following example shows the coding used to designate a binary constant.

```
|Name    |Operation |Operand                |
|--------|----------|-----------------------|
|BCON    |DC        |B'11011101'            |
|BTRUNC  |DC        |BL1'1C0100011'         |
|BPAD    |DC        |BL1'101'               |
```

BCON would have a length attribute of one.

BTRUNC would assemble with the leftmost bit truncated, as follows:

00100011

BPAD would assemble with five zeros as padding, as follows:

00000101

Fixed-Point Constants -- F and H: A fixed-point constant is written as a decimal number. The number may be an integer, a fraction, or a mixed number (i.e., one with integral and fractional portions) and may be followed by a decimal exponent if desired. The format of the constant is as follows:

1. The number is written as a signed or unsigned decimal value. The decimal point may be placed before, within, or after the number, or it may be omitted, in which case the number is assumed to be an integer. A positive sign is assumed if an unsigned number is specified Unless a scale modifier accompanies a mixed number or fraction, the fractional portion is lost, as explained under Subfield 3: Modifiers.

2. The exponent is optional. If specified it is written immediately after the number as En, where n is an optionally signed decimal value specifying the exponent of the factor 10. The exponent may be in the range -85 to +75. If an unsigned exponent is specified, a plus sign is assumed. The exponent causes the value of the constant to be adjusted by the power of 10 that it specifies before the constant is converted to its binary form.

The number is converted to its binary equivalent and is assembled as a full-word or half-word, depending on whether the type is specified as F or H. It is aligned at the proper full-word or half-word boundary if a length is not specified. An implied length of four bytes is assumed for a full-word (F) and two bytes for a half-word (H). However, any length up to and including eight bytes may be specified for either type of constant by a length modifier, in which case no boundary alignment occurs.

Maximum and minimum values, exclusive of scaling, for fixed-point constants are:

Length	Max	Min
8	$2^{63}-1$	-2^{63}
4	$2^{31}-1$	-2^{31}
2	$2^{15}-1$	-2^{15}
1	$2^{7}-1$	-2^{7}

The binary number occupies the rightmost portion of the field in which it is placed. The unoccupied portion (i.e., the leftmost bits) is filled with the sign. That is, the setting of the bit designating the sign is the setting for the bits in the unused portion of the field. If the value of the number exceeds the length, the necessary leftmost bits are dropped. A negative number is carried in 2s complement form.

If the presence or absence of a scale modifier is such that the rightmost portion of the number must be dropped, rounding occurs. A duplication factor is applied after the constant is converted to its binary format and assembled into the proper number of bytes.

A field of three full-words is generated from the statement shown here. The specified numbers occupy the rightmost three bytes, with the sign propogated through the rest of the word. This constant then appears three times in storage. The location attribute of CONWRD is the address of the leftmost byte of the first word, and the length attribute is four, the implied length for a full-word fixed-point constant. The expression CONWRD+4 could be used to address the second constant (second word) in the field.

```
|Name    |Operation |Operand                |
|--------|----------|-----------------------|
|CONWRD  |DC        |3F'658474'             |
```

The next statement causes the generation of a two-byte field containing a negative constant. Notice that scaling has been specified in order to reserve six bits for the fractional portion of the constant.

```
|Name      |Operation |Operand                |
|----------|----------|-----------------------|
|HALFCON   |DC        |HS6'-25.93'            |
```

The next constant (3.50) is multiplied by 10 to the -2 before being converted to its binary format. The scale modifier reserves eight bits for the fractional portion.

```
|Name      |Operation |Operand                |
|----------|----------|-----------------------|
|FULLCON   |DC        |HS8'3.50E-2'           |
```

The same constant could be specified as a literal:

```
|Name |Operation |Operand                     |
|-----|----------|----------------------------|
|     |AH        |7,=HS8'3.50E-2'             |
```

The final example specifies three constants. Notice that the scale modifier requests four bits for the fractional portion of each constant. The four bits are provided whether or not the fraction exists.

```
|Name      |Operation |Operand                |
|----------|----------|-----------------------|
|THREECON  |DC        |FS4'10,25.3,100'       |
```

Floating-Point Constants -- E and D: A floating-point constant is written as a decimal number, which may be followed by a decimal exponent, if desired. The number may be an integer, a fraction, or a mixed number (i.e., one with integral and fractional portions). The format of the constant is as follows:

1. The number is written as a signed or unsigned decimal value. The decimal point may be placed before, within, or after the number, or it may be omitted, in which case, the number is assumed to be an integer. A positive sign is assumed if an unsigned number is specified.

2. The exponent is optional. If specified it is written immediately after the number as En, where n is an optionally signed decimal value specifying the exponent of the factor 10. The exponent may be in the range -85 to +75. If an unsigned exponent is specified, a plus sign is assumed.

Machine format for a floating-point number is in two parts: the portion containing the exponent, which is sometimes called the characteristic, followed by the portion containing the fraction, which is sometimes called the mantissa. The number specified as a floating-point constant must be converted to a fraction before it can be translated into the proper format. For example, the constant 27.35E2 represents the number 27.35 times 10 to the 2nd. Represented as a fraction, it would be .2735 times 10 to the 4th, the exponent having been modified to reflect the shifting of the decimal point. The presence of an exponent modifier, which will pertain to each constant in the operand, may affect the exponent (see Operand Subfield 3: Modifiers). Thus, the exponent is also altered before being translated into machine format. Once the constant is converted into the proper fraction and exponent, each is translated into its binary equivalent and arranged in machine floating-point format.

The translated constant is placed in a full word or a double word, depending on whether the type is specified as E or D. The characteristic occupies the first byte, and the fraction takes up the remaining bit positions. The constant is aligned at the proper word or double word boundary if a length is not specified. An implied length of four bytes is assumed for a full word (E) and eight bytes is assumed for a double word (D). However, any length up to and including eight bytes may be specified for either type of constant by a length modifier, in which case no boundary alignment occurs.

Within the portion of the floating-point field allocated to the fraction, the hexadecimal point is assumed to be to the left of the leftmost hexadecimal digit, and the fraction occupies the leftmost portion of the field. The fraction is normalized (no leading hexadecimal zeros), unless scaling is specified. If the rightmost portion of the fraction must be dropped because of length or scale modifiers, rounding will occur. Negative fractions are carried in true representation, not in the 2s complement form.

Any of the following statements could be used to specify 46.415 as a positive, full-word, floating-point constant; the last is a machine-instruction statement with a literal operand. Note that the last two constants contain an exponent modifier.

```
|Name  |Operation  |Operand                    |
|------|-----------|---------------------------|
|      |DC         |E'46.415'                  |
|      |DC         |E'46415E-3'                |
|      |DC         |E'+464.15E-1'              |
|      |DC         |E'+.46415E+2'              |
|      |DC         |EE2'.46415'                |
|      |AE         |6,=EE2'.46415'             |
```

Each of the following would be generated as double-word floating-point constants.

```
|Name  |Operation  |Operand                    |
|------|-----------|---------------------------|
|FLOAT |DC         |DE+4'+46,-3.729,+473'      |
```

Decimal Constants -- P and Z: A decimal constant is written as a signed or unsigned decimal value. If the sign is omitted, a plus sign is assumed. The decimal point may be written wherever desired or may be omitted. Scaling and exponent modifiers may not be specified for decimal constants. The maximum length of a decimal constant is 16 bytes. No word boundary alignment is performed.

The placement of a decimal point in the definition does not affect the assembly of the constant in any way, because, unlike fixed-point and floating-point constants, a decimal constant is not converted to its binary equivalent. The fact that a decimal constant is an integer, a fraction, or a mixed number is not pertinent to its generation. Furthermore, the decimal point is not assembled into the constant. The programmer may determine proper decimal point alignment either by defining his data so that the point is aligned or by selecting machine-instructions that will operate on the data properly (i.e., shift it for purposes of alignment).

If zoned decimal format is specified (Z), each decimal digit is translated into one byte. The translation is done according to the character set shown in Appendix A. The rightmost byte contains the sign as well as the rightmost digit. For packed decimal format (P), each pair of decimal digits is translated into one byte. The rightmost digit and the sign are translated into the rightmost byte. The bit configuration for the digits is identical to the configurations for the hexadecimal digits 0-9 as shown in Hexadecimal Self-Defining Term. For both packed and zoned decimals, a plus sign is translated into the hexadecimal digit C, and a minus sign into the digit D.

If an even number of packed decimal digits is specified, one digit will be left unpaired, because the rightmost digit is paired with the sign. Therefore in the leftmost byte, the leftmost four bits will be set to zeros and the rightmost four bits will contain the odd (first) digit.

If no length modifier is given, the implied length for either constant is the number of bytes the constant occupies (taking into account the format, sign, and possible addition of zero bits for packed decimals). If a length modifier is given, the constant is handled as follows:

1. If the constant requires fewer bytes than the length specifies, the necessary number of bytes is added to the left. For zoned decimal format, the decimal digit zero is placed in each added byte. For packed decimals, the bits of each added byte are set to zero.

2. If the constant requires more bytes than the length specifies, the necessary number of leftmost digits or pairs of digits is dropped, depending on which format is specified.

Examples of decimal constant definitions follow.

```
|Name  |Operation  |Operand                    |
|------|-----------|---------------------------|
|FIRST |DC         |P'+1.25'                   |
|SECOND|DC         |Z'-543'                    |
|THIRD |DC         |Z'79.68'                   |
|FOURTH|DC         |PL3'79.68'                 |
```

FIRST would be assembled in two bytes, with the one on the right containing the positive sign and the five specified. SECOND would require three bytes; four bytes would be assembled for THIRD. The length of three specified for FOURTH would be filled with zeros on the left paired with the seven, and the assumed plus sign paired with the eight in the rightmost byte.

The following example illustrates the use of a packed decimal literal.

```
|Name  |Operation  |Operand                    |
|------|-----------|---------------------------|
|      |UNPK       |OUTAREA,=PL2'+25'          |
```

ADDRESS CONSTANTS: An address constant is a storage address that is translated into a constant Address constants are normally used for initializing base registers to facilitate the addressing of storage. Furthermore, they provide the means of communicating between control sections of a

multisection program. However, storage addressing and control section communication are also dependent on the use of the USING assembler instruction and the loading of registers. Coding examples that illustrate these considerations are provided in Programming with the Using Instruction.

An address constant, unlike other types of constants, is enclosed in parentheses There are four types of address constants: A, Y, S, and V.

Complex Relocatable Expressions: These expressions contain two or three unpaired relocatable terms or a negative relocatable term in addition to any absolute or paired relocatable terms that may be present. A complex relocatable expression can only be used to specify an A-type or Y-type address constant. In contrast to relocatable expressions, complex relocatable expressions may represent a negative value. A complex relocatable expression might consist of external symbols (which cannot be paired) and designate an address in an independent assembly that is to be linked and loaded with the assembly containing the address constant.

A-Type Address Constant: This constant is specified as an absolute, relocatable, or complex relocatable expression. (Remember that an expression may be single term or multiterm.) The value of the expression is calculated as explained in the General Information section, under Evaluation of Expressions. The maximum value allowed in this case, however, is $2^{31}-1$. The implied length of an A-type constant is four bytes, and the value is placed in the rightmost portion. Alignment is to a full-word boundary, unless a length is specified. A length modifier may be used, in which case no alignment will occur. The length that may be specified depends on the type of expression used for the constant; a length of 1-4 bytes may be used for an absolute expression, while lengths of 3 and 4 bytes may be used for a relocatable or complex relocatable expression.

The A-type address constant can be used to reference external data, in which case EXTRN and ENTRY points are required.

In this example, ADEND will be assembled as the value (or the address of the leftmost byte) of DUMP.

```
|Name   |Operation  |Operand          |
|-------|-----------|-----------------|
|ADEND  |DC         |A(DUMP)          |
```

In the following example, the field generated from the statement named ACONST contains a constant which occupies four bytes. Note that there is a Location Counter reference. The value of the Location Counter will be the address of the first byte allocated to the constant. The second statement shows the same constant specified as a literal (i.e., address constant literal).

```
|Name   | Operation | Operand             |
|-------|-----------|---------------------|
|ACONST | DC        | A(*+4096)           |
|       | L         | 4,=A(*+4096)        |
```

Note: When the Location Counter reference occurs in a literal, as in the load instruction above, the value of the Location Counter is the address of the first byte of the instruction.

Y-Type Address Constant: A Y-type address constant has the characteristics and format of the A-type constant discussed above except for the following:

1. The constant is assembled as a 16-bit value and aligned to a half-word boundary.
2. The implied length is two bytes.
3. If length specification is used, a length of two to four bytes may be designated for a relocatable or complex expression and 1 to 4 bytes for an absolute expression.

S-Type Address Constant: The S-type address constant is used to store an address in base-displacement form.

The constant may be specified in two ways:

1. As an absolute or relocatable expression, e.g., S(BETA).
2. As two absolute expressions, the first of which represents the displacement value and the second, the base register, e.g., S(400(13)).

The address value represented by the expression in (1) will be broken down by the assembler into the proper base register and displacement value. An S-type constant is assembled as a half word and aligned on a half-word boundary. The leftmost four bits of the assembled constant represents the base register designation, the remaining 12 bits the displacement value.

If length specification is used, only two bytes may be specified. S-type address

constants may not be specified as literals. A duplication factor may not be used.

V-Type Address Constant: This constant is used to reserve storage for the address of an external symbol that is used in branching to other programs. A V-type constant may not be used for external data references. The constant is specified as one relocatable symbol, which need not be identified by an EXTRN statement. Whatever symbol is used is assumed to be an external symbol by virtue of the fact that it is supplied in a V-type address constant.

Note that specifying a symbol as the operand of a V-type constant does not constitute a definition of the symbol for this assembly. Until the program is loaded, the value of the assembled constant is zero. The implied length of a V-type address constant is four bytes, and boundary alignment is to a full word. A length modifier may be used to specify a length of either three or four bytes, in which case no such boundary alignment occurs.

In the following example, four bytes will be reserved on a full-word boundary, and filled with zeros until loading time.

Name	Operation	Operand
VCONST	DC	V(SORT)

DS -- DEFINE STORAGE

The DS instruction is used to reserve areas of storage and to assign names to those areas. The use of this instruction is the preferred way of symbolically defining storage for work areas, input/output areas, etc. The size of a storage area that can be reserved by using the DS instruction is limited only by the maximum value of the Location Counter. Because the maximum length specification is 256, an area larger than 256 must be specified with a duplication factor. For example, the statement DS 2CL200 can be used to reserve 400 positions of main storage.

Name	Operation	Operand
A symbol or blank	DS	One operand written in the format described in the following text

The format of the DS operand is identical to that of the DC operand; exactly the same subfields are employed and are written in exactly the same sequence as they are in the DC operand, with the following exception:

> The specification of data (subfield 4) is optional in a DS operand, but it is mandatory in a DC operand.

If a DS operand specifies a constant in subfield 4, (and no length is specified in subfield 3) the assembler determines the length of the data and reserves the appropriate amount of storage. It does not assemble the constant. The ability to specify data and have the assembler calculate the storage area that would be required for such data is a convenience to the programmer. If he knows the general format of the data that will be placed in the storage area during program execution all he needs to do is show it as the fourth subfield in a DS operand. The assembler then determines the correct amount of storage to be reserved, thus relieving the programmer of length considerations.

If the DS instruction is named by a symbol, its value attribute is the location of the leftmost byte of the reserved area. The length attribute of the symbol is the length (implied or explicit) of the type of data specified. Any positioning required for aligning the storage area to the proper type of boundary is done before the address value is determined. Because no data is assembled at this time, skipped bytes are not zeroed.

Each field type (e.g., hexadecimal, character, floating-point) is associated with certain characteristics (these are summarized in Appendix F). The associated characteristics will determine which field-type code the programmer selects for the DS operand and what other information he adds, notably a length specification or a duplication factor. For example, the E floating-point field and the F fixed-point field both have an implied length of four bytes. The leftmost byte is aligned to a full-word boundary. Thus, either code could be specified if it were desired to reserve four bytes of storage aligned to a full-word boundary. To obtain a length of eight bytes, one could specify either the E or F field type with a length modifier of eight. However, a duplication factor would have to be used to reserve a larger area, because the maximum length specification for either type is eight bytes. Note also that specifying length would cancel any special boundary alignment.

In contrast, packed and zoned decimal (P and Z), character (C), hexadecimal (X), and binary (B) fields have an implied length of one byte. Any of these codes, if used,

would have to be accompanied by a length modifier, unless just one byte is to be reserved. Although no alignment occurs, the use of these field types permits greater latitude in length specifications the maximum for these types being 256 bytes. However, if a symbol that is defined by a P or Z field type with a length modifier greater than 16 is used as an operand in a decimal machine instruction, a length error will occur. Unless a field of one byte is desired, either the length must be specified for the C, X, P, Z, or B field types, or else the data must be specified (as the fourth subfield), so that the assembler can calculate the length.

To define four 10-byte fields and one 100-byte field, the respective DS statements might be as follows:

```
|Name  |Operation  |Operand          |
|------|-----------|-----------------|
|FIELD |DS         |4CL10            |
|AREA  |DS         |CL100            |
```

Although FIELD might have been specified as one 40-byte field, the preceding definition has the advantage of providing FIELD with a length attribute of 10. This would be pertinent when using FIELD as a machine- instruction operand governed by a length consideration.

Additional examples of DS statements are shown below:

```
|Name  |Operation|Operand                          |
|------|---------|---------------------------------|
|ONE   |DS       |CL80(one 80-byte field,          |
|      |         |  length attribute of 80)        |
|TWO   |DS       |80C(80 one-byte fields,          |
|      |         |  length attribute of            |
|      |         |  one)                           |
|THREE |DS       |6F(six full words, length        |
|      |         |  attribute of four)             |
|FOUR  |DS       |D(one double word, length        |
|      |         |  attribute of eight)            |
|FIVE  |DS       |4H(four half-words,              |
|      |         |  length attribute of            |
|      |         |  two)                           |
```

Note: A DS statement causes the storage area to be reserved but not set to zeros. The programmer should not assume that the area will contain zeros when the program is loaded.

Special Uses of the Duplication Factor

FORCING ALIGNMENT: The Location Counter can be forced to a double-word, full-word, or half-word boundary by using the appropriate field type (e.g., D, F, or H) with a duplication factor of zero. This method may be used to obtain boundary alignment that otherwise would not be provided. For example, the following statements would set the Location Counter to the next double-word boundary and then reserve storage space for a 128-byte field (whose leftmost byte would be on a double-word boundary).

```
|Name  |Operation  |Operand          |
|------|-----------|-----------------|
|      |DS         |0D               |
|AREA  |DS         |CL128            |
```

DEFINING FIELDS OF AN AREA: A DS instruction with a duplication factor of zero can be used to assign a name to an area of storage without actually reserving the area. Additional DS and/or DC instructions may then be used to reserve the area and assign names to fields within the area (and generate constants if DC is used).

For example, assume that 80-character records are to be read into an area for processing and that each record has the following format:

```
Positions 5-10     Payroll Number
Positions 11-30    Employee Name
Positions 31-36    Date
Positions 47-54    Gross Wages
Positions 55-62    Withholding Tax
```

The following example illustrates how DS instructions might be used to assign a name to the record area, then define the fields of the area and allocate the storage for them. Note that the first statement names the entire area by defining the symbol RDAREA; the statement gives RDAREA a length attribute of 80 bytes, but does not reserve any storage. Similarly, the fifth statement names a 6-byte area by defining the symbol DATE; the three subsequent statements actually define the fields of DATE and allocate storage for them. The second, ninth, and last statements are used for spacing purposes and, therefore, are not named.

Name	Operation	Operand
RDAREA	DS	0CL80
	DS	CL4
PAYNO	DS	CL6
NAME	DS	CL20
DATE	DS	0CL6
DAY	DS	CL2
MONTH	DS	CL2
YEAR	DS	CL2
	DS	CL10
GROSS	DS	CL8
FEDTAX	DS	CL8
	DS	CL18

CCW -- DEFINE CHANNEL COMMAND WORD

The CCW instruction provides a convenient way to define and generate an eight-byte channel command word aligned at a double-word boundary. The format of the CCW instruction statement is:

Name	Operation	Operand
A symbol or blank	CCW	Four operands, separated by commas, specifying the contents of the channel command word in the format described in the following text

All four operands must appear. They are written, from left to right, as follows:

1. An absolute expression that specifies the command code. This expression's value is right-justified in byte 1.

2. An absolute or relocatable expression specifying the data address. The value of this expression is right-justified in bytes 2-4.

3. An absolute expression that specifies the flags for bits 32-36 and zeros for bits 37-39. The value of this expression is right-justified in byte 5. (Byte 6 is set to zero.)

4. An absolute expression that specifies the count. The value of this expression is right-justified in bytes 7-8.

The following is an example of a CCW statement:

Name	Operation	Operand
	CCW	2,READAREA,X'48',80

Note that the third operand sets bits 37-39 to zero, as required. The bit pattern of this operand is as follows:

32-35	36-39
0100	1000

If there is a symbol in the name field of the CCW instruction, it is assigned the address value of the leftmost byte of the channel command word. The length attribute of the symbol is eight. The internal machine format of a channel command word is:

Byte	Bits	Usage
1	0-7	Command code
2-4	8-31	Data address
5	32-36	Flags
	37-39	Must be zero
6	40-47	Set to zero
7-8	48-63	Count

LISTING CONTROL INSTRUCTIONS

The listing control instructions are used to identify an assembly listing and assembly output cards, to provide blank lines in an assembly listing, and to designate how much detail is to be included in an assembly listing. In no case are instructions or constants generated in the object program. If listing control statements are used within a DSECT, they are treated as comments and not executed. For example, if the EJECT instruction is used within a DSECT, it does not cause the listing to be ejected.

TITLE -- IDENTIFY ASSEMBLY OUTPUT

The TITLE instruction enables the programmer to identify the assembly listing and assembly output cards. The format of the TITLE instruction statement is as follows:

Name	Operation	Operand
Name or blank	TITLE	A sequence of characters, enclosed in single quotation marks

If the first TITLE statement in a program appears before the START statement, it may contain an entry in the name field. This entry may contain one to four alphabetic or numeric characters in any combination. Any additional characters are ignored. The contents of the name field are punched into columns 73-76 of all the output cards for the program, except in those cards produced by means of a REPRO or PUNCH assembler instruction. An entry in the name field of any other TITLE statement is ignored.

The operand field of a TITLE statement may contain up to 100 characters, enclosed in single quotation marks. A continuation card may be used, if necessary. Any characters in excess of 100 are ignored. The contents of the operand field are printed at the top of each page of the assembly listing. The TITLE statement itself does not appear in the source listing.

A program may contain more than one TITLE statement. Each TITLE statement provides the heading for pages in the assembly listing that follows it, until another TITLE statement is encountered. Each TITLE statement encountered after the first statement causes the listing to be advanced to a new page (before the heading is printed).

For example, if the following statement is the first TITLE statement to appear in a program, and it appears before the START statement:

Name	Operation	Operand
PGM1	TITLE	'FIRST HEADING'

then PGM1 is punched into all of the output cards (columns 73-76), except those produced by a REPRO or PUNCH statement, and this heading appears at the top of each page: FIRST HEADING.

If the following statement occurs later in the same program:

Name	Operation	Operand
	TITLE	'A NEW HEADING'

then PGM1 is still punched into the output cards, but each following page begins with the heading: A NEW HEADING.

EJECT -- START NEW PAGE

The EJECT instruction affects only the assembly listing and provides a convenient way to separate program routines in the listing. This instruction causes the remainder of the present page to be skipped and the listing to continue at the top of the next page, below the heading line. If the ejection occurs at the first line of the page, the entire page is skipped.

If two or more EJECT instructions are issued in succession, a complete page is skipped for each EJECT after the first, and the listing continues on the page that is in printing position after the last EJECT has been executed. Each page that is skipped is printed with a heading line, however.

The format of the EJECT instruction statement is:

Name	Operation	Operand
Blank	EJECT	Not used; any operand is treated as a comment

The EJECT statement itself does not appear in the source listing.

SPACE -- SPACE LISTING

The SPACE instruction is used to insert one or more blank lines in the listing. The format of the SPACE instruction statement is as follows:

Name	Operation	Operand
Blank	SPACE	A decimal value or blank

A decimal value is used to specify the number of blank lines to be inserted in the assembly listing. A blank operand causes one blank line to be inserted. If this value exceeds the number of lines remaining on the listing page, the statement will have the same effect as an EJECT statement The SPACE statement itself does not appear in the source listing.

PRINT -- PRINT OPTIONAL DATA

The PRINT instruction is used to control printing of the assembly listing. The format of the PRINT instruction statement is:

Name	Operation	Operand
Blank	PRINT	One to three operands

One to three of the following operands are used:

ON - A listing is printed.

OFF - No listing is printed.

GEN - All statements generated by macro-instructions are printed.

NOGEN - Statements generated by macro-instructions are not printed. However, the macro-instruction itself and messages resulting from the MNOTE instruction, if used, will appear in the listing.

DATA - Constants are printed out in full in the listing.

NODATA - Only the first eight bytes (16 hexadecimal digits) or the first constant, whichever is shorter, of the assembled data, is printed on the listing.

A program may contain any number of PRINT statements. A PRINT statement controls the printing of the assembly listing until another PRINT statement is encountered.

Until the first PRINT statement (if any) is encountered, the following is assumed:

Name	Operation	Operand
	PRINT	ON,NODATA,GEN

For example, if the statement:

Name	Operation	Operand
	DC	XL256'00'

appears in a program, 256 bytes of zeros are assembled. If the statement:

Name	Operation	Operand
	PRINT	DATA

is the last PRINT statement to appear before the DC statement, all 256 bytes of zeros are printed in the assembly listing. However, if:

Name	Operation	Operand
	PRINT	NODATA

is the last PRINT statement to appear before the DC statement, only eight bytes of zeros are printed in the assembly listing.

PROGRAM CONTROL INSTRUCTIONS

The program control instructions are used to specify the end of an assembly, to set the Location Counter to a value or word boundary, to specify the placement of literals in storage, to check the sequence of input cards, to indicate statement format, and to punch a card. Except for the LTORG and CNOP instructions, none of these assembler instructions generate instructions or constants in the object program.

If program control instructions are used within a DSECT, they are treated as comments and not executed. For example, if the XFR instruction is used within a DSECT, it does not cause any transfer card to be generated.

If the user plans to write his own macro instruction routines, the assembler instructions ICTL (input format control), ISEQ (input sequence checking), and LTORG (begin literal pool) may not be used as instructions within the macro routine.

ICTL -- INPUT FORMAT CONTROL

The ICTL instruction allows the programmer to alter the normal format of his source program statements. The ICTL statement must precede all other statements in the source program and may be used only once. The format of the ICTL instruction statement is as follows:

Name	Operation	Operand
Blank	ICTL	1-3 decimal values of the form b,e,c,

Operand b specifies the begin column of the source statement. It must always be specified, and must be from 1-40, inclusive. Operand e specifies the end column of the source statement. The end column, when specified, must be from 41-79, inclusive; when not specified, it is

assumed to be 71. The column after the end column is used to indicate whether the next card is a continuation card. Operand c specifies the continue column of the source statement. The continue column, when specified, must be from 2-40 and must be greater than b. If the continue column is not specified, the assembler assumes that there are no continuation cards and all statements must be contained in a single card.

If no ICTL statement is used in the source program, the assembler assumes that 1, 71, and 16 are the begin, end, and continue columns, respectively.

The ICTL card itself is processed under normal format and any non-blank character punched into column 72 indicates the presence of continuation cards. Therefore, column 72 should be left blank because continuation cards are not required for the ICTL card. If column 72 contains any non-blank character, the card following the ICTL card is treated as a continuation card and reading begins in column 16, causing columns 1-15, inclusively, to be ignored.

The next example designates the begin column as column 25. Since the end column is not specified, it is assumed to be column 71. No continuation cards are recognized because the continue column is not specified.

Name	Operation	Operand
	ICTL	25

ISEQ -- INPUT SEQUENCE CHECKING

The ISEQ instruction is used to check the sequence of input cards. The format of the ISEQ instruction statement is as follows:

Name	Operation	Operand
Blank	ISEQ	Two decimal values of the form 1,r; or blank

The operands 1 and r, respectively, specify the leftmost and rightmost columns of the field in the input cards to be checked. Operand r must be equal to or greater than operand 1. Operand 1 must be greater than the end column plus one. The field specified by operands 1 and r must not be greater than seven bytes.

Sequence checking begins with the first card following the ISEQ statement. Comparison of adjacent cards makes use of the eight-bit internal collating sequence

An ISEQ statement with a blank operand terminates the operation. Checking may be resumed with another ISEQ statement

Sequence checking is only performed on statements contained in the source program. Statements generated by a macro are not checked for sequence.

REPRO -- REPRODUCE FOLLOWING CARD

The basic operating system Linkage Editor requires Phase Definition (PHASE) and Include Module (INCLUDE) cards. The REPRO Assembler instruction allows the inclusion of such cards into the object program deck to eliminate the necessity of manually inserting them.

The REPRO Assembler instruction causes the Assembler to punch a duplicate (in 80-80 format) of the card immediately following the REPRO instruction. The punched cards resulting from REPRO instructions appear at the same point in the assembled text as they appeared in the source program. If any REPRO instructions precede the START instruction, or the implied start position (if no START instruction is used), the cards punched will precede the ESD cards for the assembly.

The format of the REPRO Assembler instruction is as follows:

Name	Operation	Operand
Blank	REPRO	Not used; any operand is treated as a comment

PUNCH -- PUNCH A CARD

The PUNCH assembler instruction may be used to perform the same functions as the REPRO assembler instruction. The PUNCH assembler instruction causes the data in the operand to be punched into a card. As many PUNCH statements may be used as are necessary. The format is:

Name	Operation	Operand
Blank	PUNCH	'PUNCH A CARD'

Using character representation, the operand is written as a string of up to 80 characters enclosed in single quotation

marks. A continuation card may be used, if necessary. Any characters in excess of 80 are ignored. All characters, including blanks, are valid. The position immediately to the right of the left quotation mark is regarded as column one of the card to be punched. The assembly program does not process the data in the operand of a PUNCH statement other than causing it to be punched in a card.

The punched cards resulting from PUNCH instructions appear at the same point in the assembled text as they appeared in the source program. If any PUNCH instructions precede the START instruction, or the implied start position (if no START instruction is used), the cards punched will precede the ESD cards for the assembly.

The main facility provided by the PUNCH instruction over the REPRO instruction is the capability of the macro generator to substitute values for symbolic parameters or SET variable symbols in the operand of a PUNCH instruction appearing in a macro definition. This allows such things as the controlled generation of phase names.

XFR -- GENERATE A TRANSFER CARD

A transfer card is used by the basic operating system Linkage Editor program to define the transfer point or entry point of a phase, or overlay. The XFR Assembler instruction is provided to cause the generation of a transfer card in the assembled text in the same location that the XFR instruction appeared in the source program.

The format of the XFR instruction is as follows:

Name	Operation	Operand
Blank	XFR	A relocatable symbol

The symbol in the operand field must appear within the assembly or be previously defined as either an entry or external symbol.

ORG -- SET LOCATION COUNTER

The ORG instruction is used to alter the setting of the Location Counter for the current control section. The format of the ORG instruction statement is:

Name	Operation	Operand
Blank	ORG	A relocatable expression or blank

Any symbols in the expression must have been previously defined. The unpaired relocatable symbol must be defined in the same control section in which the ORG statement appears.

The Location Counter is set to the value of the expression in the operand. If the operand is omitted, the Location Counter is set to a location that is one byte higher than the maximum location assigned for the control section up to this point.

An ORG statement must not be used to specify a location below the beginning of the control section in which it appears. For example, the statement:

Name	Operation	Operand
	ORG	*-500

is invalid if it appears less than 500 bytes from the beginning of the current control section.

If it is desired to reset the Location Counter to a value that is one byte beyond the highest location yet assigned (in the control section), the following statement would be used:

Name	Operation	Operand
	ORG	

If previous ORG statements have reduced the Location Counter for the purpose of redefining a portion of the current control section, an ORG statement with an omitted operand can then be used to terminate the effects of such statements and restore the Location Counter to its highest setting.

LTORG -- BEGIN LITERAL POOL

The LTORG instruction causes all literals thus far encountered in the source program up to the LTORG statement (either from the beginning of the program or from a previous LTORG statement) to be assembled at appropriate boundaries starting at the first double-word boundary following the LTORG statement. The format of the LTORG instruction statement is:

Name	Operation	Operand
Symbol or blank	LTORG	Not used; any operand is treated as a comment

The symbol represents the address of the first byte of the literal pool. It has a length attribute of one.

Special Addressing Consideration

Any literals used after the last LTORG statement in a program are placed at the end of the first control section. If there are no LTORG statements in a program, all literals used in the program are placed at the end of the first control section. In these circumstances the programmer must ensure that the first control section is always addressable. This means that the base address register for the first control section should not be changed through usage in subsequent control sections. If the programmer does not wish to reserve a register for this purpose he may place a LTORG statement at the end of each control section thereby ensuring that all literals appearing in that section are addressable.

CNOP -- CONDITIONAL NO OPERATION

The CNOP instruction allows the programmer to align an instruction at a specific word boundary. If any bytes must be skipped in order to align the instruction properly, the assembler insures an unbroken instruction flow by generating no-operation instructions. This facility is useful in creating calling sequences consisting of a linkage to a subroutine followed by parameters such as channel command words (CCW).

The CNOP instruction insures the alignment of the Location Counter setting to a half-word, word, or double-word boundary. If the Location Counter is already properly aligned, the CNOP instruction has no effect. If the specified alignment requires the Location Counter to be incremented, one to three no-operation instructions are generated, each of which uses two bytes.

The format of the CNOP instruction statement is as follows:

Name	Operation	Operand
Blank	CNOP	Two decimal terms of the form b,w

Operand b specifies at which byte in a word or double word the Location Counter is to be set; b can be 0, 2, 4, or 6. Operand w specifies whether byte b is in a word (w=4) or double word (w=8). The following pairs of b and w are valid:

b,w	specifies
0,4	Beginning of a word
2,4	Middle of a word
0,8	Beginning of a double word
2,8	Second half word of a double word
4,8	Middle (third half word) of a double word
6,8	Fourth half word of a double word

Figure 5 shows the position in a double word that each of these pairs specifies. Note that both 0,4 and 2,4 specify two locations in a double word.

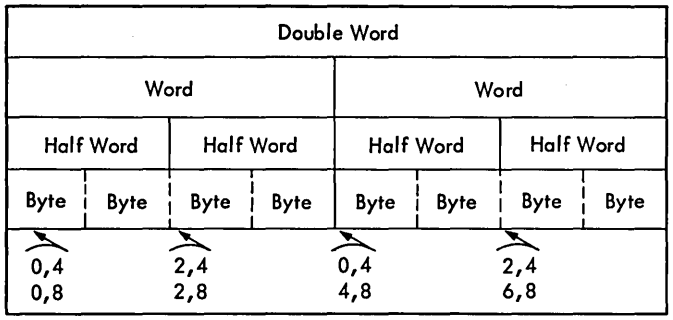


Figure 5. CNOP Alignment

Assume that the Location Counter is currently aligned at a double-word boundary Then the CNOP instruction in this sequence:

Name	Operation	Operand
	CNOP	0,8
	BALR	2,14

has no effect; it is merely printed in the assembly listing. However, this sequence:

Name	Operation	Operand
	CNOP	6,8
	BALR	2,14

causes three branch-on-conditions (no-operations) to be generated, thus aligning the BALR instruction at the last half-word in a double word as follows:

Name	Operation	Operand
	BCR	0,0
	BCR	0,0
	BCR	0,0
	BALR	2,14

After the BALR instruction is generated, the Location Counter is at a double-word boundary, thereby insuring an unbroken instruction flow.

END -- END ASSEMBLY

The END instruction terminates the assembly of a program. It may also designate a point in the program or in a separately assembled program to which control may be transferred after the program is loaded. The END instruction must always be the last statement in the source program.

The format of the END instruction statement is as follows:

Name	Operation	Operand
Blank	END	A relocatable expression or blank

The operand specifies the point to which control is transferred when loading is complete. This point is usually the first machine-instruction in the program, as shown in the following sequence

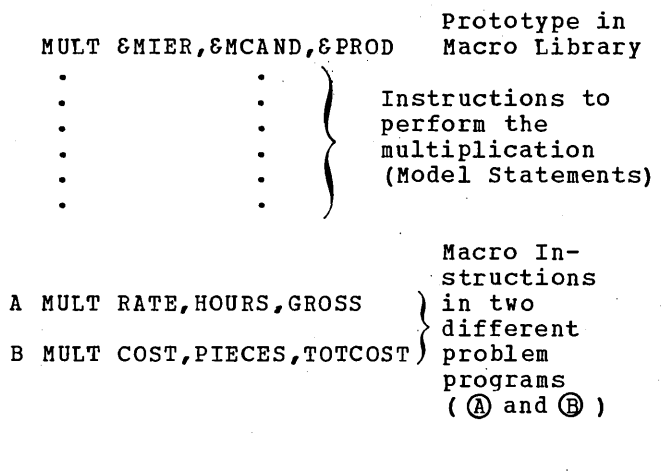
Name	Operation	Operand
NAME	CSECT	
AREA	DS	50F
BEGIN	BALR	2,0
	USING	*,2
	.	
	.	
	.	
	END	BEGIN

If the END statement contains a symbolic address in the operand field, the Assembler automatically inserts the transfer address in the END card.

If the user plans to write his own macro-instruction routines, the END instruction may not be used as an instruction within his macro routines.

Note: If the operand contains an external symbol, only a single-term relocatable expression is allowed.

The assembler provides a macro system to reduce the amount of repetitive coding required for general routines that must be reused a number of times in the same or different programs. For example, the routines for transferring records from disk to main storage , checking for accuracy, and deblocking to obtain a single record for processing are used for any logical input file on disk. Such routines involve many instructions that can be written once and, with modification of addresses, can be used repeatedly in any number of programs.



The macro system is composed of two basic parts:

1. Source-program macro instructions.

2. A macro library of prewritten flexible routines (called definitions).

A direct relationship exists between these parts. That is, a single macro instruction written in the source program is replaced, in the assembled object program, by the routine taken from the macro library. The macro library routine consists of a predetermined series of many instructions. Thus, the system derives its name. A definition of macro is "of or involving large quantities." For one instruction, many instructions are assembled.

The proper routine is included in the object program by the matching of mnemonic operation codes. Therefore, the exact same Op code is used in the macro instruction and in the identification of the corresponding set of instructions in the macro library.

As the macro-library instructions are assembled, they are tailored to fit the particular problem program. This is done by a substitution process. The first instruction of a macro routine (after the macro definition header) in the macro library is a prototype statement. This is a pattern that consists of variable symbolic operands (called parameters) for which values are substituted when the macro routine is used by a specific program. The macro instruction that is included in the user's program specifies the parameter values (commonly abbreviated to the term parameter) that are to be substituted in the macro-library routine when it is assembled. An example of this is:

This illustrates the prototype statement of a multiplication routine that could be used by any program to multiply any two factors and store the product in a specified location. Program Ⓐ might use the macro to multiply rate times hours and store the result in a field labeled gross. Program Ⓑ might use the same macro from the macro library to multiply cost times pieces to obtain total cost.

Each program specifies in the macro instruction the values (parameters) that are applicable to its own job (rate or cost, hours or pieces, gross or total cost). Then when the macro routine is assembled, these parameter values are substituted for the parameters in the prototype statement. The parameter values are also substituted in all the instructions that follow the prototype to actually perform the multiplication. The statements following the prototype are known as model statements. For the multiplication example, the complete routine might be:

```
      MACRO -- Macro Definition Header
&NAME MULT  &MIER,&MCAND,&PROD -- Prototype
                                  Statement
&NAME L     3,&MCAND
      M     2,&MIER     Model Statements
      ST    3,&PROD
      MEND -- Macro Definition Trailer
```

In both illustrations the & character preceding the symbolic name is part of the syntax of the macro definition language (see the Macro Definition Language

publication, as listed on the front cover of this manual).

IBM provides a number of prewritten macro library routines and specifies the macro instructions that can be used by the programmer to call these routines from the library. Other routines can be written by the user and stored in the macro library. User-written routines must follow the same rules as the IBM routines. The macros supplied by IBM and discussed in this publication are grouped in four categories:

- Input/output control macros
- File definition macros
- Supervisor-communication macros
- Supervisor-assembly macros.

MACRO INSTRUCTION FORMAT

A macro instruction, which is a source language statement, is interpreted by the assembler. The assembler uses the macro definition (macro library routine) to replace the single statement with many assembler language statements. For correct interpretation, the format of the macro instruction must correspond to the format of the prototype statement in the macro definition. Therefore, the format of the prototype dictates the form in which the macro instruction must be written in the problem program.

The name field in the macro instruction may contain a symbolic name if the name field of the prototype has a parameter.

The operation field in the macro instruction must contain exactly the same mnemonic operation code as the prototype (for example, MULT). This may be any alphameric code with a maximum of five characters, provided the first character is a letter.

The parameters in the operand field of a macro instruction must be written in the same format as those in the operand field of the prototype. Either of two types of formats can be used:

1. Positional format

2. Keyword format.

Each type of parameter has a set of rules that must be followed.

Positional Format

In this format the order and placement of the parameter values in the macro instruction must correspond exactly with the order and placement of the parameters in the prototype statement. Each parameter except the last must be followed by a comma. Thus, in the previous illustration:

```
 RATE   corresponds to MIER
HOURS   corresponds to MCAND
GROSS   corresponds to PROD
```

If a parameter is to be omitted in the macro instruction, while following parameters are included, a comma must be inserted to indicate the omission. In this way, the proper parameters both before and after the omission correspond. However, no commas need to be included after the last parameter used.

If the parameters cannot be contained in the operand field of one card, up to 49 continuation cards may be used. The continue column of each card (except the last) must contain a continuation punch (any nonblank character), as in any assembler language card. The maximum length of a parameter is the same as for an assembler symbol--eight characters. When one or more continuation cards are used, parameters must fill each card to the continuation column with no intervening blanks. (A blank indicates that the card contains no more parameters.)

Keyword Format

This format provides a direct mnemonic association between the macro instruction and the prototype statement. The exact parameters used in the prototype are specified (without the &) in the macro instruction, where they are equated to the value for this job. Thus, they have the form of: keyword followed by an equal sign followed by the value to be substituted in the assembled routine. For example:

```
MIER=RATE, MCAND=HOURS, PROD=GROSS
```

Because the association of parameters is performed through the use of the keywords, the parameters in the macro instruction may appear in any order, and any that are not needed may be omitted. This association is not dependent upon the order in which they are written. The term to the right of the equal sign must not exceed eight characters.

Different keyword parameters may be punched in the same card, each followed by a comma, like the positional type. Or, they may be punched in separate cards. (When continuation cards are used for a keyword format macro, the parameters need not fill the operand field to the continuation column.)

```
MIER=RATE,
PROD=GROSS,
MCAND=HOURS
```

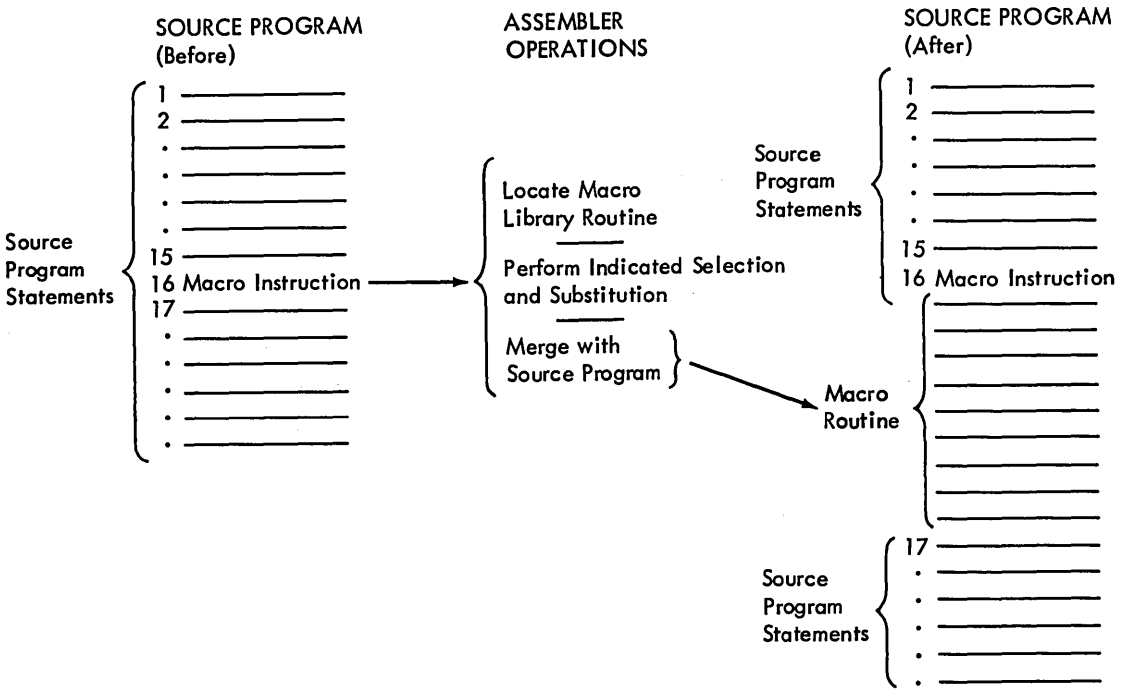


Figure 6. Schematic of Macro Processing

When separate cards are used for a set of parameter specifications each card except the last must be punched with a comma immediately following the parameter and a continuation punch in column 72. Comments may be punched in any card if at least one space is left after the comma, or, in the last card, after the parameter.

ASSEMBLY OF THE MACRO

At program assembly time, the macro instruction specifies which routine is to be called from the macro library. The routine is extracted, tailored by the parameters in the macro instruction, and inserted in the program (Figure 6). The complete program now consists of both source-program statements and tailored model statements from the macro library in assembler language. In subsequent phases of the assembly, the entire program is processed to produce the machine-language object program.

INPUT/OUTPUT CONTROL MACROS

A number of macro routines are provided by IBM for the input/output control (IOCS) of records from various I/O units. These routines control such functions as:

- Opening and closing files
- Transferring records
- Blocking and deblocking records
- Checking and writing disk or tape label
- Error checking.

IOCS handles files of records in the following I/O units:

IBM 1442 Card Read-Punch

IBM 2501 Card Reader

IBM 2520 Card Read-Punch or 2520 Card Punch

IBM 2540 Card Read-Punch

IBM 1403 Printer

IBM 1404 Printer (continuous forms only)

IBM 1443 Printer

IBM 1445 Printer

IBM 2311 Disk Storage Drive

IBM 2401, 2402, 2403, 2404 Magnetic Tape Units (If

variable-length records are written on 7-track tape, the Data Conversion special feature is required.)

IBM 1052 Printer-Keyboard (One 1052 is supported, for operator communication only. It is attached to the multiplexor channel.)

IBM 2671 Paper Tape Reader

IBM 1285, 1287 Optical Readers

STR (Synchronous Transmitter Receiver) Devices connected by leased or dial lines through an IBM Synchronous Data Adapter - Type I, on an IBM 2701 Data Adapter Unit. The following devices are supported:

a. 1009 Data Transmission Unit
b. 1013 Card Transmission Terminal
c. 1974 II Data Transmission Terminal
d. 1978 Print, Read, Punch Terminal
e. System/360, Model 30, 40, 50, 65 or 75 with a 2701 Data Adapter Unit attached.
f. System/360, Model 20 with a Communications Adapter
g. 7701, 7702 Magnetic Tape Transmission Terminal
h. 7711 Data Communication Unit.

BSC (Binary Synchronous Communication) IBM 2701 Data Adapter Unit equipped with an IBM Synchronous Data Adapter--Type II, connected by leased or dial line to a remote IBM System/360, Model 30, 40, 50, 65, 67 (working in 65 mode), or 75. The remote CPU is equipped with an IBM 2701 Data Adapter Unit with an SDA II or an IBM 2703 Transmission Control Unit with Binary Synchronous features.

Note: When BSC support routines are used, a minimum of 16K of main storage is required.

IOCS supports any channel configuration up to one multiplexor channel and two selector channels.

Both IBM-supplied system programs and user problem programs can use magnetic tape. However, the main storage required for physical and logical IOCS for both tape and disk will probably make this unfeasible in systems with less than 16K bytes of main storage. For example, IBM-supplied system programs assume that the system Supervisor (including physical but not logical IOCS) will occupy a maximum of 4096 bytes of main storage, leaving 4096 bytes (in an 8K system) available for the execution of the system programs. If, however, the user's installation includes many different types of I/O units and features, the system Supervisor may be assembled to include routines for a combination of units that, in total, require more than 4096 bytes. Similarly, the number and size of the routines included in the Supervisor affect the number of main storage positions available for a user's problem program. For detailed information about Supervisor and logical IOCS main-storage requirements, see the Programmer's Guide, as listed on the front cover of this publication.

When the user's program is executed, the portion of the problem program that communicates with IOCS and with the Supervisor must be located in the first 64K of main storage . This includes channel command words (CCW), command control blocks (CCB), file definitions (DTF), and program check, interval timer, and operator-communication routines. Furthermore, whenever the problem program is executed in a disk-resident system, the last two routines (interval-timer, and operator communication), if used, must not be located in the first 2500 main storage positions above the end of the Supervisor.

Physical IOCS vs Logical IOCS

The input/output control is considered to consist of two parts: physical IOCS and logical IOCS. Physical IOCS controls the actual transfer of records between the external medium and main storage . That is, it performs the functions of issuing channel commands and handling associated I/O interruptions. Physical IOCS consists of the following routines:

Start I/O routine
Interruption routine
Channel scheduler
Device error routines.

These physical IOCS routines are part of the Supervisor, which is permanently located in lower main storage while problem programs are being executed.

Logical IOCS controls those functions that a user would have to perform to locate a logical record for processing. A logical record is one unit of information in a file of like units; for example, one employee's record in a master payroll file, one part-number record in an inventory file, etc. One or many logical records may be included within one physical record, such as a physical disk or tape record (from gap to gap). The term logical IOCS refers to routines that perform the following functions:

- Blocking and deblocking records
- Switching between I/O areas when two areas are specified for a file

- Handling end-of-file and end-of-volume conditions
- Checking/writing labels.

The logical IOCS routines required for the execution of a problem program are assembled with that program. The particular routines required are determined from the definitions of the logical files used by the program.

Logical IOCS uses physical IOCS to execute I/O commands whenever it determines that a transfer of data is required. For example, if a file consists of blocked records and a block has been read into main storage (Figure 7), logical IOCS merely makes each record in succession available to the user, until the end of the block is reached. No physical IOCS is required. When logical IOCS determines that the last record in the block has been processed, however, it requests physical IOCS to start an I/O operation to transfer the next physical record (gap to gap) into main storage. In the illustration, only logical IOCS (LIO) is required to make records 2 and 3 (and 5 and 6) available; however, physical IOCS (PIO) is also required to make record 4 available (records 4 through 6 are transferred in one block).

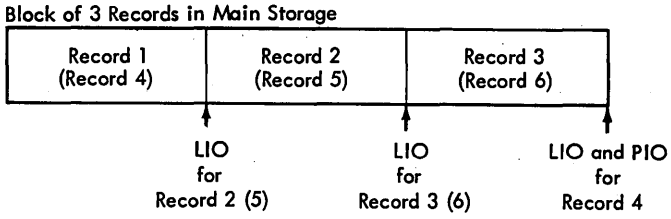


Figure 7. Physical IOCS vs Logical IOCS

Both logical IOCS macros (such as GET, PUT, READ, WRITE, etc) and physical IOCS macros (such as EXCP and WAIT) are available to the programmer for handling records. The logical IOCS macro routines cause all the functions of both logical and physical IOCS to be performed for the programmer. When he issues an imperative GET instruction for a record, for example, that record is transferred to main storage, if necessary, and it is available for the execution of the next program instruction.

The physical IOCS routines completely bypass the logical IOCS functions (for example, blocking and deblocking). They permit the problem program to utilize physical IOCS functions directly. To transfer a physical record (such as a disk or tape record), for example, the problem program issues an EXCP macro instruction (execute channel program). This causes a request for data transfer to be placed in the channel scheduler, and program execution immediately continues with the next problem-program instruction. However, the disk or tape record will not be available in main storage until some later time. Therefore when the record is needed for processing, the program must test to find out if the transfer has been completed. This is accomplished by issuing the WAIT macro instruction.

The functions of physical and logical IOCS routines are shown schematically in Figure 8.

Types of Processing

The logical IOCS routines process records in consecutive order, in random order by the Direct Access Method (DAM), or randomly and sequentially by the Indexed Sequential File Management System (ISFMS). Consecutive processing applies to all files in serial-type I/O devices (such as card reader, tape, printer, etc), and to records on 2311 disk when they are processed in a serial-type order. The DAM and ISFMS types of processing apply only to files of disk records. Other logical IOCS routines allow processing with STR (Synchronous Transmitter Receiver) devices, or allow CPU-to-CPU Binary Synchronous Communication.

Consecutive Processing: Consecutive processing is used to read/write and process successive records in a logical file. For example, card records are processed in the order the cards are fed; tape records are processed starting with the first record after a header label and continuing through the records to the trailer label; disk records are processed starting with a beginning disk address and continuing in order through the records on successive tracks (and possibly cylinders) to the ending address.

A consecutive file on disk is contained within one or more sets of limits, which are specified by Job Control XTENT cards. If the logical file consists of more than one set of limits, IOCS will automatically process each set as needed by the user. The records within each set must be adjacent and contained within one volume (disk pack). However, the sets need not be adjacent, and they may be on one or more volumes. A file written on disk by the direct access method can also be processed consecutively, if desired.

The basic macros used for consecutive processing are GET and PUT. These

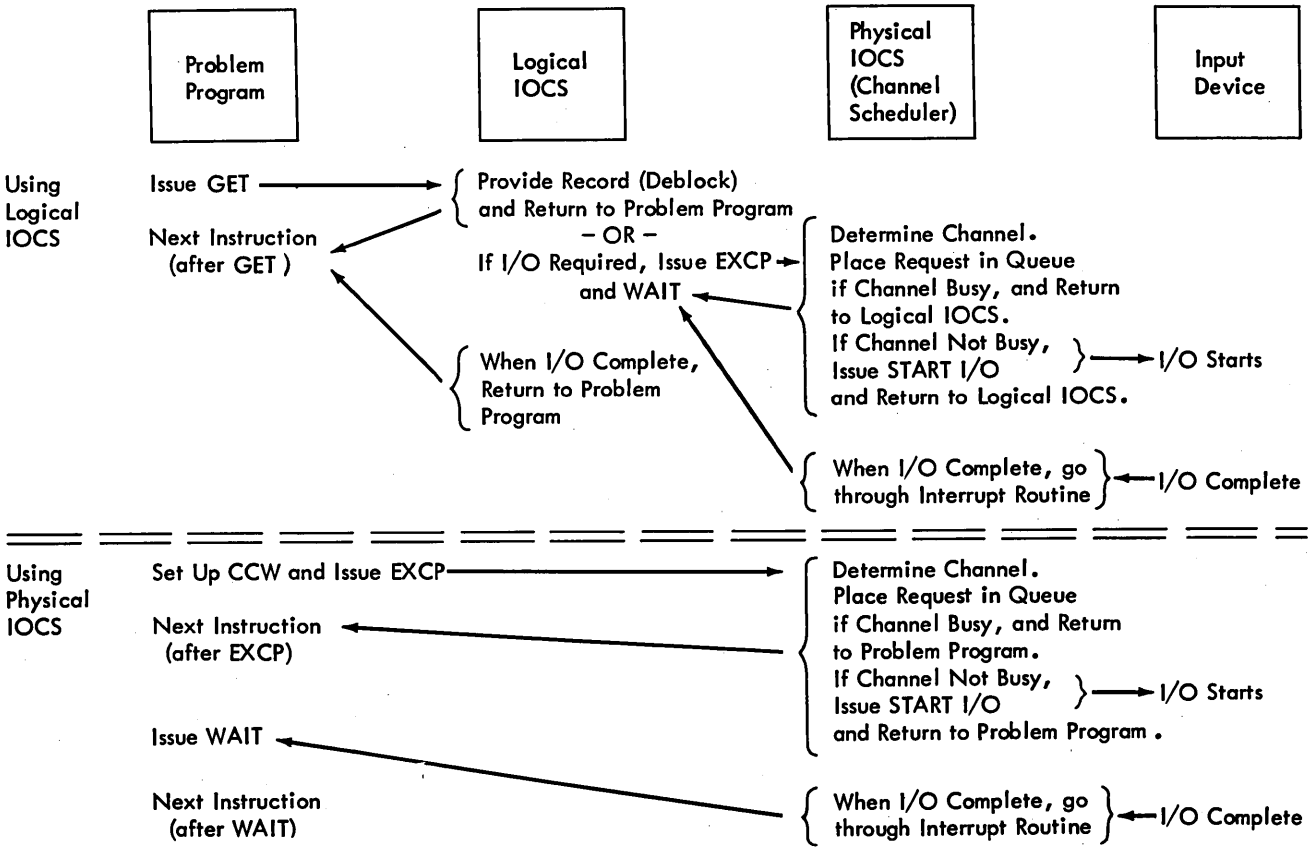


Figure 8. Schematic Example of Retrieving a Record Using Logical IOCS (One I/O Area) or Physical IOCS

instructions overlap data transfer and processing as much as possible. The extent of overlap depends on the user's I/O area assignment. In any case when a GET or PUT has been executed, the transfer of data is complete before the instruction following the GET or PUT is executed.

Direct Access Method (DAM): The direct access method is used to process disk records in a random order. Records stored at any location within the logical file can be processed at any time in the program. IOCS locates a disk record for processing by referring to record-location references supplied by the problem program. The location reference consists of two parts: a track reference and a record reference. The track reference specifies the particular track, or the first of multiple tracks, to be searched for a specified record. The record reference may be the record key, if records contain key areas, or the record ID (identifier), which is available in the count area of each disk record. IOCS seeks the specified track and searches for the specified record on that track, or on that track and on succeeding tracks in the cylinder. Because reference to the records is in random order, all packs of a multipack file must be on-line for any functions performed by the Direct Access Method.

The basic macros used for the direct access method of processing are READ and WRITE. Variations within these macros permit records to be read, written, updated, replaced, or added to a file. Thus, this method provides a means of creating and maintaining a logical file in a random order. When a READ or WRITE instruction is executed, the actual I/O operation is either started or placed in a queue for later execution. Therefore, when another problem program instruction requires that the transfer of data be complete, a test must be made. A WAITF macro is provided for this.

Indexed Sequential File Management System (ISFMS): The indexed sequential file management system is used to process disk records either in random order or in

sequential order by control information. Both orders utilize the control information of the records (such as employee number, part number, customer number, etc.), which must be available in the key area of each disk record. Random processing means that any record stored at any location in the logical file can be processed at any time in the problem program. The user merely supplies IOCS with the key (control information) of the desired record. IOCS searches for the record and makes it available for processing.

In sequential processing, IOCS makes a series of records available, one after the other, in order by the control information (key) in the records. The user specifies which record he wants first. Then IOCS retrieves the succeeding records (on demand) from the logical file, in key order, until the problem program specifies that the operation is to be terminated.

ISFMS provides the means of creating an organized file (loading) and then adding to, reading from, and updating records in that file. The file is organized originally from records that have been pre-sorted by their control information. As the records are loaded onto the disk pack, IOCS constructs indices for the logical file. These indices will permit individual records to be found quickly and easily in subsequent processing operations. The indices are created in such a way that specific records can be retrieved randomly, or all records can be retrieved sequentially. One method can be performed as readily as the other. If records are added to the file at some later time, IOCS updates the indices to reflect the new records.

The basic macros used for the indexed sequential system of processing are READ/WRITE and GET/PUT. READ and WRITE are used for random operations, and GET and PUT are used for sequential operations. When a READ, WRITE, GET, or PUT instruction for a record is executed, the operation is complete before the next instruction in the problem program is executed.

Processing with Synchronous Transmitter Receiver (STR) Devices: Logical IOCS provides macro routines for the transmission of data to, and the reception of data from, an STR device. All STR devices use identical data transmission codes and line control procedures.

Logical IOCS provides READ/WRITE level macro instructions to simplify the use of STR devices by the problem program. Another macro instruction, CNTRL, provides STR line control. The CDCNV (code conversion) macro provides for the conversion of the standard STR transmission code (fixed count four-out-of-eight [4/8]) to or from EBCDIC (used internally in the System/360).

Where the STR devices are attached over a dial network, logical IOCS provides the macro instruction DIALO to aid in establishing the connection.

Binary Synchronous Communication (BSC)

Logical IOCS provides macro support routines for sending and receiving data in a CPU-to-CPU communications environment. Both CPU's use identical line control procedures.

Logical IOCS provides READ/WRITE level macro instructions, which simplify the process of sending data to or receiving data from a remote System/360 by the problem program. The CNTRL macro instruction provides the facility for handling basic BSC line control functions.

Where the CPU's are connected by a dial line, logical IOCS provides the IDIAL macro instruction to handle establishing the connection and to provide the optional facility for ID-verification.

Macro Instructions

Generally two types of macros are required for processing the records in a logical file: one declarative file-definition macro, and one or more imperative macros.

The file-definition macro describes the logical file, indicates the type of processing to be used for the file, and specified main-storage areas for the file. Six file-definition macros are provided for defining files processed by logical IOCS (DTFSR, DTFDA, DTFIS, DTFSN, DTFBS, and DTFRF), and one for disk or tape files processed by physical IOCS (DTFPH). Which of these applies to a particular file is determined by the type of processing used for the file (see Types of Processing). The macros and their use are:

DTFSR — Define The File in SeRial-type device. This macro is used in conjunction with consecutive processing of records in any I/O device.

DTFDA — Define The File for the Direct Access method of processing. This macro is used whenever disk records are to be processed in a random order by the direct access method.

DTFIS — Define The File for the Indexed Sequential system of processing.

This macro is used whenever a file is organized by the indexed sequential file management system and is to be processed by that system.

DTFSN — Define The File for SyNchronous Transmitter-Receiver use. This macro is used for processing in a Tele-processing environment with STR devices.

DTFRF — Define The File for ReFerence. This macro is used with the DTFSN macro and the DTFBS macro to permit the problem program to reference the information in the channel command block (CCB).

DTFBS — Define The File for Binary Synchronous Communication use. This macro is used for processing in a CPU-to-CPU communications environment.

DTFPH — Define The File for processing by PHysical IOCS. This macro is used only if a disk or tape file with standard labels is to be processed by physical IOCS. No other files processed by physical IOCS require definition.

Each file definition macro requires a set of keyword parameter entry cards (Figure 9) to define the file. At assembly time these cards must precede the problem program. The details of these entries are described in the section entitled File Definition Macros. The definition of a file is utilized when an imperative macro instruction (such as GET, PUT, READ, WRITE, etc.) is issued in a problem program statement. If a GET is issued, for example, the file definition supplies such factors as:

- Record type and length
- Input device from which the record is to be retrieved
- Main-storage area where the record is to be located for processing by the problem program.

File definition macros and imperative macros that refer to the file definitions must be assembled together. That is, all file definitions must be assembled with the user's problem program. Imperative macros executed by the Indexed Sequential File Management System (ISFMS) make use of literals, and the literal pool must be addressable any time these macros are executed. Thus the pool must be available in each phase. Furthermore, it must be available in each control section (CSECT) within a phase if base registers are changed between control sections. Therefore, a LTORG statement should be included at the end of each control section whenever disk records are to be processed by ISFMS. If a user defines only one control section, he must include a LTORG statement to ensure the placement of literals at the end of his control section, because multiple control sections are also generated by ISFMS.

Imperative macro instructions are included in the problem program. They perform such functions as opening a file, making records available for processing, writing records that have been processed, etc. The macro instructions provided by IBM for input/output control (Figure 10) are presented in this section in the following groups.

Initialization: OPEN and LBRET

Processing Records Consecutively: GET, PUT, RELSE, TRUNC, READ, WAITF, RDLNE, DSPLY, RESCN, CNTRL, CHNG, and PRTOV

Processing Disk Records by the Direct Access Method: READ, WRITE, WAITF, and CNTRL

Processing Disk Records by the Indexed Sequential System: SETFL, ENDFL, WRITE, READ, SETL, ESETL, GET, and PUT

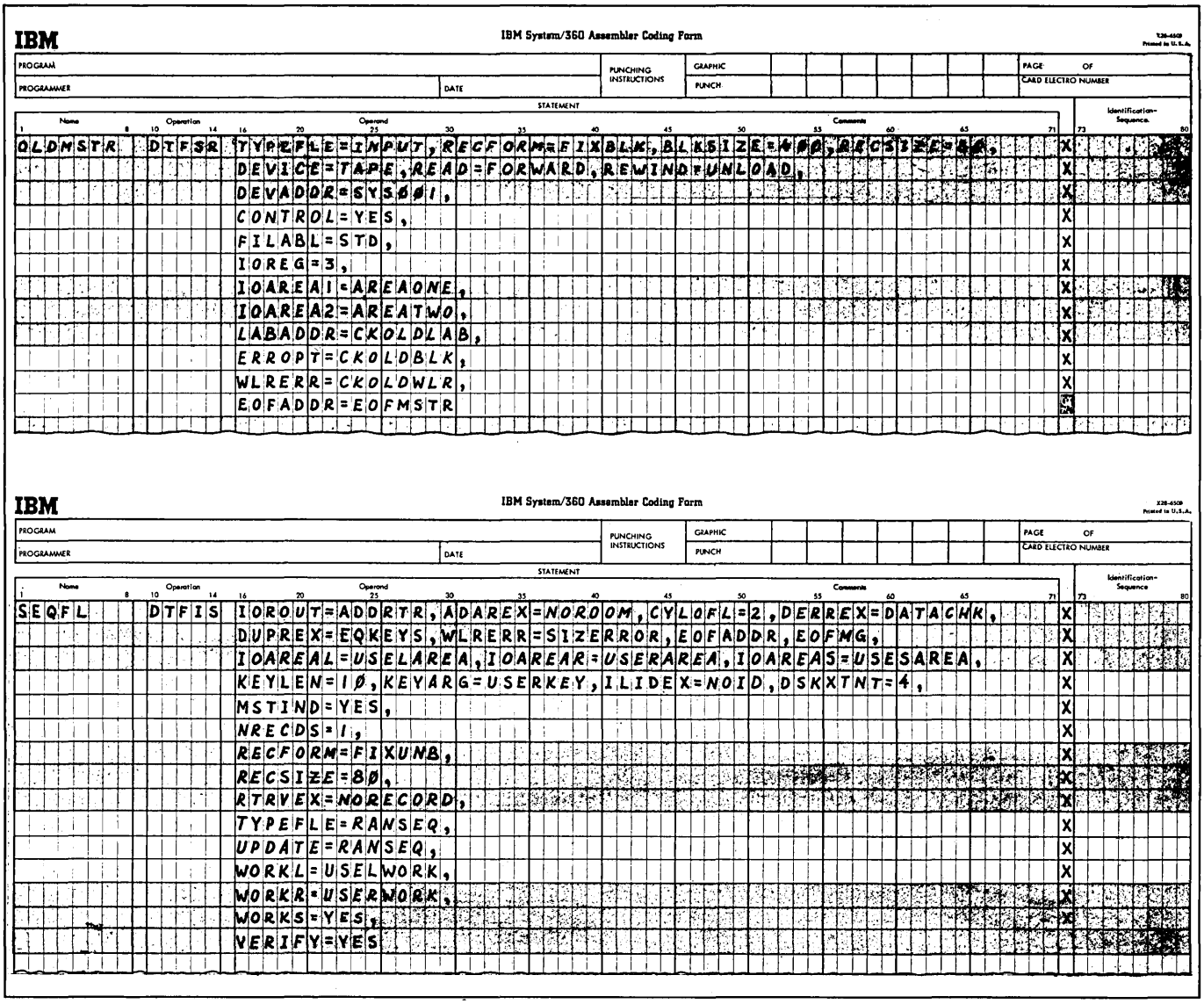
Processing with STR Devices: SOPEN, DIALO, READ, WRITE, CNTRL, CDCNV, SCLOS.

Processing Records by Physical IOCS: CCB, EXCP, WAIT, WAITM, and CHNG

Binary Synchronous Communication: BOPEN, IDIAL, READ, WRITE, CNTRL, BCLOS, and ERRPT.

Writing Checkpoint Records: CHKPT

Completion: CLOSE, LBRET, and FEOV.



```
IBM System/360 Assembler Coding Form

Name      Operation  Operand                                                    Comments
OLDMSTR   DTFSR      TYPEFLE=INPUT,RECFORM=FIXBLK,BLKSIZE=400,RECSIZE=80,      X
                     DEVICE=TAPE,READ=FORWARD,REWIND=UNLOAD,                   X
                     DEVADDR=SYS001,                                           X
                     CONTROL=YES,                                              X
                     FILABL=STD,                                               X
                     IOREG=3,                                                  X
                     IOAREA1=AREAONE,                                          X
                     IOAREA2=AREATWO,                                          X
                     LABADDR=CKOLDLAB,                                         X
                     ERROPT=CKOLDBLK,                                          X
                     WLRERR=CKOLDWLR,                                          X
                     EOFADDR=EOFMSTR                                           X

IBM System/360 Assembler Coding Form

Name      Operation  Operand                                                    Comments
SEQFL     DTFIS      IOROUT=ADDRTR,ADAREX=NOROOM,CYLOFL=2,DERREX=DATACHK,      X
                     DUPREX=EQKEYS,WLRERR=SIZERROR,EOFADDR,EOFMG,              X
                     IOAREAL=USELAREA,IOAREAR=USERAREA,IOAREAS=USESAREA,       X
                     KEYLEN=10,KEYARG=USERKEY,ILIDEX=NOID,DSKXTNT=4,           X
                     MSTIND=YES,                                               X
                     NRECDS=1,                                                 X
                     RECFORM=FIXUNB,                                           X
                     RECSIZE=80,                                               X
                     RTRVEX=NORECORD,                                          X
                     TYPEFLE=RANSEQ,                                           X
                     UPDATE=RANSEQ,                                            X
                     WORKL=USELWORK,                                           X
                     WORKR=USERWORK,                                           X
                     WORKS=YES,                                                X
                     VERIFY=YES
```

Figure 9. Sample DTFSR and DTFIS Macro Instruction

MACRO INSTRUCTION	TYPE OF PROCESSING WITH LOGICAL IOCS																	
	Consecutive										Indexed Sequential System							
	2311 Disk Drive	2400 Magnetic Tape Unit	1442/2501/2520/2540 Reader	1442/2520/2540 Punch	1403/1404/1443/1445 Printer	1052 Printer-Keyboard	2671 Paper Tape Reader	1285 Optical Reader	1287 Optical Reader	Direct Access Method	Load File	Add Records	Random Retrieve	Sequential Retrieve	STR Devices	BSC Support	PHYSICAL IOCS
Initialize																	
OPEN	X	X	X	X	X	X	X	X	X	X	X	X	X	X			X[1]
SOPEN															X		
BOPEN																X	
DIALO															X		
IDIAL																X	
LBRET[2]	X	X								X							X
Process																	
Transfer Records																	
GET	X	X	X[3]			X	X	X	X[10]					X			
PUT	X[4]	X	X[5]	X	X	X								X			
READ									X[11]	X			X		X	X	
WRITE										X	X	X	X		X	X	
RELSE[6]	X	X															
RDLNE								X	X[10]								
TRUNC[7]	X	X															
WAITF									X[11]	X							
EXCP																	X
WAIT															X	X	X
WAITM															X	X	X
CDCNV															X		
DSPLY									X[11]								
RESCN									X[11]								
Set Mode																	
SETFL											X						
ENDFL											X						
SETL														X			
ESETL														X			
Non-Date Operations																	
CNTRL	X	X	X	X	X			X	X	X					X	X	
CHNG[8]		X															X
PRTOV					X												
CCB																	X
Write Checkpoint Records																	
CHKPT	X	X	X	X	X												
Complete																	
CLOSE	X	X	X	X	X	X	X	X	X	X	X	X	X	X			X[1]
LBRET[2]																	X
FEOV																	X[9]
SCLOS															X		
BCLOS																X	
ERRPT																X	

Notes:
1. Required only for disk or tape files with standard labels.
2. Applies only if DTFSR, DTFDA, or DTFPH LABADDR is specified.
3. In the 2540, GET normally reads cards in the read feed. If TYPEFLE=CMBND is specified, GET reads cards at the punch-feed-read station.
4. PUT rewrites an input disk record if UPDATE is specified.
5. In the 1442, 2520, or 2540, PUT punches an input card with additional information if TYPEFLE=CMBND is specified.
6. Applies only to blocked input records.
7. Applies only to blocked output records.
8. Applies only when two selector channels and one or more 2-channel, simultaneous-read-while-write tape control units are installed.
9. Applies only to output tape files with standard labels.
10. Applies only to journal tape processing.
11. Applies only to document processing.

Figure 10. Macro Instructions for Input/Output Control

INITIALIZATION

Before the first record can be read from any input file or transferred to any output file by logical IOCS macro instructions, that file must be readied for use by issuing an OPEN instruction. This applies to logical files in all input/output units available in the system: card readers, card punches, magnetic tape units, disk drives, paper tape reader, printers, optical readers, and display units. When physical IOCS macro instructions will be used for a given file, OPEN is required for that file only if standard labels on disk or magnetic tape are to be checked or written.

Disk Labels

Whenever files of records are written on disk, each disk pack (volume) must contain standard labels to identify the pack and the logical files(s) on the pack. When logical IOCS is used for a file, the IOCS routines read, check, and/or write standard disk labels. When physical IOCS is used, IOCS processes the labels if the DTFPH macro instruction is included in the user's program. The entry TYPEFLE must be specified to indicate whether the file is an input file (read and check labels) or an output file (read and check old labels and write new labels).

The standard labels include one volume label for each pack and one or more file labels for each logical file on the pack. The following paragraphs describe briefly the organization of labels on disk packs. Additional information about disk labels is given in the BOS Programmer's Guide.

Volume Labels: The standard volume label identifies the entire volume (pack) and offers volume protection. It is always the third record on cylinder 0, track 0. The first two records on this track are Initial Program Loading (IPL) records. The volume-label record consists of a count area, a 4-byte key area, and an 80-byte data area. Both the key area and the first four bytes of the data area contain the label identifier VOL1. The remaining 76 bytes of the data area contain other identifying information such as the volume serial number, and the address of the set of file labels for the pack (see Standard File Labels). The volume label is generally written once, when the disk pack is received, by an IBM-supplied utility program.

The standard volume label may be followed by one to seven additional volume labels (starting with record 4 on cylinder 0, track 0). These labels must contain the label identifier VOL2, VOL3 etc in the four-byte key areas and in the first four bytes of the data areas. The other 76 bytes may contain whatever information the user requires. The additional volume labels are also written by the utility program that writes the standard volume label. However, IOCS does not make them available to the user for checking or rewriting when problem programs are executed. These labels are always bypassed by the OPEN routines.

Standard File Labels: The standard file labels identify the logical file, give its location(s) on the disk pack, and offer file protection. The labels for all logical files on a pack are grouped together and stored in a specific area of the pack.

The number of labels required for any one logical file is affected by the file organization (see Standard File-Label Formats) and by the number of separate areas of the pack (extents) used by the file. The data records for a logical file may be contained within one area of the pack, or they may be scattered in different areas of the pack. The limits (starting and ending addresses) of each area used by the file are specified by the standard file label(s).

Because each file label contains file limits, the group of labels on the pack is essentially a directory of all data records on the pack (volume). Therefore, it is known as the Volume Table of Contents (VTOC). The VTOC itself becomes a file of records (one or more standard-label records per logical file) and, in turn, has a label. The label of the VTOC is the first record in the VTOC. This label identifies the file as the VTOC file, and gives the file limits of the VTOC file. The Volume Table of Contents is contained within one cylinder of a disk pack. It does not overflow onto another cylinder.

If a logical file of data records is recorded on more than one disk pack (volume), standard labels for the file must be included in the VTOC of each pack used. The label(s) on each pack identifies the portion of the logical file on that pack and specifies the extent(s) used on that pack.

Standard File Label Formats: All standard-file-label records have a count area and a 140-byte key/data area. Three standard-label formats are provided.

Format 1. This format is used for all logical files, and it has a 44-byte key area and a 96-byte data area. It is always the first of the series of labels when a

file requires more than one label on a disk pack (as discussed in Formats 2 and 3).

The Format 1 label identifies the logical file (by a file name assigned by the user and included in the 44-byte key area), and it contains file- and data-record specifications. It also provides the addresses for three separate disk areas (extents) for the file. If the file is scattered over more than three separate areas on one pack, a Format 3 label is also required. In this case, the Format 1 label points to the second label set up for the file on this pack.

If a logical file is recorded on more than one disk pack, the Format 1 label is always the first label for the file in the VTOC on each pack.

Format 2. This format is required for any file that is organized by the Indexed Sequential File Management System. The 44-byte key area and the 96-byte data area contain additional specifications unique to this type of file organization (such as the highest record in the overflow area).

If an indexed sequential file is recorded on two or more packs, the Format 2 label is used only on the pack containing the indices. This pack may, or may not, contain data records. The Format 2 label is not repeated on the additional packs (as the Format 1 label is).

Format 3. If a logical file uses more than three extents on any pack, this format is used to specify the addresses of the additional extents. It is used only for extent information. It has a 44-byte key area and a 96-byte data area that provide for 13 extents.

The Format 3 label is pointed to by the Format 1 label for the logical file. It is included as required on the first pack, or on additional packs if the logical file is recorded on two or more packs.

User-Standard File Labels: The user may include labels to define his file further, if he desires, provided the file is processed consecutively (DTFSR macro specified), by the direct access method (DTFDA macro specified), or by physical IOCS (DTFPH macro specified). A file organized and processed by the indexed sequential file management system (DTFIS specified) permits standard labels only. A file that is to be processed in consecutive order may have up to eight user-header labels and up to eight user-trailer labels. The trailer labels can be written to indicate an end-of-volume condition. That is, when the end of an extent on one volume (pack) is reached and the next extent is on a different volume, or when the end of the file is reached, user-trailer labels can be included to contain whatever trailer information the user desires (for example, a record count for the completed volume).

User-standard labels are not stored in the Volume Table of Contents. Instead, they are written on the first track of the first extent allotted for the logical-file data records. In this case, the user's data records start with the second track in the extent, regardless of whether the labels require a full track. If a file is written on two or more packs, the labels are written on each of the packs.

All labels must be eighty bytes long and they must contain standard information in the first four bytes. The remaining 76 bytes may contain whatever information the user wants.

The standard information in the first four bytes is used as a record key when reading or writing the labels. The header labels are identified by UHL1, UHL2,---UHL8. The trailer labels, when applicable, are identified by UTL0, UTL1,---UTL7. Each user-label set (header or trailer) is terminated by an end-of-file record (a record with data length 0), which is written by IOCS. For example, if a file has five header labels and four trailer labels, the contents of the user-label track is:

```
R0   Standard information
R1   UHL1--user's 1st header label  )
R2   UHL2--user's 2nd header label  |
R3   UHL3--user's 3rd header label  } DL=80
R4   UHL4--user's 4th header label  |
R5   UHL5--user's 5th header label  )
R6   UHL6--end-of-file record          --DL=00
R7   UTL0--user's 1st trailer label )
R8   UTL1--user's 2nd trailer label |
R9   UTL2--user's 3rd trailer label } DL=80
R10  UTL3--user's 4th trailer label )
R11  UTL4--end-of-file record          --DL=00
```

When files are processed by the direct access method, (or processed by physical IOCS defined with DTFPH with MOUNTD=ALL) only user-standard header labels can be written. In this case, the user-label track contains:

```
R0      Standard information
R1      UHL1--user's 1st header label  )
R2      UHL2--user's 2nd header label  |
          .                            } DL=80
          .                            |
          .                            )
R(n)    UHL(n)--user's nth header      )
        label where ≤8
R(n+1)  UHL(n+1)--end-of-file record   } DL=00
R(n+2)  UTL0--end->-file record        )
```

The user's label routine can determine if a label is a header or trailer label by testing the first four bytes of the label (see OPEN Macro).

Standard Tape Labels

When a tape input or output file that has standard labels is opened, IOCS can handle the label checking (on input) or writing (on output). When logical IOCS macros are used in the program, the DTFSR entry FILABL=STD must be included to specify IOCS-processing of labels. When physical IOCS macros are used, the DTFPH entry TYPEFLE must be included to indicate whether this is an input file (check labels) or an output file (write labels).

If an input tape contains standard labels but the user does not want IOCS to check them, FILABL=NSTD should be specified in the file definition.

The standard labels for a tape file are: a volume label, a file header label, and a file trailer label.

Volume Labels: The standard volume label, which is the first record (eighty characters) on a reel of tape, identifies the entire volume (reel) and offers volume protection. It contains the label identifier VOL1 in the first four positions, and other identifying information such as the volume serial number. This is a unique number generally assigned to the reel when it is first received in the installation. The volume label is generally written once, when the reel of tape is received, by an IBM-supplied utility program. The standard volume label may be followed by a maximum of seven additional volume labels if desired. These must be identified by VOL2, VOL3, etc, in the first four positions of each succeeding label. However, IOCS does not permit the checking or writing of additional volume labels by the user in the problem program. These labels are always bypassed on input.

File labels: The volume label set is followed by a standard file header label. This label (eighty characters) identifies the logical file recorded on the tape and offers file protection. It contains the label identifier HDR1 in the first four positions, and other identifying information such as file identifier, file serial number, creation date, etc. An input tape may contain standard header labels HDR1-HDR8. If so, IOCS checks only label HDR1 and bypasses HDR2-HDR8.

The standard file header label(s) may be followed by a maximum of eight user-written standard labels if desired. If so, the additional file header labels must be identified by UHL1, UHL2, etc. A tape mark follows the last file header label.

A standard file trailer label is located at the end of a logical file (EOF), or at the end of a volume (EOV) if a logical file is continued on another volume. The trailer label has the same format as the header label. It is identified by EOF1 or EOV1 (instead of HDR1) and contains a physical record count (block count). Like the file header label, the standard file trailer label on an input file may be followed by seven additional standard trailer labels identified by EOV2-EOV8 or EOF2-EOF8, whichever is applicable. IOCS bypasses these labels. If desired, the standard trailer label(s) may be followed by a maximum of eight user-written standard trailer labels, which must be identified by UTL1-UTL8.

All user-written standard labels must be eighty characters long and must contain the standard identification in the first four positions. The remaining 76 positions may contain whatever information the user wants. Additional information about tape labels is given in the Programmer's Guide, listed on the front cover of this manual.

Nonstandard Tape Labels

Any tape labels that do not conform to the standard-label specifications are considered nonstandard and, if desired, must be read, checked, or written by the user. On input the nonstandard labels may, or may not, be followed by a tapemark. This choice, combined with the user's requirements to check the labels, or not, results in the following four possible conditions that can be encountered:

1. Label(s), followed by a tapemark, are to be checked.
2. Label(s), not followed by a tapemark, are to be checked.
3. Label(s), followed by a tapemark, are not to be checked.
4. Label(s), not followed by a tapemark, are not to be checked.

For conditions 1 and 2, the DTFSR entries FILABL=NSTD and LABADDR=Name must be specified in the file definition. For condition 3, the DTFSR entry FILABL=NSTD must be specified. DTFSR LABADDR is omitted and IOCS skips all labels, passes the tapemark, and positions the tape at the first data record to be read. For condition 4, the DTFSR entries FILABL=NSTD and LABADDR = Name must be specified. In this case IOCS cannot distinguish labels

from data records because there is no tapemark to indicate the end of the labels. Therefore, the user must read all labels even though checking is not desired. This positions the tape at the first data record.

For output files created by logical IOCS, a tapemark always follows the last nonstandard label, unless the user specifies the DTFSR entry TPMARK=NO.

Unlabeled Tape Files

On input, unlabeled tapes (DTFSR FILABL=NO) may or may not contain a tapemark as the first record. If the tapemark is present, the next record is considered to be the first data record. If there is no tapemark, IOCS reads the first record, determines that it is not a tapemark, and backspaces to the beginning of the first record which it considers to be the first data record. For unlabeled output files (DTFSR FILABL=NO) created by logical IOCS, the first record is always a tapemark, unless the user specifies otherwise. If the user does not wish to have a tapemark written, he must specify DTFSR TPMARK=NO.

OPEN Macro

Name	Op	Operand
	OPEN	filename
	OPEN	filename1,filename2,------

The OPEN macro instruction is used to activate each file that is to be utilized in the problem program. The symbolic name of the logical file (assigned in the DTFSR, DTFDA, DTFIS, or DTFPH header entry) is entered in the operand field of this instruction. As many as 16 files may be opened by one instruction, by entering additional file-name parameters. In this case, the files are opened in the same order as they are specified in the OPEN instruction.

For the card readers, card punches, printers, and paper-tape readers, OPEN simply makes the file available for input or output.

When LIOCS is used for processing journal tapes on the 1285 and 1287 Optical Reader, the OPEN macro must be issued at the beginning of each input roll.

If certain procedures are followed when an end-of-tape condition occurs, it is possible to process two or more rolls on the 1287 as one file. The method is to press the Optical Reader start key (creating an intervention required condition) instead of the end-of-file key to run out this tape. The next tape can then be loaded and processed as a continuation of the previous tape. However, since OPEN is not reissued, no header information can be entered between tapes.

When LIOCS document processing, OPEN must be issued to make the file available. OPEN allows header (identifying) information to be entered at the 1285 or 1287 keyboard, if desired, for journal tape or cut documents (1287). When header information is entered, it is always read into IOAREA1, which must be large enough to accommodate the desired information.

OPEN does not clear the eight binary counters used in the optical readers. These counters are initially zeroed. They accumulate error statistics, as listed under COREXIT=Name, in the section on DTFSR detail entries.

When a magnetic tape file with standard labels (STD specified in DTFSR FILABL) is opened, IOCS expects the first record read to be a label. The first record is a label if the tape file being opened is the first file on the reel and if IOCS rewinds the reel (see DTFSR REWIND). If, however, other specifications are given in DTFSR REWIND or if a file starting in the middle of the reel is opened, it is the user's responsibility to position the tape properly so that the first record read is a label. If the first record is not a label, IOCS indicates an error condition by issuing a message to the operator. An unlabeled file (DTFSR FILABL=NO) can, however, be opened in the middle of data records without causing an error condition. If a file with nonstandard labels (DTFSR FILABL=NSTD) is opened, all label processing is the responsibility of the user's routines (see Initialization:Nonstandard Tape Labels).

Whenever an input/output disk or tape file is to be opened and the user plans to process user-standard labels or nonstandard labels, he must provide the information for checking or building the labels. If this information is obtained from another input file, that file must be opened ahead of the disk or tape file. This is done by specifying the input file ahead of the disk or tape file in the same OPEN instruction, or by issuing a separate OPEN instruction preceding the OPEN for the disk or tape file.

The specific functions that occur on an OPEN for a disk or tape file vary with the type of operation (input or output) and with the use of file labels. These

functions are discussed in the following sections.

When building an output file and IOREG is specified, OPEN positions IOREG to the beginning of IOAREA1.

In addition to the registers used by logical IOCS, OPEN also uses register 5. The programmer may use register 5 because the OPEN macro routine saves and restores this register. However, if the programmer plans to use register 5 as a base register, he should be aware that register 5 is dropped at the end of the OPEN routine.

For a discussion of reopening a file after it has been closed, see CLOSE Macro.

For a discussion of SOPEN, to initialize the adapter for STR devices, see: Processing with STR Devices. For a discussion of BOPEN, to initialize the data adapter for BSC, see: Binary Synchronous Communication.

Disk Input File

When an input file that is recorded on disk is opened, the OPEN routines:

- Check the standard label(s) for the file.
- Make any user-standard labels available to the user for checking, if the DTFSR, DTFDA, or DTFPH entry LABADDR is included in the file definition. The indexed sequential system (DTFIS) does not permit user-standard labels.
- Locate the area(s) of the disk pack (extent) where the file is written.
- Make the file records available for processing.

For these functions, the OPEN routines refer to the information supplied by the user in Job Control VOL, XTENT, and DLAB cards. A VOL card and a DLAB card must be supplied for each logical file, and an XTENT card must be supplied for each separate area (in each volume) used by the file.

The functions of the OPEN routines may occur at different times during the job, depending on the type of processing specified for the file by the file-definition macro instruction. If a file is to be processed consecutively (DTFSR specified), OPEN initially checks the standard label(s) on the pack or on the first pack of a multipack file, makes any user-standard labels on the first pack available for checking, and then locates and makes available the first extent on the first pack. IOCS processes one extent at a time, in the sequence specified by the user's control cards. When IOCS detects the end of the current extent, it branches to the end-of-extent routine. OPEN then locates the next extent specified by the control cards, makes sure it is on-line and ready, and makes it available for processing. If the next extent is the first extent of a different pack used by the file, OPEN checks the standard labels on that pack ans makes any user labels on that pack available to the user for checking. If the user has included user-trailer labels for a consecutive file, they are made available for checking when the last extent on one pack is completed and before the first extent on the next pack is opened (see Completion: Disk Input File).

When a file is to be processed by the direct access method (DTFDA specified) or by the indexed sequential system (DTFIS specified), all disk areas specified by theuser's control cards for the file must be mounted and ready when the file is first opened. Therefore, the OPEN routines initially check all standard labels on all packs used by the file, make all user labels (if any) available for checking, and check all specified extents and make them available for processing. For a multi-pack file, OPEN processes the standard labels followed by all user labels on the first pack, then processes standard labels on the second pack, etc. Therefore, the user's label routine is entered for each user label on each pack in turn, after the standard label on that pack has been checked.

If a file is to be processed by physical IOCS (DTFPH specified) and if the DTFPH entry MOUNTD=SINGLE is specified, the file is treated as consecutive at the initial opening. That is, only the pack containing the first extent specified by the user's control cards must be on-line when the initial OPEN is issued. The labels on this pack are processed, and the first specified extent is made available for processing. Thereafter the user must keep track of the extents and issue an OPEN for each succeeding extent when it is required for processing. Each additional time that OPEN is issued for the file, IOCS locates and makes available the next extent specified by the user's control cards. If the DTFPH entry MOUNTD=ALL is specified, all extents specified for the file must be mounted and ready when the initial OPEN is issued. All labels are checked (or made available for user-checking) and all extents are checked and made available for processing. Only one OPEN is issued for the file, the function for MOUNTD=ALL are similar to

those performed when a direct access or indexed sequential file is opened.

Checking User-Standard Labels: When a disk file contains user-standard labels, the programmer can check them in his own routine if the DTFSR, DTFDA, or DTFPH entry LABADDR is included in the file definition. The OPEN routines branch to the user's routine after each user's label has been read. IOCS reads each label into the label read-in area. The address of this area is supplied to the user in register 1 upon entry into his routine. The user's routine can identify the various labels (and distinguish between header and trailer labels) by testing bytes 1-4 of the read-in area. After checking each label, the user must return to the OPEN routines by use of the LBRET macro.

Disk Output File

When an output file that is to be recorded on disk is opened, the OPEN routines:

- Audit the extents specified by control cards to make sure that any data previously recorded is no longer active and may be destroyed.
- Create and write required standard label sets.
- Permit the user to create user-standard labels and write those labels, if the DTFSR, DTFDA, or DTFPH entry LABADDR is specified. The indexed sequential system (DTFIS) does not permit user-standard labels.
- Locate the area(s) of the disk pack where the records are to be written.
- Make the area(s) available to logical IOCS.

Similar to the opening functions for an input file, the OPEN routines refer to the information supplied by the user in the Job Control VOL, XTENT, and DLAB cards for the output file. For the creation of the standard labels, OPEN also uses information supplied by the file-definition macro (DTFSR, DTFDA, DTFIS, or DTFPH).

As each pack is opened for a file, IOCS constructs and writes the standard label (s) for that pack. For consecutive processing (DTFSR specified) only the first pack is opened at the initial OPEN. After the first, each other pack is opened and the standard labels are written when, during execution, the processing of records for one pack is completed and the file is to be continued on a different pack. For the direct access method (DTFDA) or the indexed sequential system (DTFIS), all packs are opened and all standard labels are written at the initial OPEN for the logical file.

After the standard labels are written for any pack, IOCS branches to the user's label routine (if one is specified by the DTFSR, DTFDA, or DTFPH entry LABADDR) and the user can construct his label(s).

The extents specified by the user are checked in turn. That is, they are checked as they are ready to be written when processing consecutively. Or, they are checked, one after the other, at the initial OPEN when the direct access method or the indexed sequential system is used. This checking is based on the expiration dates in the existing standard file labels. If any files are to be partially or wholly destroyed, their standard labels are removed from the Volume Table of Contents. This, in effect, deletes the entire file from this pack.

If an output file is to be processed by physical IOCS, both the initial and subsequent opening functions are similar to those described for an input file (see OPEN Macro: Disk Input File), except that labels are written rather than checked. Thus, the file is treated like a consecutive file if DTFPH MOUNTD=SINGLE is specified. It is treated like a direct access or indexed sequential file if DTFPH MOUNTD=ALL is specified.

Writing User-Standard Header Labels: When the user specifies that user-standard labels are to be written (by including the DTFSR,DTFDA, or DTFPH entry LABADDR), the user constructs the labels and IOCS writes them. The OPEN routines branch to the user's routine after the standard label(s) is written. OPEN prepares for the user's label(s) by setting up a label area where the user can construct the label(s), supplying the address of the area in register 1, placing UHL1 (for the first label) in the first four bytes of the area, and storing a return address in register 14. In his routine the user constructs a 76-byte label and returns to IOCS by use of the LBRET macro. IOCS checks for a set-completed indication, writes the label, determines if eight labels have been written and, if not, increases the UHL identification by 1 and returns to the user's label routine. Because a maximum of eight user-standard labels is permitted, IOCS automatically terminates the label set after a label with UHL8 is written. If the user requires fewer labels, he can force the end of the label set by issuing the LBRET macro with the operand "1". Whenever the DTFSR, DTFDA, or DTFPH entry LABADDR is specified, at least one additional label must be written. Upon return to IOCS, the

set is terminated and IOCS writes an end-of-file record.

Tape Input File

When an input file is recorded on magnetic tape, OPEN rewinds the tape according to the specifications in the DTFSR entry REWIND. (No rewind is performed if the file is defined by DTFPH.)

Standard Labels: Both the first volume label (VOL1) and first file header labels (HDR1) are automatically read and checked if standard label checking is specified (STD specified in DTFSR FILABL or INPUT specified in DTFPH TYPEFLE) and if the tape is read forward (FORWARD specified in DTFSR READ). The fields are checked with the information supplied by the Job Control VOL and TPLAB cards.

By this label checking, IOCS locates the file to be processed if more than one file is written on a tape reel (multifile reel). For this, IOCS compares the file sequence number in the label with that in the TPLAB card. The file sequence number gives the sequential position of the file on the reel. For example, if the first file on a multifile reel has file sequence number 1, the third file has file sequence number 3. If the first file is numbered 15, the third file is numbered 18. The OPEN routines bypass all files until a header label with the matching file sequence number is read, or until the end of the tape is reached. Several files on a reel may be processed in succession without rewinding the tape if the file sequence numbers are specified in ascending sequence. If not, the tape must be rewound before the file to be processed is opened. If the tape is positioned beyond the desired file when the OPEN for that file is executed, message 4114A is given to the operator.

If a tape file that will be read backwards (BACK specified in DTFSR READ) is opened, the file trailer label is automatically read and checked if label checking is specified. The volume label is not repeated at the end of the tape. Because the file trailer label is processed at this time, it must be complete and contain both the trailer and header information (except HDR) to identify the file. If the file labels were originally written by IOCS routines, the trailer label will be complete. If the tape is not positioned at the trailer label (EOF1) when the file is opened, the user is notified and reading continues. This situation is possible if the user begins reading backwards in the middle of his file. When physical IOCS macros are used to read records backwards, labels cannot be checked and the file must not be defined with DTFPH statements.

If a tape contains user-standard header labels (UHL1-UHL8) following the standard file header label (or user-standard trailer labels, UTL1-UTL8, preceding the standard trailer when reading backwards), the programmer can check them in his own routine. The OPEN routine branches to the user's routine (identified by the DTFSR or DTFPH entry LABADDR) after each user's label has been read. Each label is read into the label read-in area used by IOCS. The address of this area is supplied in register 1 upon entry into the user's routine. After checking each label, the user must return to the OPEN routine by useof the LBRET macro. If user-standard labels exist but the user does not specify LABADDR, the user labels are bypassed by IOCS.

When the tapemark at the end of the labels is read, IOCS opens the next file specified in the OPEN macro, or returns control to the problem program if all files have been opened.

After the labels (if any) for a file have been processed, that file is ready for the first GET instruction.

Nonstandard Labels: To process nonstandard labels, the user must specify FILABL=NSTD and LABADDR=Name, and he must define his own label read-in area. To read the nonstandard labels, physical IOCS macro instructions must be used instead of logical IOCS instructions. A Command Control Block (CCB) and a Channel Command Word (CCW) must be established, and an EXCP instruction must be issued for each label must be issued for each label record (see Processing Records with Physical IOCS).

In his label routine, the user issues the EXCP and WAIT macro instructions and then performs whatever checking he desires for the labels. After all labels have been read and processed, the user returns control to the OPEN routines by use of the LBRET macro.

If a file with nonstandard labels utilizes an alternate tape drive (DTFSR ALTTAPE=SYSnnn), IOCS supplies a code identifying the symbolic unit (see Figure 26) of the drive currently being used in the two low-order bytes of register 1. This value must be moved to bytes 4 and 5 of the Command Control Block (CCB) used by the EXCP macro for label reading.

Tape Output File

For a magnetic tape output file, OPEN rewinds the tape as specified in the DTFSR

entry REWIND. (No rewind is performed if the file is defined by DTFPH.)

Standard Labels: When standard labels are to be written (STD specified in DTFSR FILABLE, or OUTPUT in DTFPH TYPEFLE), the volume label is checked and the old file header is read and checked to make sure that the data on the tape is no longer active and may be destroyed. The tape is then backspaced and the new file header label is written with the information supplied by the Job Control TPLAB card. The volume label is not rewritten.

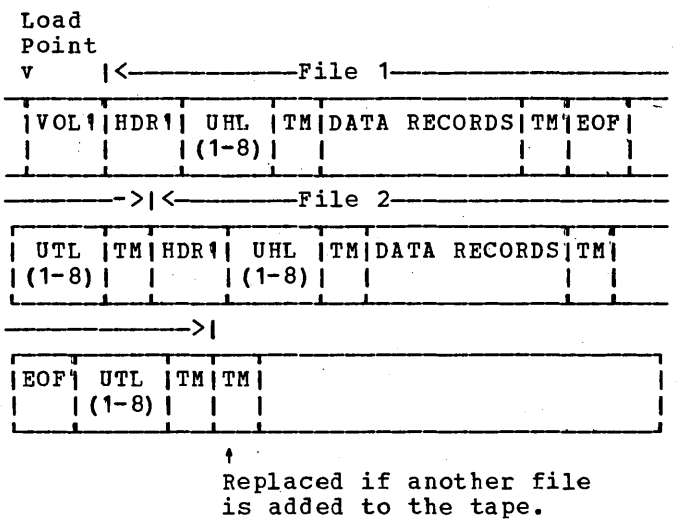
If user-standard header labels (UHL1-UHL8) are to be written following the standard header label, the OPEN routine branches to the user's routine (specified by the DTFSR or DTFPH entry LABADDR) after each standard label. In his routine, the programmer can build a maximum of eight user-standard labels. Each label must be built in the label output area used by IOCS. The address of this area is supplied to the user in register 1 upon entry into his routine. IOCS also supplies the letter O in the low-order byte of Register 0, to indicate that a header label should be built. After building each label, he must return to the OPEN routine by use of the LBRET macro. Then IOCS writes the label. When the user determines that the last user label has been written, he must issue the LBRET macro with the operand "1". Whenever the DTFSR or DTFPH entry LABADDR is specified, at least one additional label must be written.

After the header labels (if any) for a file have been written, the tape is ready for the first PUT instruction for that file.

Multifile Reels. More than one file of records may be written on one reel of tape (multifile reel), if desired. If this is planned, the DTFSR entry REWIND=NORWD should be specified for each file. With this specification, the tape is located at the correct position for the OPEN routines to write the standard file header label for each additional file (after the first) on the reel. For the first file, the programmer can include a CNTRL macro instruction (with the operand REW) ahead of the OPEN instruction. Or, the operator can position the tape at the load point.

For the standard file header label of each file after the first, the OPEN routines obtain the file serial number, volume sequence number, and file sequence number from the preceding standard trailer label. They increase the file sequence number by 1 for the new file. OPEN writes the remaining fields of the header label by using the information supplied in the corresponding Job Control TPLAB card. As with the first file on the reel, user-standard header labels may follow the standard header label.

If a tape is rewound or repositioned after a file is closed, it is the user's responsibility to properly position the tape for writing any additional file. The tape must be positioned so that the file header label is written immediately after the tape mark that follows the last file currently on the tape. The header label will replace the second tape mark that normally follows the trailer label(s) of the last file on a tape (see CLOSE Macro: Tape Output File). The tape can be advanced from the load point to the correct position by skipping three tape marks for each file presently on the tape. The layout of tape records is:



A Job Control FILES card should be used to skip the required number of tape marks. When the tape has been positioned and the file is opened, the OPEN routines obtain the information for the standard file header label from the preceding standard trailer label and the Job Control TPLAB card, as described in the preceding paragraphs. The DTFSR entry REWIND=NORWD must be included for this file.

Nonstandard Labels: To process nonstandard labels, the user must specify FILABL=NSTD and LABADDR=Name, and he must define his own label read-out area. To write the nonstandard labels, physical IOCS macro instructions must be used instead of logical IOCS macro instructions. A Command Control Block (CCB) and a Channel Command Word (CCW) must be established, and an EXCP instruction must be issued for each label record (see Processing Records with Physical IOCS).

Upon branching to the user's label routine, IOCS supplies the letter O in the low-order byte of Register 0 to indicate that a header label(s) should be written (see DTFSR LABADDR). In his routine, the user issues the EXCP and WAIT macro instructions after he has built each label record. After all labels have been written, the user returns control to the OPEN routines by use of the LBRET macro.

If a file with nonstandard labels utilizes an alternate tape drive (DTFSR ALTTAPE=SYSnnn), IOCS supplies a code identifying the symbolic unit (see Figure 28) of the drive currently being used in the two low-order bytes of register 1. This value must be moved to bytes 4 and 5 of the Command Control Block (CCB) used by the EXCP macro for label writing.

LBRET Macro

Name	Op	Operand
	LBRET	1
	LBRET	2

The LBRET (label return) macro instruction applies only to disk or tape files that contain user-standard labels, or nonstandard labels, that the user wants to check or build/write. It must be issued at the end of the user's label routine (specified by the DTFSR, DTFDA, or DTFPH entry LABADDR), to return to IOCS after header or trailer labels have been processed. This instruction requires one of the following operands:

Operand 1

User-Standard Labels, Input File: To return to IOCS when the user wants to eliminate the checking of one or more user-standard labels. IOCS then skips the remaining labels in the set, and processing continues. If all labels are to be checked, operand 1 is not used and IOCS terminates label processing when the disk end-of-file record or the tapemark following the last label is read.

User-Standard Labels, Output File: To return to IOCS when the user determines that the last user-standard label has been built. IOCS writes the last label (from the label output area) and processing continues. Operand 1 is always required to terminate the output label set.

Nonstandard labels: Operand 1 is invalid for files that contain nonstandard labels (FILABL=NSTD).

Operand 2

User-Standard Labels, Input File: To return to IOCS after each user-standard label has been checked. IOCS makes the next label, if any, available for checking in the label input area. When IOCS senses the endof the label set (disk end-of-file record or tapemark), it terminates label processing.

User-Standard Labels, Output File: To return to IOCS after each user-standard label except the last has been built. IOCS writes the label from the label output area and returns to the user's label routine to permit him to build his next label. The label set is terminated by LBRET 1.

Nonstandard labels: To return to IOCS after all nonstandard labels have been checked or written. For nonstandard labels, IOCS branches to the user's label routine only once, and the problem program must read or write every required label before issuing LBRET to return to IOCS.

The LBRET routine requires the values that the IOCS has placed into registers 14 and 15. Hence, if the user requires one or both of these registers in his routine, he must save the value placed into these registers by the IOCS before he starts using them. He must restore this value prior to issuing the LBRET macro instruction.

PROCESSING RECORDS CONSECUTIVELY

Records in serial-type devices (such as card reader, tape unit, printer) and records in a 2311 disk file used in a serial-type order can be processed consecutively. In this type of processing, successive records are processed, starting at the beginning of a logical file and continuing, one record after the other, to the end of the file (see Types of Processing: Consecutive Processing). In this method the user issues GET or PUT macro instructions to transfer records.

Whenever a file of records is to be processed in consecutive order, the logical file, the device used for the file, and the

main-storage areas allotted to the file must be defined by the declarative macro DTFSR (Define The File in a SeRial-type device). The detail parameter entries for this definition are described under File Definition Macros.

Record Types

IOCS handles records that are:

- Blocked - two or more logical records in one physical record, such as a tape record
- Unblocked - one logical record per physical record
- Fixed-Length - all records the same length
- Variable-Length - the records differ in length
- Undefined - the record characteristics are unknown to IOCS.

IOCS can process all the different types in the same program. However, all the records in a given file must be the same type, and this must be defined in the DTFSR entry RECFORM for that file. The types of records that can be processed vary with the type of I/O device used for reading/writing the file of records, as shown in the illustration (Figure 11).

When an application using blocked records is planned, the number of records that can be allocated for a block depends on the size of the records and the amount of main storage that can be reserved for the block. The programmer must predetermine the maximum block size and specify this in the DTFSR entry BLKSIZE. All records within the block may be fixed length or variable length. If the records are fixed length, the length of the records is specified in DTFSR RECSIZE.

If the blocked records are variable-length, the size of each record must be included within the record itself (Figures 12 and 13). This record-length field must occupy the first four bytes of each record. The first two bytes specify the length of the record (including the four bytes for the record-length field itself), and the next two are blank. In addition, the actual length of each block must be recorded on disk or tape, preceding the first record in the block. Block length is also a four-byte field. The first two bytes specify the length of the block (including the four bytes for the block-length field itself), and the next two are blank. Both block length and record length are expressed in 16-bit binary form.

TYPE OF I/O DEVICE	TYPES OF RECORDS				
	Fixed-Length		Variable-Length[1]		
	Unblocked	Blocked	Unblocked	Blocked	
2311 Disk Storage Drive	X	X	X	X	X
2400 Series Magnetic Tape Unit	X	X	X	X[2]	X
1442, 2501, 2520, 2540 Card Reader	X				
1442, 2520, 2540 Card Punch	X		X		X
1403, 1404, 1443, 1445, Printer	X		X		X
1052 Printer-Keyboard	X				X
2671 Paper Tape Reader	X				X[3]
1285 Optical Reader	X				X
1287 Optical Reader	X				X

NOTES:
1. For disk or tape records, each record must contain a record-length field and each block must contain a block-length field. In the case of unblocked records, block length is a block of 1 record and it equals record length +4.
2. Read backwards must not be specified.
3. Each record must contain an end-of-record character as the last character in the record.

Figure 11. Types of Records and I/O Devices for Consecutive Processing

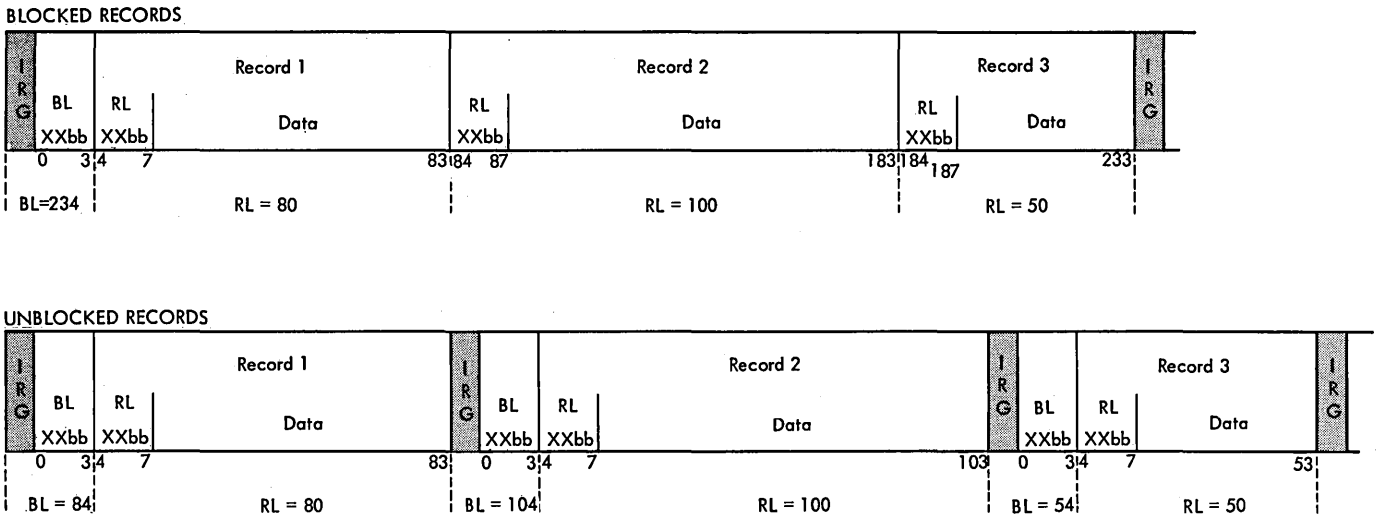


Figure 12. Schematic of Variable-Length Records on Tape

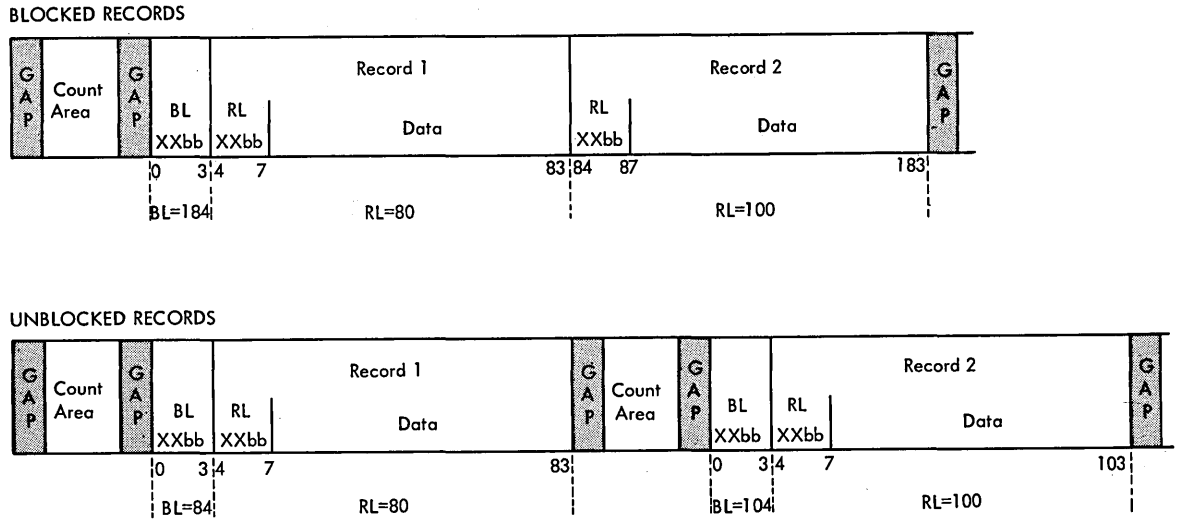


Figure 13. Schematic of Variable-Length Records on Disk (Consecutive Processing)

When unblocked variable-length records on disk or tape are processed, each record must contain the record-length field in the first four bytes, the same as for blocked variable-length records. Also, a block-length field must precede each record (a block of one record). This four-byte field always specifies the record length plus 4.

Because a block-length field must precede the first data record whenever variable-length records (either blocked or unblocked) are processed, this must be included in the BLKSIZE specification and in the amount of main storage allotted to the input/output area.

When unblocked variable-length records are specified for the card punch or printer, a record-length field must be included in the first four bytes of each output record in main storage . The first two bytes must specify the length of the record (including the four bytes for the record-length field itself), and the next two are blank. The user must supply the record length in this field when he builds the record. Punching, or printing, of the actual data record starts with the first position after the record length field, unless a control character is included in the record (see PUT Macro: Punch and Printer Control). The amount of main storage allotted for the punch or printer output area must allow for the 4-byte record-length field, and the DTFSR BLKSIZE specification must include these four bytes.

When undefined records are to be read or written, the DTFSR entry RECSIZE must specify a register. On input, IOCS supplies the physical record size in this register. For output, the programmer must load the length of each record in this register before he issues the PUT for that record.

Record Sizes: The minimum size physical tape record (gap to gap) that can be handled is 12 characters (11 characters or less are considered a noise record). The maximum size tape record is 32K. The maximum size input/output record for the IBM 1052 Printer-Keyboard is 256 characters; the minimum size input record is one character. The maximum size card or printer record cannot exceed the capacity of the corresponding I/O unit.

Storage Areas

When logical IOCS macro instructions are used, each input record can be made available to the program for processing either in an input area or a work area. Similarly, on output, each record can be built in a work area or directly in an output area.
Input/output areas and work areas for a particular file can be specified and handled by IOCS in any of the following combinations:

1. One I/O area
2. One I/O area and one work area
3. Two I/O areas
4. Two I/O areas and one work area.

If one I/O area (combination 1) is used, a register must be specified in DTFSR IOREG whenever blocked records are processed or unblocked variable-length records are read backwards. The register is used to point to the beginning of each record and thus locate the record for processing. A register must also be specified whenever two I/O areas (combination 3) are used, regardless of whether the records are blocked or unblocked. If the blocked records are variable length and are being built in the output area(s), an additional register must be specified in DTFSR VARBLD. This register provides the programmer with the remaining space in the output area each time a PUT instruction is executed.

Whenever a work area (combinations 2 and 4) is used, a register is not required and IOREG should be omitted. Instead, DTFSR WORKA must be specified and the work area must be named in each GET or PUT instruction. The various combinations are discussed further in the BOS Programmer's Guide, as listed on the front cover of this publication.

GET Macro

Name	Op	Operand
	GET	filename
	GET	filename,workname

This instruction makes the next consecutive logical record from an input file available for processing in either an input area or a specified work area. It is used for any input file in the system, and for any type of record: blocked or unblocked, fixed or variable length, and undefined. When the GET macro detects an end-of-file condition, IOCS branches to the user's end-of-file routine (specified by DTFSR EOFADDR).

The GET macro instruction is written in either of two forms, depending on the area where the records will be processed. Either form, but not both, can be used for one DTFSR-specified logical file. The

first form is used if records are to be processed directly in the input area(s), and it requires only one parameter. This parameter specifies the name of the file from which the record is to be retrieved. The file name must be the same as the one specified in the DTFSR header entry for this file.

The input area must be specified in the DTFSR entry IOAREA1. Two input areas may be used to permit an overlap of data transfer and processing operations. The second area is specified in DTFSR IOAREA2. Whenever two input areas are specified, the IOCS routines transfer records alternately to each area. They completely handle this "flip-flop" so that the next consecutive record is always available to the problem program for processing.

When records are processed in the input area(s), a general purpose register must be specified in the DTFSR entry IOREG if:

1. Records are blocked,
2. Variable-length unblocked tape records are read backwards, or
3. Two input areas are used, for either blocked or unblocked records.

This register identifies the next single record to be processed. It always contains the absolute base address of the currently available record. The GET routine places the proper address in the register.

The second form of the GET instruction is used if records are to be processed in a work area. It causes the GET macro to move each individual record from the DTFSR-specified input area to a work area. As in the first form, the file name must be entered as the first parameter. The name of the work area must be entered as the second parameter, and YES must be specified in the DTFSR entry WORKA. The work-area name must be the same as that specified in the DS instruction that reserves this area of main storage . All records from a logical file may be processed in the same work area, or different records from the same logical file may be processed in different work areas. In the first case, each GET for the file specifies the same work area. In the second case, different GET instructions specify different work areas. It might be advantageous to plan two work areas, for example, and to specify each area in alternate GET instructions. This would permit the programmer to compare each record with the preceding one, for a control change. Only one work area can be specified in any one GET, however. Whenever this form of the GET instruction is used for a logical file, a register is not required for indexing (as it is when records are processed directly in the input area).

The GET macro is used to acquire data from journal tapes from 1285/1287 optical reader files. Processing overlap is obtained by using multiple input areas.

IBM 2671 Paper Tape Reader: Whenever a GET instruction is issued for the 2671, bits 6 and 7 of main-storage location filename+16 should be tested to determine if an error is detected when a record is read. If a data check or equipment check is detected, an error message is issued to the operator. The operator's reply to this message determines if bit 6 or 7 is turned on.

If bit 6 is on, the record has been read, but one or more characters in the record are in error. Bit 6 is turned on by an operator reply (4) to continue reading the record and ignore the character that caused the error.

If bit 7 is on, the operator has positioned the tape either at the beginning of the same record or at the beginning of the next available record. Bit 7 is turned on by an operator reply (other than 0-5) either to reread or ignore the record containing the error. The next GET attempts another read operation on the current record or reads the next available record. If a torn tape prompts the operator to advance the tape, more than one record may be lost.

As another choice for a data check, the operator may backspace two characters and attempt another read operation (reply 5). If this read is correct, no data check has been detected and processing continues. Neither bit 6 nor 7 is turned on. If an error is detected, however, the operator may reply with a 4 that turns on bit 6, or with any character other than 0-5 that turns on bit 7.

Unblocked Records

Records retrieved from any input file except disk or magnetic tape are always considered unblocked (specified as unblocked or undefined). Records on disk or tape are treated as unblocked if this is specified in the DTFSR entry RECFORM.

Whenever records are unblocked (either fixed or variable length) and only one input area is used, each GET transfers a single record from an I/O device to the input area, and then to a work area if one is specified in the GET instruction. If two input areas are specified, each GET makes the last record that was transferred

to main storage available for processing in the input area or work area. The same GET also starts the transfer of the following record to the other input area.

When an IBM 2540 Card Read-Punch is used for a card input file, each GET instruction normally reads the record from a card in the read feed. However, if the 2540 has the punch-feed-read special feature installed and if CMBND is specified in the DTFSR entry TYPEFLE, each GET reads the record from a card in the punch feed, at the punch-feed-read station. This record can be updated by additional information and punched back into the same card, when that card passes the punch station and a PUT instruction is issued. (See PUT: Updating.)

On the IBM 1285 and 1287 Optical Readers, fixed unblocked records should be used when processing journal tapes containing lines with an equal number of characters. For documents on the 1287, processing is similar to that of undefined records, because the fields read may be treated as either fixed- or variable-length by setting the suppressed length indicator (SLI--flag bit 34) in the CCW used to read the document field. However, the user is not required to use RECSIZE and IOREG, and can eliminate these two registers if he desires.

Blocked Records

When records on disk or tape are specified as blocked in the DTFSR entry RECFORM, each individual record must be located for processing (deblocked). Therefore, blocked records (either fixed or variable length) are handled as follows:

1. The first GET instruction transfers a block of records from disk or tape to the input area. It also initializes the specified register to the absolute address of the first data record, or it transfers the first record to the specified work area.

2. Subsequent GET instructions either add an indexing factor to the register or move the proper record to the specified work area, until all records in the block have been processed.

3. Then the next GET makes a new block of records available in main storage , and either initializes the register or moves the first record.

Undefined Records

When undefined records are to be handled, the DTFSR entries RECFORM=UNDEF and RECSIZE=n must be included in the file definition. GET treats undefined records as unblocked, and the programmer must locate individual records and fields. Undefined records are considered to be variable in length by IOCS. No other characteristics of the record are known by IOCS. They are the responsibility of the user.

Journal tapes containing lines of variable character length are processed in this manner on the 1285 and 1287. When documents specified as undefined are handled by the 1287 reader, the entry RECSIZE pertains only to the last field read by the channel command word chain.

Read Backwards, Tape

If records on tape are to be read backwards (BACK specified in DTFSR entry READ), unblocked records or blocks of fixed-length records are transferred from tape to main storage in reverse order. The last block is read first; the next-to-the-last block, second; etc. For blocked records, each GET instruction also makes the individual records available in reverse order. The last record in the input area is the first record available for processing (either by indexing or in a work area). Variable-length blocked records cannot be read backwards.

Nine-track tape can be read backwards without qualification, but 7-track tape can be read backwards only if:

- the tape was originally written on a magnetic tape unit of the IBM System/360,
- the Data Conversion special feature was not used when the tape was written, and
- a tape mark was written at the beginning of the tape preceding the data records.

PUT Macro

Name	Op	Operand
	PUT	filename
	PUT	filename,workname

This instruction writes, punches, or displays logical records that have been built directly in the output area or in a specified work area. It is used for any output file in the system, and for any type of record: blocked or unblocked, fixed or variable length, and undefined. It operates much the same as GET but in reverse. It is issued after a record has been built.

Similar to GET, the PUT macro instruction is written in either of two forms, depending on the area where the records are built. Either form, but not both, can be used for one DTFSR-specified logical file. The first form is used if records are built directly in the output area(s), and it requires only one parameter. This parameter specifies the name of the file to which the record is to be transferred. The file name must be the same as the one specified in the DTFSR header entry for this file.

The output area must be specified in the DTFSR entry IOAREA1. Two output areas may be used to permit an overlap of data transfer and processing operations. The second area is specified in DTFSR IOAREA2. Whenever two output areas are specified, the IOCS routines transfer records alternately from each area. They completely handle this "flip-flop" so that the proper output-record area is always available to the program for the next consecutive output record.

When records are built in the output area(s), a general-purpose register must be specified in the DTFSR entry IOREG if:

1. Records are blocked, or
2. Two output areas are used, for either blocked or unblocked records.

This register always contains the absolute base address of the currently available output-record area. The PUT routine places the proper address in the register.

The second form of the PUT instruction is used if records are built in a work area. It causes the PUT macro to move a record from a specified work area to the proper location in the DTFSR-specified output area. As in the first form, the file name must be entered as the first parameter. The name of the work area is entered as the second parameter, and YES must be specified in the DTFSR entry WORKA. The work-area name must be the same as that specified in the DS instruction that reserves the area of main storage . Individual records for a logical file may be built in the same work area or in different work areas. Each PUT instruction specifies the work area where the completed record was built. However, only one work area can be specified in any one PUT instruction. Whenever this form of the PUT instruction is used for a logical file, a register is not required for indexing.

Whenever an output data record is transferred from an output area to an I/O device (by a PUT instruction), the data also remains in the output area until it is either cleared or replaced by other data. IOCS does not clear the output area. Therefore, if the user plans to build another record whose data does not use every position of the output record area, he must clear that area before he builds the record. If this is not done, the new record will contain interspersed characters from the preceding record. For example, in the case of output to a printer, the forms design may require printing in selected positions on one print line and in different positions on another line. In this case, the output area for the printer file should be cleared between lines.

Unblocked Records

Records transferred to any output file except disk or magnetic tape are always considered unblocked (specified as unblocked or undefined). Records for disk or tape recording are treated as unblocked if this is specified in the DTFSR entry RECFORM.

Whenever records are unblocked (either fixed or variable length), each PUT transfers a single record from the output area (or input area if updating is specified) to the file. If a work area is specified in the PUT instruction, the record is first moved from the work area to the output area (or input area) and then to the file.

Blocked Records

When blocked records are to be written on disk or tape (as specified in DTFSR RECFORM), the individually built records must be formed into a block in the output area. Then the block of records is transferred to the output file. The blocked records may be either fixed or variable length.

Fixed-length blocked records can be built directly in the output area or in a work area. Each PUT instruction for these records either adds an indexing factor to the register, or moves the completed record from the specified work area to the proper location in the output area. When an output block of records is complete, PUT causes the block to be transferred to the output file and initializes the register if one is used.

Variable-length blocked records can also be built in either the output area or a work area. The length of each variable-length record must be determined by the problem program and included in the output record as it is built. The problem program can calculate the length of the output record from the length of the

corresponding input records. That is, variable-length output records are generally developed from previously written variable-length input records, perhaps modified by current- records. Each variable-length input record must include the field that contains the length of the record (see Figures 12 and 13).

When variable-length blocked records are built in a work area, the PUT instruction performs approximately the same functions as it does for fixed-length blocked records. The PUT routines check the length of each output record to determine if the record will fit in the remaining portion of the output area. If the record will fit, PUT immediately moves the record. If it will not fit, PUT causes the completed block to be written and then moves the record. Thus, this record becomes the first record in a new block.

If variable-length blocked records are to be built directly in the output area however, an additional DTFSR entry, a TRUNC macro, and additional user programming are required. The user's program must determine if each record to be built will fit in the remaining portion of the output area. This must be known before processing of the record is started so that, if the record will not fit, the completed block can be written and the record can be built at the beginning of a new block. Thus, the length of the record must be pre-calculated and compared with the amount of remaining space.

The amount of space available in the output area at any time can be supplied to the program (in a register) by the IOCS routines. For this, the user must specify a general-purpose register in the DTFSR entry VARBLD. This register is in addition to the register specified in DTFSR IOREG. Each time a PUT instruction is executed, IOCS loads into this register the number of bytes remaining in the output area. The problem program uses this to determine if the next variable-length record will fit. If it will not fit, a TRUNC macro instruction must be issued to transfer the block of records to the output file and make the entire output area available for building the next block.

Undefined Records

When undefined records are handled, PUT treats them as unblocked. The programmer must provide any blocking he wants. He must also determine the length of each record (in bytes) and load it in a register for IOCS use, before he issues the PUT instruction for that record. The register that will be used for this purpose must be specified in the DTFSR entry RECSIZE.

Updating

A consecutive file on 2311 disk, a card input file in a 1442 or 2520, or a card file in the punch feed of a 2540 equipped with the punch-feed-read special feature, can be updated. That is, each disk or card record can be read, processed, and transferred back to the same disk location, or card, from which it was read. This function must be specified in the file definition. For a disk file, the DTSFR entry UPDATE must be specified. In the case of a card file, the file must be specified as a combined file (CMBND) in the DTFSR entry TYPEFLE.

One I/O area can be specified (DTFSR entry IOAREA1) for both the input and output of a disk or card record. If the IBM 1442 is used, however, separate areas can be specified for input and output (DTFSR entries INAREA and OUAREA). Each disk or card record is transferred to the specified input area (IOAREA1 or INAREA) by a GET instruction. After the record is processed, the next PUT instruction causes the updated record to be written in the same disk location, or punched in the same card, from which the record was read. PUT transfers the record from the main storage area specified by IOAREA1 (same area for both input and output) or by OUAREA (separate input and output areas). If a work area is specified in the PUT instruction, PUT first moves the updated record from the work area to the area specified by IOAREA1 or OUAREA, and then writes the disk record or punches the card.

A GET instruction must always precede a PUT instruction for a disk or card record, and only one PUT can be issued for each record. For a file in a 2540 or 2520 with the punch-feed-read special feature, a PUT instruction must be issued for each card, and between jobs the 2540 punch must be run out. A PUT instruction may be omitted, however, if a particular disk record or a card record in a 1442 does not require updating.

The user, while using logical IOCS (TYPEFLE=CMBND), is provided with the standard read error recovery procedure when reading and punching into the same card. No punch error recovery procedure, however, is provided.

Punch and Printer Control

Card selection in a card read-punch and line spacing or skipping in a printer can be controlled either by specified characters in the data records or by the CNTRL macro instruction. Either method, but not both, may be used for a particular logical file.

When control characters in data records are to be used, the DTFSR entry CTLCHR must be specified, and every record must contain a control character in the main-storage output area. This must be the first character of each fixed-length or undefined record, or the first character following the record-length field in a variable-length record. The DTFSR BLKSIZE specification for the output area must include the byte for the control character and, if undefined records are specified, the DTFSR RECSIZE specification must also include this byte.

The particular character included in the record is determined by the function to be performed. For example, if double spacing is to occur after a particular line is printed, the code for double spacing must be the control character in the output line to be printed. The contrcl-character codes are the same as the command codes (including the modifier bits) used for a punch or delayed-print command.

When a PUT instruction is executed, the control character in the data record becomes the command code (byte) of the Channel Command Word (CCW) that IOCS establishes. The first character after the control character in the output data becomes the first character punched or printed.

If the CNTRL macro instruction is used for non-data orders to the punch or printer (see CNTRL Macro), the DTFSR entry CONTROL is specified and DTFSR CTLCHR must be omitted. In this case, any control characters included in data records are ignored when the PUT instruction is executed. They are treated as data.

RELSE Macro

Name	Op	Operand
	RELSE	filename

The RELSE macro instruction is used in conjunction with blocked input records read from disk or tape. It allows the programmer to skip the remaining records in a block and continue processing with the first record of the next block when the next GET instruction is issued. This function can apply to a job in which records on disk or tape are categorized. Each category (perhaps a major grouping) is planned to start as the first record in a block. For selective reports, specified categories can be located readily by checking only the first record in each block.

The symbolic name of the file, specified in the DTFSR header entry, is the only parameter required for this instruction.

The release instruction discontinues the deblocking of the present block of records, which may be either fixed or variable length. RELSE causes the next GET instruction to transfer a new block to the input area, or switch I/O areas, and make the first record of the next block available for processing. GET initializes the register or moves the first record to a work area.

TRUNC Macro

Name	Op	Operand
	TRUNC	filename

The TRUNC (truncate) macro instruction is used in conjunction with blocked output records that will be written on disk or tape. It allows the programmer to write a short block of records. (Blocks do not include padding.) Thus the TRUNC macro can be used for a function similar to the RELSE instruction for input records, but in reverse. That is, when the end of a category of records is reached, that block can be written and the new category can be started at the beginning of a new block.

The symbolic name of the file, specified in the DTFSR header entry, is the only parameter required in this instruction. If this macro is issued for fixed-length blocked disk records, the DTFSR entry TRUNCS must be included in the file definition.

When TRUNC is issued, the short block is written (on disk or tape) and the output area is made available to build the next block. The last record included in the short block is the record that was built before the last PUT instruction preceding TRUNC was executed. Therefore if records are built in a work area and the problem program determines that a record belongs in a new block, the TRUNC instruction should be issued first, followed by the PUT instruction for this particular record. If records are built in the output area, however, the programmer must determine if a record belongs in the block before he builds the record.

Whenever variable-length blocked records are built directly in the output area, this TRUNC instruction must be used to write a completed block of records. When the PUT instruction is issued after each variable-length record is built, the output routines supply the programmer with the

space (number of bytes) remaining in the output area. From this the programmer determines if his next variable-length record will fit in the block. If it will not fit, he issues the TRUNC instruction to write out the block and make the entire output area available to build the record. The amount of remaining space is supplied in the register specified in the DTFSR entry VARBLD (see PUT Macro and DTFSR VARBLD).

READ Macro

Name	Op	Operand
	READ	filename,OR, name (r)

The READ macro is used to transfer records from the IBM 1287 Optical Reader operating in document mode. All operands are required. The first parameter specifies the symbolic name of the file as given in the DTFSR header entry. The second parameter, OR, indicates an optical character reader. The third parameter specifies the address of the user-provided channel command word list. The first channel command word in the list cannot be a transfer-in-channel CCW. To designate the address of the channel command word list, a register entry is used in this parameter.

Note: Document ejection and/or stacker selection and document increment functions can also be accomplished by including the appropriate CCW(s) within the channel command word list addressed by the READ macro rather than by using the CNTRL macro. This technique results in increased document throughput.

READ generates an EXCP and a CCB macro, which cause a branch to the user-provided channel command chain. When documents are being processed, only one input area may be used. The contents of the input area may later be moved to a user-defined work area. Overlap can thus be obtained by processing the contents of the work area as the subsequent document is being read into the input area. Dividing the fields in a document into blocks, and processing one block while reading the rest, is another means of achieving overlap.

At least one reference mark is required for all documents. New coordinates for the reference mark must be specified whenever a rotation in printing occurs or when a document is incremented. For this reason, a load format CCW that specifies the coordinates of the reference mark associated with a particular group of fields must be the first CCW in all user channel command word chains.

WAITF Macro

Name	Op	Operand
	WAITF	filename

The WAITF macro instruction is used only with the 1287 Optical Reader in document mode for consecutive processing. It ensures that the transfer of a record has been completed. One parameter, the symbolic name of the file, is required.

With this instruction, the program waits until data transfer is complete. Therefore, the instruction must come after a READ and before the next READ for the same file. It must be issued before the problem program attempts to process an input record for that file.

RDLNE Macro

Name	Operation	Operand
[name]	RDLNE	filename

The RDLNE macro selectively performs on-line correction when journal tapes are processed on the IBM 1285 or 1287 Optical Reader. This macro causes the reader to read a line in the on-line correction mode while processing in the off-line correction mode. If the reader cannot read a character, IOCS retries the line that contains the unread character. If this retry is unsuccessful, the user is informed of the condition in his error correction routine (specified in DTFSR COREXIT). He may then issue the RDLNE macro causing another attempt to read the line. If the character in the line cannot be read during this attempt, the character is displayed on the 1285 or 1287 display scope and the operator, if possible, keys in the correct character. If the operator cannot readily correct the error by keying in the correct character, he may enter the reject character in the line in error. This condition is posted in Filename+17 and is available for the user to examine. Wrong-length records or incomplete reads are also posted in Filename+17. See the description of COREXIT for hexadecimal indications. RDLNE should be used only in COREXIT. Otherwise, the line following the one in error is read in the on-line correction mode.

The macro requires only one parameter (the symbolic name of the 1285 or 1287 file

from which the record is to be retrieved). This name is the same as that specified in the DTFSR header entry for this file.

Note: When the RDLNE macro is used, the user must include the parameter OFFLINE=YES in his DTFSR entries.

DSPLY Macro

Name	Operation	Operand
[name]	DSPLY	Filename,r,r

The DSPLY macro is used to display a document field on the display scope of the 1287. The field is displayed for the purpose of keying in a complete field on the keyboard when a 1287 read error makes this type of correction necessary. If a 1287 read error occurs and the reject character is entered in the field in error (either by the operator if processing in the on-line correction mode or by the device if processing in the off-line correction mode), the user may use the DSPLY macro to display the field in error. When the 1287 display tube displays the full field, the operator, if possible, keys in the correct field from the keyboard. The field read from the keyboard is always read into the address specified in the CCW (normally within IOAREA1) that was originally intended for the field. The macro first blanks this field. At completion of the operation, the data is left-justified in this field.

This instruction always requires three parameters. The first parameter specifies the symbolic name of the 1287 file from which the record is to be retrieved. This name is the same as that specified in the DTFSR header entry for this file. The second parameter specifies a general purpose register (2-11) into which the problem program has placed the address of the Load Format CCW that provides the document coordinates for the field to be displayed. The address of this Load Format CCW is obtained by subtracting 8 from the address found in a half-word core location at Filename+10 when the macro is used in the COREXIT routine. Otherwise the user must determine the Load Format CCW address.

The third parameter specifies a general purpose register (2-11) into which the problem program has placed the address of the Load Format CCW that provides the coordinates of the reference mark associated with the field to be displayed. When using the DSPLY Macro, the user must ensure that the Load Format CCW that provides the document coordinates for the field to be displayed (second parameter), is command chained to the CCW used to read that field.

Note: The contents of Filename+17 are meaningful only for X'40' (1287 scanner cannot locate the reference mark) and X'04' (wrong length record), after issuing the DSPLY macro. Therefore the user must determine whether the operator was able to recognize the unreadable line of data.

RESCN Macro

Name	Operation	Operand
[name]	RESCN	filename,r,r,n,F

The RESCN macro selectively rereads a field on a document when a defective character makes this type of operation necessary. The field read is always read right-justified into the address specified in the CCW (normally within IOAREA1) that was originally intended in the field.

The parameter filename specifies the symbolic name of the 1287 file from which the record is to be retrieved. This name is the same as that specified in the DTFSR header entry for this file. The second parameter specifies a general purpose register (2-11) into which the problem program has placed the address of the Load Format CCW that provides the document coordinates for the field to be read. The address of this Load Format CCW is obtained by subtracting 8 from the address found in a half-word core location at Filename+10 when the macro is used in the COREXIT routine. Otherwise the user must determine the Load Format CCW address. The third parameter specifies a general purpose register (2-11) into which the problem program has placed the address of the Load Format CCW that provides the coordinates of the reference mark associated with the field to be read. The previous three parameters are always required, and result in one reread of the unreadable field.

The fourth parameter is required if the user wishes to attempt more than one reread of the unreadable field. This parameter (n) is the number of additional retries (nine maximum) to be attempted. The fifth parameter (F) indicates one more reread. It forces on-line correction of any unreadable characters by individually projecting the unreadable character(s) on the 1287 display scope. The operator must then key in a correction (or reject) character(s). The user must determine whether the read operation generated by RESCN has resulted in a more satisfactory read. If the reread of the field results in a wrong length record, incomplete read,

or an unreadable character error condition, it is posted in Filename+17. (See description of COREXIT for hexadecimal values.)

When using the RESCN macro, the user must ensure that the Load Format CCW, which provides the document coordinates for the field to be read (second parameter), is command chained to the CCW used to read that field.

CNTRL Macro

Name	Op	Operand
	CNTRL	filename,code,n,m

The CNTRL (control) macro instruction provides orders for these input/output units: magnetic tape units, card read-punches, printers, optical readers, and disk drives. Orders apply to physical non-data operations of a unit (with the exception of the 1285 and 1287 Optical Readers) and are peculiar to the unit involved. They specify such functions as rewinding tape, card stacker selection, line spacing on a printer, etc. For optical readers they specify marking an error line or keyboard correcting a line for journal tapes, and stacker selecting, ejecting, and incrementing documents. When a CNTRL macro instruction is executed, except for certain mnemonics used for optical readers, operation does not wait for completion of the order before returning control to the user.

CNTRL is used in conjunction with a logical file in a unit, and it requires either two, three, or four parameters. The first parameter must be the name of the file specified in the DTFSR header entry. The second parameter is the mnemonic code for the order to be performed. This must be one of a set of predetermined codes (Figure 14). The third parameter (n) is required whenever the UCS (Universal Character Set) feature is used, or a number is needed for stacker selection, immediate printer carriage control or signal count for STR devices. The fourth parameter (m) applies only to printer control and is required for delayed spacing or skipping. A number specified as either the third or fourth parameter must be a self-defining value.

Whenever CNTRL is issued in the problem program, the DTFSR entry CONTROL must be included in the file definition.

The CNTRL macro instruction must not be used for printer or punch files if the data records contain control characters and the DTFSR entry CTLCHR is included in the file definition.

Magnetic Tape Units

The CNTRL macro instruction is used to control magnetic-tape functions that are not concerned with reading or writing data on the tape. These functions are grouped in the following categories:

Rewinding tape to the load point
 REW - Rewind
 RUN - Rewind and unload

Moving tape to a specified position
 BSR - Backspace to interrecord gap
 BSF - Backspace to tapemark
 FSR - Forward space to interrecord gap
 FSF - Forward space to tapemark

Writing a tapemark
 WTM - Write tapemark

Erasing a portion of the tape
 ERG - Erase gap (writes blank tape)

The tape rewind (REW and RUN) and tape movement (BSR,BSF,FSR,and FSF) functions can be used before a tape file is opened. This allows the tape to be positioned at a desired location for opening a file under conditions such as:

- The file is located in the middle of a multifile reel.
- The DTFSR entry REWIND specifies NORWD, but for some conditions rewinding is required for the file.

UNIT	MNEMONIC CODE	n*	m**	ORDER
2400 Series Magnetic Tape Units	REW	-	-	Rewind Tape
	RUN	-	-	Rewind and Unload Tape
	ERG	-	-	Erase Gap (Writes Blank Tape)
	WTM	-	-	Write Tape Mark
	BSR	-	-	Backspace to Interrecord Gap
	BSF	-	-	Backspace to Tape Mark
	FSR	-	-	Forward Space to Interrecord Gap
	FSF	-	-	Forward Space to Tape Mark
1403, 1404, 1443, 1445 Printers	SP	a	d	Carriage Space n Lines
	SK	b	e	Skip to Channel n
1403 Printer with UCS Feature	UCS	YES	-	Ignore Data Checks
	UCS	NO	-	Accept Data Checks
2540 Card Read-Punch	PS	c	-	Select Stacker n
1442, 2520 Card Read-Punch	SS	c	-	Select Stacker n
2311 Disk Storage Drive	SEEK	-	-	Seek
1285 Optical Reader	READKB	-	-	Read 1285 Keyboard
	MARK	-	-	Mark Error Line
1287 Optical Reader	READKB	-	-	Read 1287 Keyboard
	MARK	-	-	Mark Error Line in Journal Tape Mode
	EJD	-	-	Eject Document
	SSD	c	-	Select Stacker n
	ESD	c	-	Eject and Select Document
	INC	-	-	Increment Document at Read Station
STR Devices †	EOF	f	-	End of Transmission
	INQ	f	-	Inquiry
	PREP	f	-	Prepare
	TEL	f	-	Alternate Mode
BCS Support †	PRP	-	-	Prepare
	EOT	-	-	End of Transmission
	WABT	-	-	Wait before Transmitting
	DSC	-	-	Disconnect
	ENQ	-	-	Inquiry

* a = Number of lines to be spaced immediately
b = Number of the carriage tape channel to skip to immediately
c = Number of the stacker to which a card or document is to be selected
** d = Number of lines to be spaced after printing
e = Number of carriage tape channel to skip to after printing
* f = Count to be used if other than:

EOF - 2
INQ - 10
PREP - Not applicable
TEL - 2

† See Input/Output Control Macros

Figure 14. CNTRL Macro Instruction Codes

The tape movement functions (BSR, BSF, FSR, and FSF) apply to input files only, and the following factors should be considered:

1. The FSR (or BSR) function permits the user to skip over a physical tape record (from one interrecord gap to the next). The record is passed without being read into main storage . The FSF (or BSF) function permits the user to skip to the end of the logical file (identified by a tapemark).

2. The functions of FSR, FSF, BSR, and BSF always start at an interrecord gap.

3. If blocked input records are being processed and if the user does not want to process the remaining logical records in the block, as well as one or more succeeding blocks (physical records), he must issue a RELSE macro before the CNTRL macro. Then the next GET will make the first record of the new block available for processing. If the CNTRL macro, with FSR for example, were issued without a preceding RELSE, the tape would be advanced but the next GET would make the next record in the old block available for processing.

4. For any I/O area combination except one I/O area and no work area, IOCS is always reading one physical tape record ahead of the one that is being processed. Thus, the next physical record (block) after the one being processed will be in main storage ready for processing. Therefore if a CNTRL FSR function is performed, the second physical tape record beyond the present one will be passed without being read into main storage.

5. If an FSR function (or BSR in a read backwards file) passes a tape mark, IOCS branches to the end-of-volume routine.

6. If any of these four functions is used during the processing of a file, the block count accumulated for checking standard labels, or accumulated for the checkpoint macro, could be wrong. The operator can bypass an erroneous block count when checking standard labels. However, since it is impossible to reposition the tape correctly if the block count is wrong in a checkpoint record, these commands should not be issued when using the checkpoint macro.

Printers

The CNTRL macro instruction is used for any printer forms control other than the standard single spacing.

The CNTRL macro codes for printer operations cause spacing (SP) over a specified number of lines, skipping (SK) to a specified location on the form (represented by a carriage-tape channel), or ignoring/accepting data checks for unprintable characters when the Universal Character Set (UCS) special feature is installed in a 1403. The third parameter is required for immediate spacing or skipping (before printing), or to ignore (YES) or accept (NO) data checks. The fourth parameter is required for delayed spacing or skipping (after printing).

The SP and SK operations can be used in any sequence. However, two or more consecutive immediate skips (SK) to the same carriage channel on the same printer have the same effect as the first skip only. That is, any skip order after the first is ignored. Two or more consecutive delayed spaces (SP) and/or skips (SK) to the same printer result in the last space or skip only. Any other combination of consecutive controls (SP and SK), such as immediate space followed by a delayed skip or immediate space followed by another immediate space, causes both specified operations to occur.

The CNTRL UCS macro instruction generates a command code to ignore, or accept, data checks for unprintable characters. If the parameter YES is specified, data checks resulting from unprintable characters are ignored and processing continues. If the parameter NO is specified, an unprintable character data check causes processing to stop and a message to be issued to the operator. The generated command code remains fixed until another CNTRL UCS instruction is issued, or until the command code is changed by an ASSGN control card or the UCS load buffer program. If CNTRL UCS is issued for a printer without the Universal Character Set special feature, a command reject occurs and the system enters the wait state.

IBM 2540 Card Read-Punch

Cards fed into the IBM 2540 read feed are normally selected by IOCS to stack in the R1 pocket, and those fed in the punch feed are selected to stack in the P1 pocket. The CNTRL macro code PS is used to select a card into a different stacker, specified bythe third parameter(n) in this instruction. Whenever CNTRL is used for any cards in a file, all cards must be selected by this macro. For an input file,

each GET instruction must be followed by a CNTRL instruction to properly select the card just read. For an output file, each PUT must be preceded by a CNTRL instruction to properly select the card that will contain the record being built. The possible selections are:

Feed	Pocket	Selection Number
Read	R1	1
Read	R2	2
Read	RP3	3
Punch	P1	1
Punch	P2	2
Punch	RP3	3

For input files, the CNTRL macro can be used only when one I/O area, or one I/O area and one work area, is specified for the file. For output files, the CNTRL macro may be used in conjunction with any of the permissible I/O area and work area combinations (see Processing Records Consecutively: Storage Areas).

IBM 1442 or 2520 Card Read-Punch

Cards fed in the IBM 1442 or 2520 are normally stacked in pocket 1. However, they may be selected to stack in pocket 2 by using the CNTRL macro code SS.

In a card-read operation (input or combined file), a card can be selected to pocket 2 by issuing the CNTRL instruction after the GET instruction for that card, and before the GET instruction for the following card is issued. When the following card is read, the first card is stacked in pocket 2. Whenever CNTRL is used for any card in an input file with a work area, all cards must be selected by this macro.

Whenever CONTROL = YES is specified in DTFSR (Output File), the first PUT for that file must be preceded by a CNTRL macro.

For input files, the CNTRL macro can be used only when one I/O area, or one I/O area and one work area, is specified for the file. For output files, the CNTRL macro may be used in conjunction with any of the permissible I/O area and work area combinations (see Processing Records Consecutively: Storage Areas). For combined files (DTFSR TYPEFLE=CMBND), the CNTRL macro can be used only when one I/O area (no work area) is specified for the file.

IBM 2311 Disk Storage Drive

The CNTRL macro for seeking on the 2311 applies only to files processed consecutively or by the direct access method (DAM). It does not apply to files processed by the indexed sequential system (ISFMS). This macro permits access movement to begin for the next READ, WRITE, GET, or PUT instruction for a file. While the arm is moving, the programmer may process data and/or request I/O operations on other devices.

To use CNTRL for seeking in the direct access method, the user must first specify a track address to which access movement should begin. This address must be stored in the track-reference field specified by the DTFDA entry SEEKARD. (The user must supply this address before issuing the CNTRL macro instruction.) For consecutive files, IOCS seeks the track that contains the next block (or physical record) for the file. The user does not supply a track address.

If the CNTRL macro is not used, IOCS performs the seek operation when a READ, WRITE, GET, or PUT instruction is executed.

IBM 1285 or 1287 Optical Reader

The CNTRL macro instruction with the READKB mnemonic allows the user to read a complete line from the 1285 or 1287 keyboard when processing journal tapes. This permits the user to key in a complete line on the keyboard when a read error makes this type of correction necessary. When IOCS exits to the user's COREXIT routine, the user may issue the CNTRL macro instruction to read the keyboard. The display tube will display the full line and the operator will, if possible, key in the correct line from the keyboard. The line read from the keyboard will be available in the area that previously contained the erroneous record, i.e., the area that the GET macro instruction was serving when the error occurred. The READKB mnemonic causes the problem program to wait for completion of the operation before control is regained.

If an error (data transfer error) occurs while processing in the journal tape mode, the CNTRL macro instruction used with the MARK code can provide a program-controlled means of marking lines resulting in actual or suspected errors. To ensure that the proper line is marked, the user must issue the CNTRL macro in his COREXIT routine. If the CNTRL macro is issued at any other time, the line following the line in error will be marked.

When processing in the document mode, the CNTRL macro may be used with the EJD mnemonic to eject each document. The EJD mnemonic ejects the current document and feeds the next document. The CNTRL macro with the SSD mnemonic may be used to perform the stacker selection function.

It is possible to combine the ejection and selection functions by using the ESD mnemonic. In such cases, the combined mnemonic must not be immediately preceded by an eject command or immediately followed by a stacker select command.

A selection number of 1, 2, or 3 will direct the document to stacker A, B, or R (reject), respectively. It is also possible to select stacker A and B in an alternate stacking mode. In this mode, stacker switching is automatically initiated when one stacker becomes full. The selection number for alternate stacking is 4. Stacker A fills first when selection number 4 is used in the first stacker selection macro. If selection number 4 is used subsequently to other selection numbers, the selection number immediately preceding number 4 determines the stacker. Ejection and selection of documents must occur alternately.

For documents with a scannable length greater than 6 inches, the INC mnemonic effects document incrementation, which is forward document movement of 3 inches. It may be used only once for each document.

Notes:

Document ejection and/or stacker selection and document increment functions can also be accomplished by including the appropriate CCW(s) within the channel command word list addressed by the READ macro rather than by using the CNTRL macro. This technique results in increased document throughput.

The stacker select command must follow the eject command within 270 milliseconds if the document was incremented or within 295 milliseconds if the document was not incremented. If timing requirements are not met, a late stacker selection condition occurs (see COREXIT routine).

STR Devices

The CNTRL macro for STR devices is described under Processing with STR Devices: CNTRL Macro.

BSC Support

The CNTRL macro for BSC support is described under Binary Synchronous Communication: CNTRL Macro.

CHNG Macro

Name	Op	Operand
	CHNG	SYSnnn

A system may have one or more 2-channel, simultaneous read-while-write tape control units, which must be connected to selector channels 1 and 2 (special feature). When it does, the programmer should assign all tape input files for a job to one channel, and all tape output files to the other channel. This utilizes the maximum reading-while-writing capability of such a system. During the job, however, the program may require that a particular tape unit be switched from one channel to the other. For example, an output file of records may be completed during one phase of the job, and then be needed as input to a subsequent phase. Conversely, after the records from an input file have been processed, that same file may become an output file to write changed, or different records.

The CHNG (change) macro instruction can be used to switch a tape unit from one selector channel to the other (from input to output status, or vice versa). The symbolic unit, specified in the DTFSR entry DEVADDR and named in the Job Control ASSGN card, is the only parameter required in this instruction. The CHNG macro merely switches channel assignments; it does not change the symbolic reference to the unit. This macro may be used in conjunction with either logical or physical IOCS instructions.

This macro instruction must not be issued while the file(s) associated with the symbolic unit is open.

PRTOV Macro

Name	Op	Operand
	PRTOV	filename,n,routine-name

The PRTOV (printer overflow) macro instruction is used in conjunction with a logical file in a printer to specify the operation to be performed on a carriage overflow condition. Whenever this macro instruction is to be issued in a problem program, the DTFSR entry PRINTOV must be included in the file definition.

PRTOV requires two or three parameters. The first parameter must be the name of the logical file specified in the DTFSR header entry. The second parameter must specify the number of the carriage tape channel (9 or 12) used to indicate the overflow. This is entered as immediate data 9 or 12. The channel 9 or 12 overflow signal is turned on by any PUT macro instruction that causes printing on the overflow line, or by a CNTRL SP macro instruction that causes spacing over the overflow line of the form.

When the first two parameters are specified and an overflow condition occurs, IOCS restores the printer carriage to the first printing line on the form (Channel 1), and printing of detail lines continues.

A third parameter is entered in this instruction if the programmer prefers to branch to his own routine on an overflow condition, rather than skipping directly to channel 1 and continuing with the detail printing. It specifies the symbolic name of the user's routine. In this case, IOCS does not restore the carriage to channel 1.

In his routine, the user may issue any IOCS macro instructions (except PRTOV) to perform whatever functions he desires. For example, this allows him to print total lines, skip to channel 1, and print overflow page headings. At the end of his routine, the user must return to IOCS by branching to the address in register 14. IOCS supplies this address upon entry to the user's routine. Therefore, if IOCS macros are used in the routine, the address must be saved.

If the user requires register 15 in his routine, he must also save this register and restore it prior to returning to his main program.

The PRTOV macro instruction may be issued anywhere in the problem program. The macro causes a skip to channel 1, or a branch to the user's routine, only if an overflow condition (a punch in carriage channel 9 or 12) was previously detected as a line was printed. (An overflow punch is not recognized during a carriage skip operation.) An overflow condition that is detected as any line is printed (PUT) is recognized by the PRTOV macro after the following line is printed. For example in the following program steps, if a channel 12 punch was read as line X was printed, PRTOV causes a skip to channel 1 after line X+1 is printed.

Thus, one extra line is always printed after the overflow punch (9 or 12) is detected and before the overflow functions can occur. Therefore, in planning a printer operation, the overflow punch must coincide with next-to-the-last line to be printed on the form.

```
PUT     FILEA       -for Line X
 .
 .                  Process
 .
PRTOV   FILEA,12
 .
 .                  Process
 .
```

```
PUT     FILEA       -for Line X+1
 .
 .                  Process
 .
PRTOV   FILEA,12    Skips to channel 1 if
                    overflow punch was
                    detected as Line X was
                    printed
 .
 .                  Process
 .
```

```
PUT     FILEA       -for Line X+2
```

IOCS causes Line X+2 to be printed on the first line of the following page whenever PRTOV specifies only the first two parameters. If a user's routine is specified, however, the functions performed in that routine affect the positioning of the form and thus determine where IOCS will print Line X+2 when the following PUT is executed.

The channel 9 or 12 overflow signal is turned off after the PRTOV macro has been executed, or when a CNTRL macro instruction is issued to cause skipping to the first printing line on the form (channel 1).

If a channel 9 punch is used in the carriage tape, the PRTOV macro must request posting of device end in the Command Control Block (CCB) or a physical IOCS error message will result.

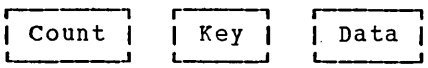
PROCESSING DISK RECORDS BY THE DIRECT ACCESS METHOD

Disk records can be processed in a random order by the Direct Access Method (DAM). In this method the user specifies the address of the record to IOCS, and issues a READ or WRITE macro instruction to transfer the specified record. Variations in the parameters of the READ or WRITE instructions permit records to be read, written, updated, replaced, or added to a file. Whenever this method of processing records is used, the logical file and main-storage area(s) allotted to the file must be defined by the declarative macro DTFDA (Define the File for Direct Access). The detail parameter entries for this definition are described under File Definition Macros.

Record Types

Records on disk that will be processed by DAM can have either of two formats: with a key area, or without.

With key area



Without key area

Whenever records in a file have keys that are to be processed:

- Every record must have a key,
- All keys must be the same length, and
- The length of the keys must be specified in the DTFDA entry KEYLEN (maximum length is 255 bytes).

Whenever the DTFDA entry KEYLEN is not specified for a file, IOCS ignores keys and the disk records may, or may not, contain key areas.

IOCS considers all records as unblocked (one logical record per one physical record). If the user wants blocked records, he must provide his own blocking and deblocking. Records are also considered to be either fixed length or undefined. (Undefined includes variable-length records.) The type of records in the file must be specified in the DTFDA entry RECFORM. Whenever records specified as undefined are to be loaded, added, or written in a file, the user must determine the length of each data record and load it in a register (specified by the DTFDA entry RECSIZE) before he issues the WRITE instruction for that record. IOCS adds the length of the key when required.

Storage Areas

Records in one logical file are transferred to or from one I/O area in main storage. This area must be large enough to contain the largest record in the file. If disk-record key areas are to be transferred by a READ/WRITE instruction, the I/O area must also provide space for the the length of the key (specified in DTFDA KEYLEN). Furthermore, if a file is to be created or if records are to be added to a file, the main-storage I/O area must include an eight-byte count area. The I/O area requirements are illustrated schematically in Figure 15 and described in detail in the DTFDA entry IOAREA1.

Reference Methods

With the direct access method of processing, each record that is to be read or written is specified by providing IOCS with two references:

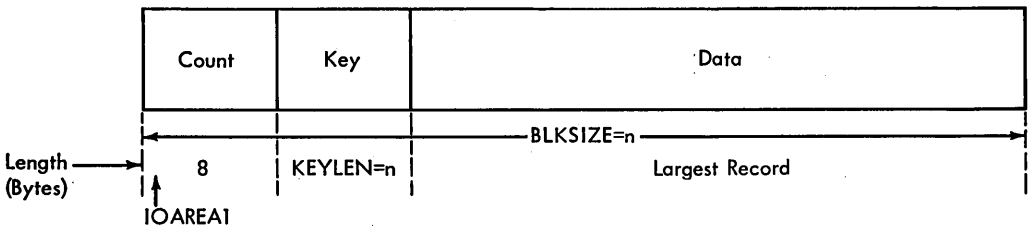
- Track reference. This gives the track on which the desired record is located.
- Record reference. This may be either the record key (if the records contain key areas) or the record identifier (ID).

IOCS seeks the specified track, searches it for the individual record, and causes the record to be read or written, as indicated by the macro instruction. If a specified record is not found, IOCS sets a no-record-found indication in the user's error/status field, which is specified by the DTFDA entry ERRBYTE. This indication can be tested by the problem program, and additional processing can be programmed to suit the user's requirements.

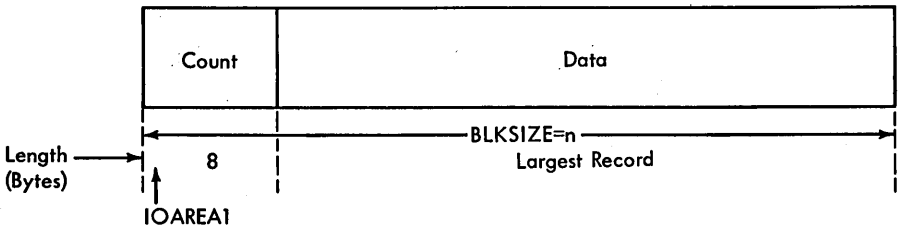
Multiple tracks can be searched for a record specified by key, if the DTFDA entry SRCHM is included in the file definition. In this case, if the record is not found after an entire cylinder is searched, the end-of-cylinder bit (instead of the no-record-found bit) is set on in the error/status field.

When the I/O operation is started, control is returned immediately to the problem program. Therefore when the program is ready to process the input record, or build the succeeding output record for the same file, a test must be made to ensure that the previous transfer of data is complete. This is done by issuing a WAITF macro instruction in the problem program.

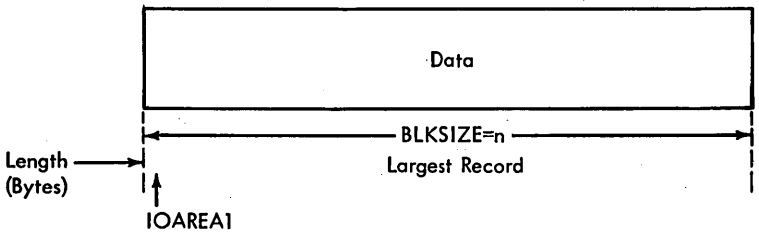
CREATE A FILE OR ADD RECORDS TO A FILE: RECORDS WITH KEY AREAS



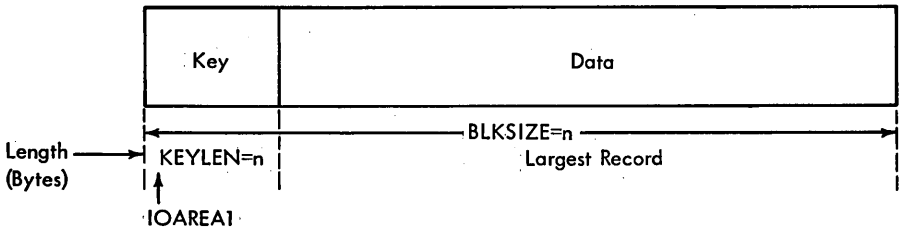
CREATE A FILE OR ADD RECORDS TO A FILE: RECORDS WITHOUT KEY AREAS



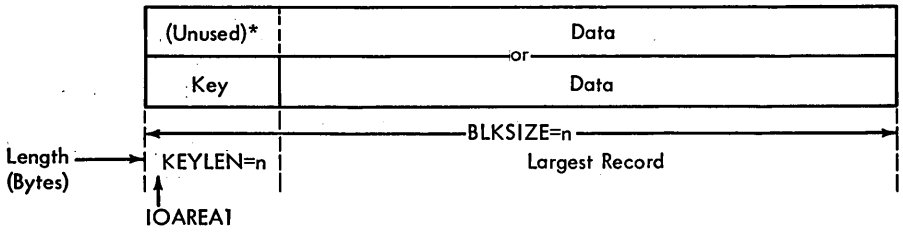
READ OR WRITE (UPDATE) BY KEY, OR BY ID WITHOUT DTFDA KEYLEN



READ OR WRITE (UPDATE) BY ID WITH DTFDA KEYLEN



READ OR WRITE (UPDATE) BY KEY AND BY ID WITH DTFDA KEYLEN*



* DTFDA specifies READKEY (or WRITEKY), READID (or WRITEID), and KEYLEN for a file. The first n bytes are unused when a READ (or WRITE) by key is executed.

Figure 15. Schematic of I/O Area in Main Storage, for DAM

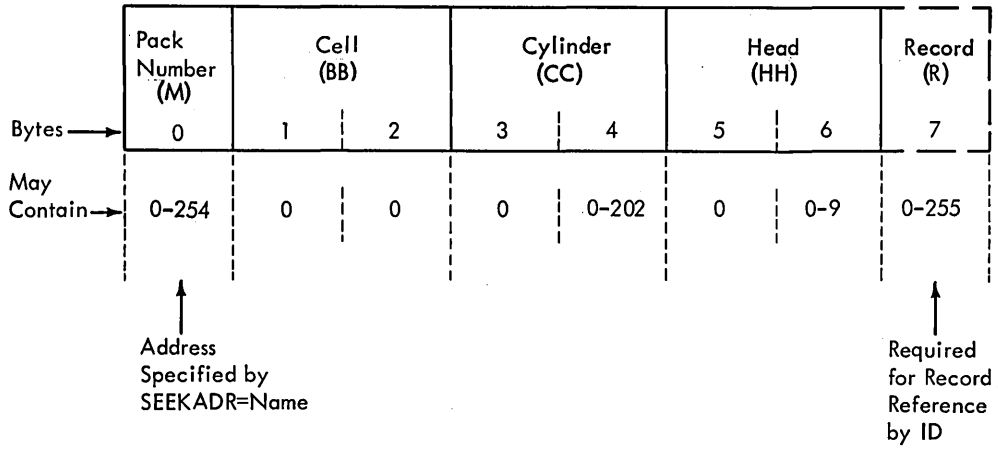


Figure 16. Track Reference Field

After a READ or WRITE instruction for a specified record has been executed, IOCS can make the ID of the next record available to the problem program. When record reference is by key and multiple tracks are searched, the ID of the specified record (rather than the next record) is supplied. The function of supplying the ID is useful for a random updating operation or for the processing of successive disk records. If the user is processing consecutively on the basis of the next ID and does not have an end-of-file record, he can check the ID supplied by IOCS against his file limits to determine when he has reached the end of his logical file. To request that IOCS supply the ID, the user must set up a 5-byte field (in which IOCS can store the ID) and specify the symbolic address of this field in the DTFDA entry IDLOC.

Track Reference: To provide IOCS with the track reference, the user sets up an 8-byte track-reference field in main storage, assigns a symbolic name, and specifies the symbolic name in the DTFDA entry SEEKADR. Before issuing any read or write instruction for a record, the user must store the proper track information (MBBCCHH) in the first seven bytes of this field. The field (Figure 16) contains the following seven bytes for track reference. The eighth byte (R), listed here and shown in the figure, is used when reference to records is by record number (see Record Reference: Identifier). All numbers must be supplied in binary notation.

Byte	Ident.	Contents	Information
0	M	0-254	Number of the pack (0-254) on which the record is located. All packs for a file must be numbered consecutively starting with 0. That is, the first pack must be number 0, the second pack number 1, etc. This number relates to a numbered symbolic unit (SYS000-SYS254). Two or more symbolic units for a file must be numbered consecutively, but the numbering may start with any SYSnnn number.
1-2	B,B	0,0	Reserved for cell number, which relates to the IBM 2321 Data Cell Drive. These two bytes are always zero for 2311 disk-storage references.
3-4	C,C	0,0-202	Number of the cylinder (0-202) in which the record is located. The first byte is always zero, and the second byte specifies one of the 203 cylinders in a disk pack. These two bytes with the next two

			(HH) provide the track identification.
5-6	H,H	0,0-9	Number of the read/write head (0-9) that applies to the record. The first byte is always zero, and the second byte specifies one of the ten disk surfaces in a disk pack.
7	R	0-255	Sequential number of the record on the track.

When the READ or WRITE is executed, IOCS refers to this field to select the specific track on the appropriate disk pack.

Record Reference: The Direct Access Method allows records to be specified by record key or by record identifier.

Key: If records contain key areas, the records on a particular track can be randomly searched by their key numbers. This allows the user to refer to records by the logical control information associated with the records, such as an employee number, a part number, a customer number, etc.

For this type of reference the programmer must specify, in the DTFDA entry KEYARG, the symbolic name of a main-storage key field. He then stores each desired key in this field.

Identifier (ID): Records on a particular track can be randomly searched by their position on the track, rather than by control information. This is accomplished by using the record identifier (ID). The record identifier, which is part of the count area of any 2311 disk record, consists of five bytes (CCHHR). The first four bytes (cylinder and head) refer to the location of the track, and the fifth byte (record) uniquely identifies the particular record on the track. When records are specified by ID, they must be numbered in succession, and without missing numbers, on each track. The first data record on a track must be record number 1, the second number 2, etc.

Whenever records are to be identified by the record ID method, the eighth byte (R) of the track-reference field (Figure 16) must contain the number of the desired record. When a READ or WRITE instruction that searches by ID is executed, IOCS refers to the track-reference field to determine which record is requested by the program. The number in this field is compared with the corresponding fields in the count areas of the disk records. The R byte specifies the particular record on the track.

Creating a File or Adding Records to a File

In addition to reading, writing, and updating records randomly, the direct access method permits the user to create a file or add records to a file. When this is done, all three areas of a disk record are written: the count area, the key area (if present), and the data area. The new record may be written after the last record written on a specific track. This may be done by using the WRITE instruction with the parameter AFTER. The remainder of the track is erased.

When records are to be added to a disk file by the AFTER method (specified by DTFDA AFTER=YES), IOCS ensures that each record will fit on the track specified for it. If the record will fit, IOCS writes the record; if it will not fit, IOCS sets a no-room-found indication in the user's error/status field (specified by the DTFDA entry ERRBYTE). In the AFTER method IOCS determines the location where the record is to be written.

For this, IOCS uses the first record on each track (R0) to maintain updated information about the data records on the track. Record 0 (Figure 17) has a count area and a data area, and contains the following:

Count Area
- Flag (not normally transferred to main storage)
- Identifier
- Key Length (KL)
- Data Length (DL)

Data Area (8 bytes)
- 5 Bytes - ID of last record written on track (CCHHR).
- 2 Bytes - Number of unused bytes remaining on track.
- 1 Byte - (Unused)

Each time WRITE AFTER is executed, IOCS updates the data area of this record.

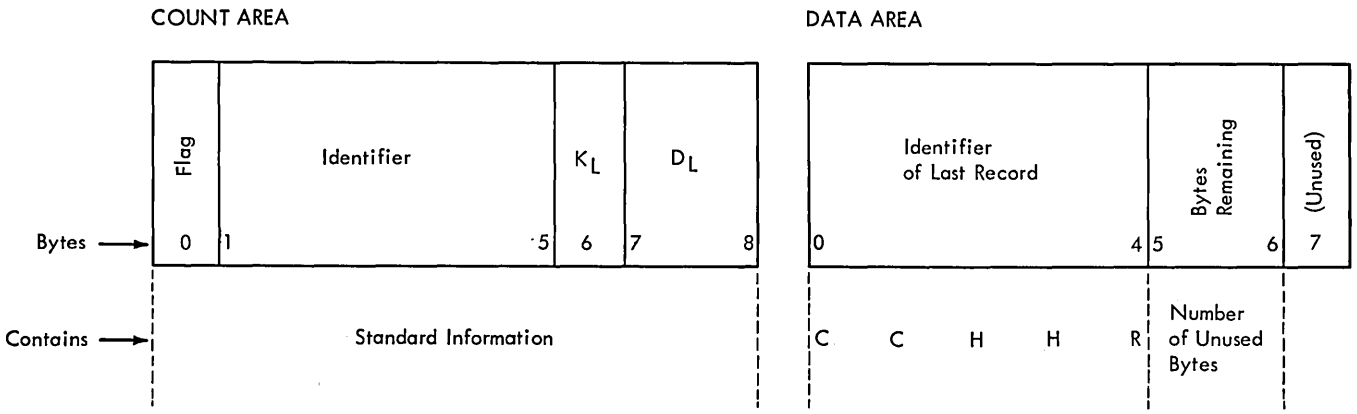


Figure 17. Contents of Record 0

READ Macro

Name	Op	Operand
	READ	filename,KEY
	READ	filename,ID

This instruction causes a record to be transferred from disk storage to an input area in main storage . The input area must be specified in the DTFDA entry IOAREA1.

The READ macro instruction is written in either of two forms, depending on the type of reference used to search for the record. Both forms may be used for records in any one DTFDA-specified logical file if the logical file has keys.

The instruction always requires two parameters. The first parameter specifies the symbolic name of the file from which the record is to be retrieved. This name is the same as that specified in the DTFDA header entry for this file. The second parameter specifies the type of reference used for searching the records in the file.

Record Reference by Key

If the record reference is by key (control information in the key area of the disk record), the second parameter in the READ instruction must be the word KEY, and the DTFDA entry READKEY must be included in the file definition.

Whenever this method of reference is used, the problem program must supply the key of the desired record to IOCS before the READ instruction is issued. For this, the key must be stored in the key field (specified in the DTFDA entry KEYARG).

When the READ instruction is executed, IOCS searches the previously specified track (stored in the 8-byte track-reference field) for the desired key.

Then when a disk record containing the specified key is found, the data area of the record is transferred to the main-storage input area.

Only the specified track is searched unless the programmer requests that multiple tracks be searched on each READ instruction. A search of multiple tracks is specified by including the DTFDA entry SRCHM in the file definition. With this entry, the specified track and all following tracks are searched until the desired record is found or the end of the cylinder is reached. The search of multiple tracks continues through the cylinder even though part of the cylinder may be assigned to a different logical file.

Record Reference by ID

If the record reference is by ID (identifier in the count area of records), the second parameter in the READ instruction must be the letters ID, and the DTFDA entry READID must be included in the file definition.

Whenever this method of reference is used, the problem program must supply both the track information and the record number in the 8-byte track-reference field. When the READ instruction is executed, IOCS searches the specified track for the particular record. When a disk record containing the specified ID is found, both the key area (if present and specified in DTFDA KEYLEN) and the data area of the

record are transferred to the main-storage input area.

WRITE Macro

Name	Op	Operand
	WRITE	filename,KEY
	WRITE	filename,ID
	WRITE	filename,RZERO
	WRITE	filename,AFTER

This instruction, except in the case of RZERO, causes a record, which has been built in an output area of main storage, to be transferred from main storage to disk storage. The output area must be specified in the DTFDA entry IOAREA1.

The WRITE macro instruction is written in one of four forms, depending on the type of reference that is used to search for the record location in the file. All forms may be used for records in any one DTFDA-specified logical file if the logical file has keys.

The instruction always requires two parameters. The first parameter specifies the name of the file to which the record is to be transferred. This name is the same as the one specified in the DTFDA header entry for this file. The second parameter specifies the type of reference that is used for searching the records on disk to find the proper location to write the output record.

Record Reference by Key

If the disk-storage location for writing records is determined by the record key (control information in the key area of the disk record), the work KEY is entered as the second parameter of the WRITE instruction. Also the DTFDA entry WRITEKY must be included in the file definition.

Whenever this method of reference is used, the problem program must supply the key of the desired record to IOCS before the WRITE instruction is issued. For this, the key must be stored in the key field (specified by the DTFDA entry KEYARG). When the WRITE instruction is executed, IOCS searches the previously specified track (stored in the 8-byte track-reference field) for the desired key. Then, when a disk record containing the specified key is found, the data in the main-storage output area is transferred to the data area of the disk record. This replaces the information previously recorded in the data area. IOCS uses the count field of the original record to control the writing of the new record. If a record is shorter than the original record, it is padded with zeros. A record longer than the original record is written only to the extent of the area indicated in the count field on the track, and any excess bytes are lost. In either case (short or long records) IOCS turns on the wrong-length-record bit in the error/status field.

Only the specified track is searched unless the programmer requests that multiple tracks be searched on each WRITE instruction. Searching multiple tracks is specified by including the DTFDA entry SRCHM in the file definition. In this case, the specified track and all following tracks are searched until the desired record is found or the end of the cylinder is reached. The search of multiple tracks continues through the cylinder even though part of the cylinder may be assigned to a different logical file.

Record Reference by ID

If the disk-storage location for writing records is determined by the record ID (identifier in the count area of records), the letters ID are entered as the second parameter of the WRITE instruction. Also the DTFDA entry WRITEID must be included in the file definition.

Whenever this method of reference is used, the problem program must supply both the track information and the record number in the 8-byte track-reference field. When the WRITE instruction is executed, IOCS searches the specified track for the particular record. When the disk record containing the specified ID is found, the information in the main-storage output area is transferred to the key area (if present and specified in DTFDA KEYLEN) and the data area of the disk record. This replaces the key and data previously recorded, IOCS uses the count field of the original record to control the writing of the new record. If a record is shorter than the original record, it is padded with zeros. A record longer than the original record is written only to the extent of the area indicated in the count field on the track, and any excess bytes are lost. In either case (short or long records) IOCS turns on the wrong-length-record bit in the error/status field.

Record Reference: Record Zero

If record zero (R0) is to be written, the second parameter of the WRITE instruction must be the specification RZERO. Also the DTFDA entry AFTER must be included in the file definition.

This reference should be used each time the problem program reuses a certain

portion of a disk pack. It may be used as a utility function to initialize a limited number of tracks or cylinders. Only one track at a time, however, may be initialized. This is done by issuing a WRITE RZERO instruction with the address of each track to be initialized.

Whenever this method of reference is used, the problem program must supply the track information (cylinder and track number) in the 8-byte track-reference field. Any record number is valid but will be ignored. When WRITE is executed, IOCS writes a new R0 with the maximum capacity of the track (3625 characters) and erases the full track after R0.

Record Reference: After

If a record is to be written following the last record previously written on a disk track (regardless of its key or ID), the second parameter of the WRITE instruction must be the specification AFTER. For this operation the DTFDA entry AFTER must be included in the file definition.

Whenever this method of reference is used for writing records, the problem program must supply the track information in the first seven bytes of the 8-byte track-reference field. When WRITE is executed, IOCS examines the capacity record (Record 0) on the specified track to determine the location and amount of space available for the record. If the remaining space is large enough, the information in the main storage output area is transferred to the disk track in the location immediately following the last record. The count area, the key area (if present and specified by DTFDA KEYLEN), and the data area are written. IOCS then updates the capacity record.

If the space remaining on the track is not large enough for the record, IOCS does not write the record and, instead, sets a no-room-found indication in the user's error/status field (specified by the DTFDA entry ERRBYTE).

Whenever this instruction will be used in a problem program, it is the user's responsibility to ensure that the capacity record reflects the present condition of the file. Therefore, if he is going to build a new file in an area of the disk pack that contains outdated records, the capacity records must first be set up to reflect empty tracks. An IBM-supplied utility program is available to construct Record 0.

If records in the file are specified as undefined (RECFORM=UNDEF), the programmer must determine the length of each record and load it in a register for IOCS use, before he issues the WRITE instruction for that record. The register that will be used for this purpose must be specified in the DTFDA entry RECSIZE.

WAITF Macro

Name	Op	Operand
	WAITF	filename

The WAITF macro instruction is used to ensure that the transfer of a record has been completed. It requires only one parameter: the symbolic name of the file containing the record.

This instruction must be issued before the problem program attempts to process an input record or build another output record for the file concerned. The program enters a waiting loop until the transfer of data is complete. Thus, the WAITF macro instruction must be issued after any READ or WRITE instruction for a file, and before the succeeding READ or WRITE instruction for the same file.

The WAITF macro makes error/status information, if any, available to the problem program in the field specified by DTFDA ERRBYTE.

CNTRL Macro

Name	Op	Operand
	CNTRL	filename,SEEK

The CNTRL (control) macro instruction is used to begin access movement for the next READ or WRITE for a file. It requires two parameters.

The first parameter specifies the symbolic name of the file, which is the same name as that specified in the DTFDA header entry for the file. The second parameter must be the word SEEK.

Before issuing the CNTRL macro instruction, the user must specify a track address to which access movement should begin. This address must be stored in the track-reference field specified by the DTFDA entry SEEKADR. While the disk-storage arm is moving, the programmer may process data and/or request I/O operations for files on other devices.

If the CNTRL macro is not used, IOCS performs the seek operation when a READ, WRITE, GET, or PUT instruction is executed.

PROCESSING DISK RECORDS BY THE INDEXED SEQUENTIAL SYSTEM

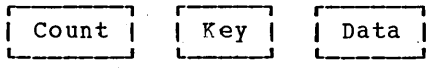
The Indexed Sequential File Management System (ISFMS) permits disk records to be processed in random order or in sequential order by control information. For random processing, the user supplies the key (control information) of the desired record to ISFMS, and issues a READ or WRITE macro instruction to transfer the specified record. For sequential processing by control information (key), the user specifies the first record to be processed and then issues GET or PUT macro instructions until all desired sequential records have been processed. The successive records are made available in sequential order by key. Variations in macro instructions permit:

- A logical file of records to be loaded onto disk (created).
- Individual records to be read from, added to, or updated in the file.

Whenever the indexed sequential system of processing is used, the logical file and main-storage areas allotted to the file must be defined by the declarative macro DTFIS (Define The File for Indexed Sequential System). The detail parameter entries for this definition are described under File Definition Macros.

Record Types

When an ISFMS file is originally organized, it is loaded onto the disk pack(s) from pre-sorted input records. These records must have been sorted by control information. All records in the disk file must contain key areas:



All keys must be the same length, and the length must be specified in the DTFIS entry KEYLEN, (maximum length is 95 bytes).

The logical records must be fixed length, and the length must be specified in the DTFIS entry RECSIZE. Logical records may be either blocked (two or more logical records in one physical record) or unblocked (one logical record per one physical record). This must be specified in the DTFIS entry RECFORM. When blocked records are specified, the key of the highest record (last) in the block is the key for the block, and therefore must be stored in the key area of the disk record.

The location of the key within each logical record must be specified in the DTFIS entry KEYLOC. The number of records in a block must be specified in the DTFIS entry NRECDS. This specification is "1" for unblocked records.

Storage Areas

Records in one logical file are transferred to, or from, one or more I/O areas in main storage . The areas must always be large enough to contain the key area and a block of records, or a single record if unblocked records are specified. In addition, it must allow space for the count area when a file is to be loaded, or when records are to be added to a file. For the functions of adding or retrieving records, the I/O area must also provide space for a sequence link field that is used in conjunction with overflow records (see Addition of Records, and Overflow Areas). The I/O area requirements are illustrated schematically in Figure 18 and described in detail in DTFIS entries IOAREAL, IOAREAR, and IOAREAS.

Records may be retrieved and processed directly in the I/O area or in a work area. If the records are to be processed in the I/O area, a register must be specified in the DTFIS entry IOREG. This is used to point to the beginning of the data portion of each record and thus locate the record for processing. Note: For sequential unblocked records, the key is at the beginning of the I/O area.

If the records are to be processed in a work area, the DTFIS entry WORKL, WORKR, or WORKS must be specified. ISFMS moves each individual input record from the I/O area to the work area where it is available to the problem program for processing. Similarly, on output, ISFMS moves the completed record from the work area to the I/O area where it is available for transfer to disk storage. Whenever a work area is used, a register is not required.

Organization of Records on Disk

When a logical file of presorted records is loaded onto disk, ISFMS organizes the file in a way that allows the user access to any record, in the most efficient manner.

Reference can be made to records at random throughout the logical file, or to a series of records in the file in their presorted sequence (collating sequence). The organization also provides for additions to the file at a later time, while still maintaining both the random and sequential reference capabilities.

ISFMS loads the records, one after the other, into a specified area of the disk pack. This is called the prime area of the

logical file on disk. The starting and ending limits of this area are specified by the user in Job Control XTENT cards. The prime data area must start on the first track (track 0) of a cylinder (other than 0), and it must end on the last track (track 9) of the same or a different cylinder. Prime data extents cannot start or end in the middle of a cylinder. Whenever the prime area extends into two or more disk packs, it must be continuous from one pack to the next and may not be interrupted. In this case, an XTENT card is required to define each area of each disk pack on which the prime area is located. For example, if the prime area extends over three disk packs, three XTENT cards are required, one for each disk pack.

Whenever any type of processing is being done for an Indexed Sequential file, all packs in a multipack file must be on-line.

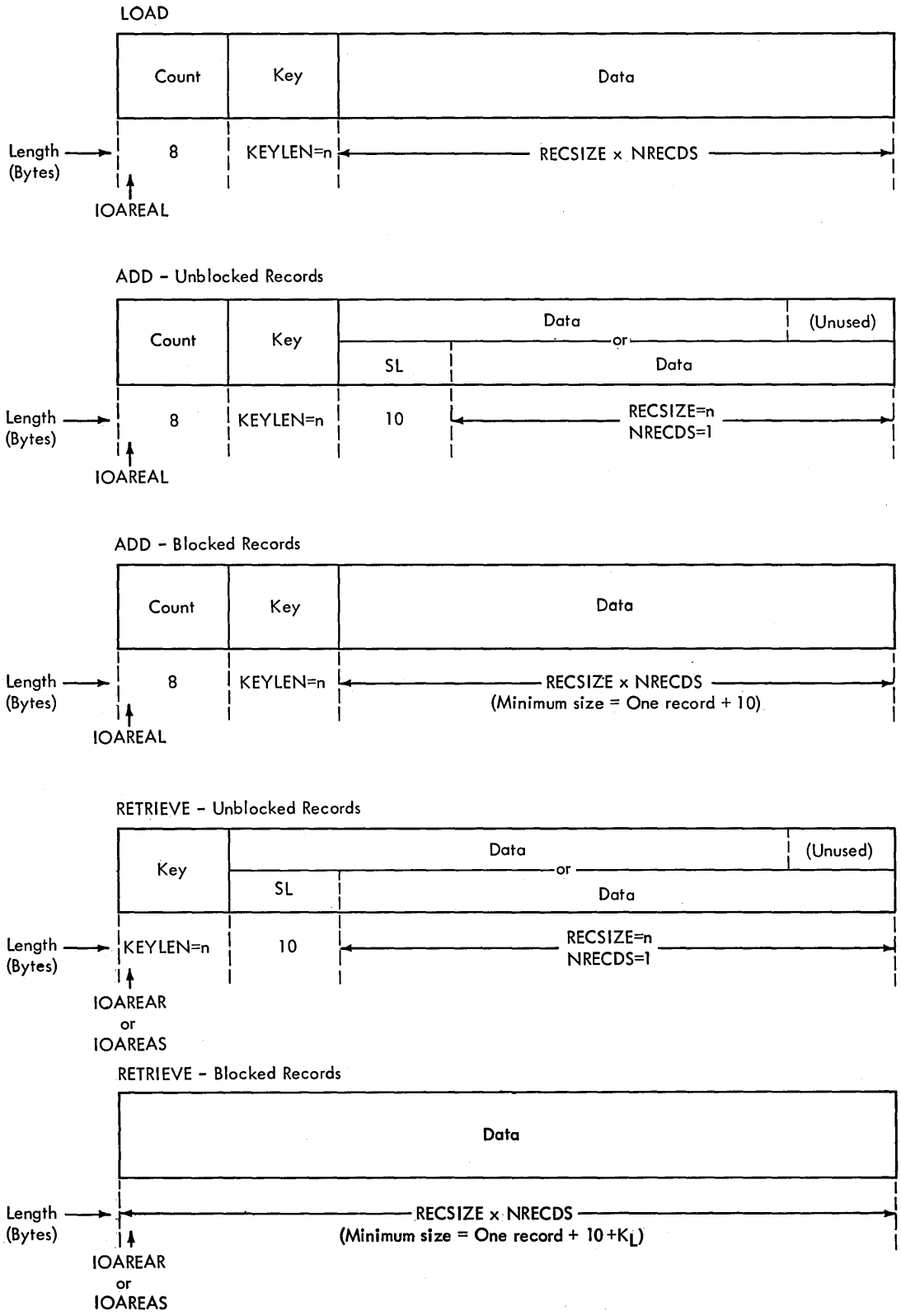
Indices

As ISFMS loads a file of records sorted by control information, it builds a set of indices for the file. The indices are utilized for both random and sequential reference to records as follows:

- They permit rapid access to individual records for random processing.
- They supply the means of providing records in key order during sequential processing.

Either two or three indices are built, depending on the user's specifications. Both a track index and a cylinder index are always constructed. A master index is also constructed if the DTFIS entry MSTIND is included in the file definition.

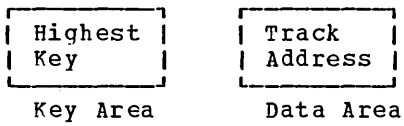
Once a file has been loaded and the related indices have been built, the ISFMS routines search for specified records by referring to the indices. When a particular record (specified by key) is requested for processing, ISFMS searches the master index (if used), then the cylinder index, then the track index, and finally the individual track. Each index narrows the search by pointing to the portion of the next-lower index whose range includes the specified key. Because of the high speed and efficiency of the direct access devices in a System/360, a master index should be established only for exceptionally large files, for which the cylinder index occupies several tracks (possibly four or more). That is, it is generally faster to search only the cylinder index (followed by the track index) when the cylinder index occupies less than four tracks.



SL = Sequence Link

Figure 18. Schematic of I/O Areas in Main Storage, for ISFMS

The indices are made up of a series of entries, each of which includes the address of a disk track and the highest key on that track, or cylinder. Each entry is a separate disk record composed of both a key area and a data area. The key area contains the highest key on the track or cylinder, and its length is the same as that specified for logical data records (in the DTFIS entry KEYLEN). The data area of each index is ten bytes long, and it contains track information including the track address.



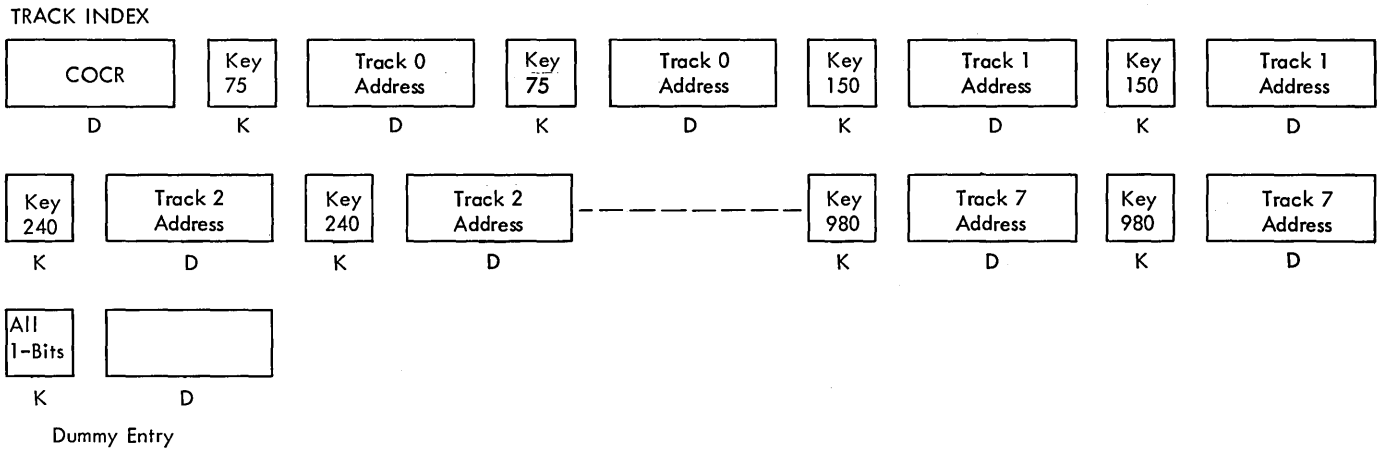
Track Index: The track index is the lowest level index for the logical file. A separate track index is built for each cylinder used by the file, and it contains index entries for that cylinder only. Each track index is located on the cylinder that it is indexing. It is always on the first track of that cylinder. Track indices are considered part of the prime data area specified by a Job Control XTENT card.

When the track indices are originally constructed, they contain two entries (normal and overflow) for each track utilized on the cylinder. For example, if the prime area of the logical file utilizes eight tracks on a cylinder, the track index might contain the entries shown in Figure 19. The use of two index records for each track is required because of overflow records that will occur if more records are inserted in the file at a later time (see Addition of Records, and Overflow Areas). When overflow records for a track exist, the second (overflow) index record contains the key of the highest record in the overflow chain for this track and the address of the lowest record in the overflow chain. The dummy entry indicates the end of the track index. Any following records are logical- file data records. The COCR entry is required whenever a cylinder overflow area is specified (see Example of an Organized File).

Cylinder Index: The cylinder index is an intermediate-level index for the logical file. It contains an index entry for each cylinder occupied by the file. This index is built in the location specified by the user in a Job Control XTENT card. The cylinder index may be built wherever the user chooses, but it may not be on one of the cylinders that contains data records for this file. It must be on a separate cylinder, or it may be on a separate disk pack that will be on-line whenever this logical file is processed.

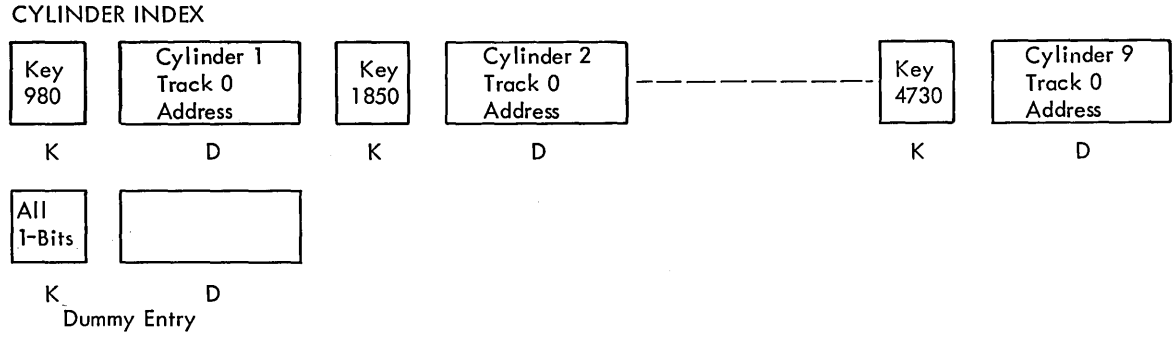
The cylinder index may be located on one or more successive cylinders. Whenever the index is continued from one cylinder to another, the last index entry on the first cylinder contains a linkage field that points to the first track of the next cylinder. A cylinder index may not be continued from one pack to another, however. It must be completely contained within one disk pack.

This index contains one entry for each cylinder occupied by the data file. The key area contains the highest key associated with the cylinder, and the data area contains the address of the track index for that cylinder. For example if a file requires nine cylinders, the cylinder index might contain the entries shown in Figure 20. The dummy entry indicates the end of the cylinder index.



K = Key Area
D = Data Area
COCR = Cylinder Overflow Control Record (R0)

Figure 19. Schematic Example of a Track Index



K = Key Area
D = Data Area

Figure 20. Schematic Example of a Cylinder Index

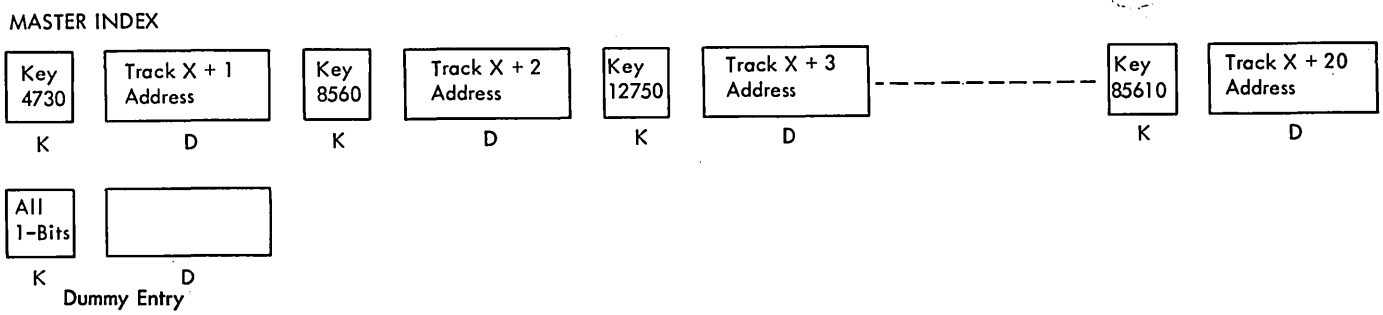


Figure 21. Schematic Example of a Master Index

Master Index: The master index is the highest level index for a logical file built by the IBM System/360 Basic Operating System. This index is optional and it is built only if it is specified by the DTFIS entry MSTIND. It is built in the location specified by a Job Control XTENT card. Like the cylinder index, it may be located on the same disk pack with the logical-file records or on a different pack that will be on-line whenever the records are processed.

The master index must immediately precede the cylinder index on a disk pack, and it may be located on one or more successive cylinders. Whenever it is continued from one cylinder to another, the last index entry on the first cylinder contains a linkage field that points to the first track of the next cylinder. A master index may not be continued from one pack to another, however. It must be completely contained within one disk pack.

The master index contains an entry for each track of the cylinder index. The key area contains the highest key on the cylinder-index track, and the data area contains the address of that track. For example, if a master index is located on track X and a cylinder index is located on tracks X + 1 through X + 20, the master index might contain the entries shown in Figure 21. The dummy entry indicates the end of the master index.

Addition of Records, and Overflow Areas

Some time after a logical file has been organized on disk it may become necessary to add records to the file. These records may contain keys that are above the highest key presently in the file, and thus constitute an extension of the file. Or, they may contain keys that fall between keys already in the file and therefore require insertion in the proper sequence in the organized file.

If all records to be added have keys that are higher than the highest key in the organized file, the upper limit of the prime area of the file can be adjusted (if necessary) by the specification in a Job Control XTENT card. Then the new records, which must be presorted, can be added by loading them into the file. No overflow area is required. The file is merely extended further on the disk pack.

If records must be inserted among those already organized, however, an overflow area will be required. The ISFMS system uses the overflow area to permit the insertion of records without necessitating a complete reorganization of the established file. The fast random and sequential retrieval of records is maintained by inserting references to the overflow area in the track indices, and by using a chaining technique in the overflow records. For chaining, a sequence-link field is prefixed to the user's data record in the overflow area. The sequence-link field contains the address of the record in the overflow area that has the next-higher key. Thus a chain of sequential records can be followed in a search for a particular record. The sequence-link field of the highest record in the chain indicates the end of the chain. All records in the overflow area are unblocked, regardless of the specification (in DTFIS RECFORM) for the data records in the logical file.

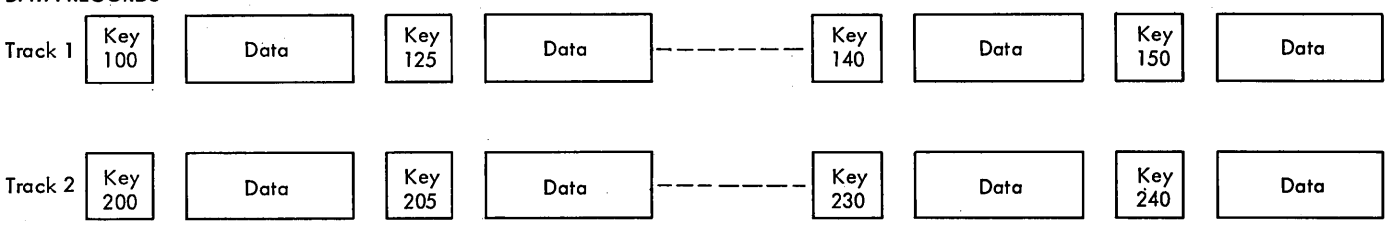


Figure 22. Example of Data Records, as Originally Organized on Tracks 1 and 2

To add a record by insertion, ISFMS searches the established indices first to determine on which track the record must be inserted. The keys of the last records on the tracks in the originally organized file determine the track where an inserted record belongs. A record is always inserted on the track where:

1. The last key is higher than the insertion, and

2. The last key of the preceding track is lower than the insertion.

For example, assume Tracks 1 and 2 are organized with the record keys shown in Figure 22. Then records with keys such as 151, 175, 199, 215, and 239 are inserted on Track 2 (or in the related overflow chain that has developed). Any key lower than 150 is added to either Track 0 or Track 1; any key higher than 240 belongs to Track 3 or above. The track indices always retain the highest key of each track as it was originally organized.

After the proper track is determined, ISFMS searches the individual records on the track or overflow area (if necessary) to find where the record belongs in key order. This results in either of two conditions:

1. The record falls between two records presently on the track. ISFMS adds the record by inserting it in the proper sequence and shifting each succeeding record one record location higher on the track, until the end record is forced off the track. ISFMS transfers the end record to the overflow area, and prefixes the record (data area) with a sequence-link field. The first time a record is inserted on a track, the sequence link of the overflow record indicates that this is the highest record associated with the track. Thereafter, the sequence-link field of each overflow record points to the next-higher record for that track.

ISFMS also updates the track index to reflect this change. The first index record for the track has the key field changed to indicate the new last-record located on the track. The second index record for the track has the track address (in the data area) changed to point to the address of the overflow record. If a record with key 105 is added to a file organized as shown in the previous illustrations and if the overflow area is located on Track 8, the track index records contain the information shown in Figure 23.

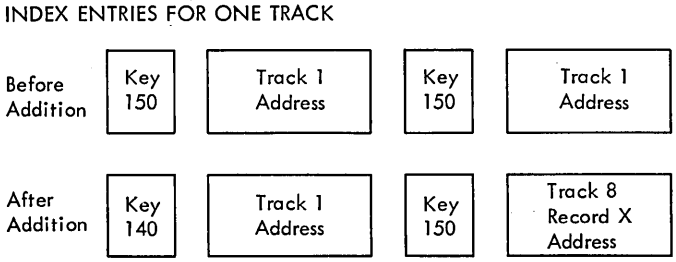


Figure 23. Example of Track Index Entries Before and After Addition of a Record on Track 1

2. The record falls between the last record presently on the track and the last record originally on the track. Thus, it belongs in the overflow area. ISFMS writes the record in the overflow area following the last record previously written. ISFMS searches through the chain of records associated with the corresponding track for this record and identifies the sequential position the record should take. Then the sequence-link fields of the new record, and of the record preceding it by sequential key, are adjusted to point to the proper records. If records 150, 140, and 130 are already in the overflow area and record 135 is to be added, for example, the sequence-link fields of records 130 and 135 must be adjusted (Figure 24).

RECORD	SEQUENCE-LINK FIELD	
	Before Addition	After Addition
130	140	135
135 (New Record)	—	140

Figure 24. Example of Sequence-Link Fields Adjusted for Addition of a Record (135)

Overflow-Area Option: The location of the overflow area(s) for a logical file may be specified by the user. The overflow areas may be built by one of three methods:

1. Overflow areas for records may be located on each cylinder within the prime area that is specified by a Job Control XTENT card for the data file. In this case the user must specify the number of tracks to be reserved for overflow on each cylinder occupied by the file. The overflow records that occur within a particular cylinder are written in the cylinder overflow area for that cylinder.

   The number of tracks to be reserved for each cylinder overflow area must be specified in the DTFIS entry CYLOFL when a file of records is to be loaded and when records are to be added to an organized file.

2. An independent overflow area may be specified for storing all overflow records for the logical file. This area may be on the same pack with the data records, or on a different pack that is on-line. However it must be contained within one disk pack. A Job Control XTENT card must be included when the program is executed to specify the area of the disk pack to be used for this overflow area. This card must be the last XTENT card that Job Control reads for the file.

3. Both cylinder overflow areas (method 1) and an independent overflow area (method 2) may be used. In this case overflow records are placed first in the cylinder overflow areas within the data file. When any cylinder overflow area becomes filled, the additional overflow records from that cylinder are written in the independent overflow area. The specifications required for both methods 1 and 2 must be included for this combined method of handling overflows.

Example of an Organized File

A simplified example of a file organized on disk by the Indexed Sequential File Management System is shown schematically in Figure 25. The assumptions made and the items to be noted are:

1. The track index occupies part of the first track, and data records fill the rest of the track.

2. The data records occupy part of Track 0 and all of Tracks 1-7. Tracks 8 and 9 are used for overflow records in this cylinder.

3. The master index is located on Track X on a different cylinder. The cylinder index is located on Tracks X+1 through X+20.

4. A dummy entry signals the end of each index.

5. The file was originally organized with records as follows:

Track	Records
0	5-75
1	100-150
2	200-240
.	.
.	.
.	.
7	900-980

6. The track index originally had two entries for each track. It now shows that overflow records have occurred for Tracks 1 and 7.

7. Records 150, 140, and 130 were forced off the track by insertions on the track. Record 135 was added directly in the overflow area.

8. A sequence-link field (SL) has been prefixed to each overflow record. The records for Track 1 can be searched in sequential order by following the SL fields:

Record	Sequence-Link field (SL)
130	SL points to record with key 135.
135	SL points to record with key 140.
140	SL points to record with key 150.
150	End of search. (Key 150 was the highest key on Track 1 when the file was loaded.)

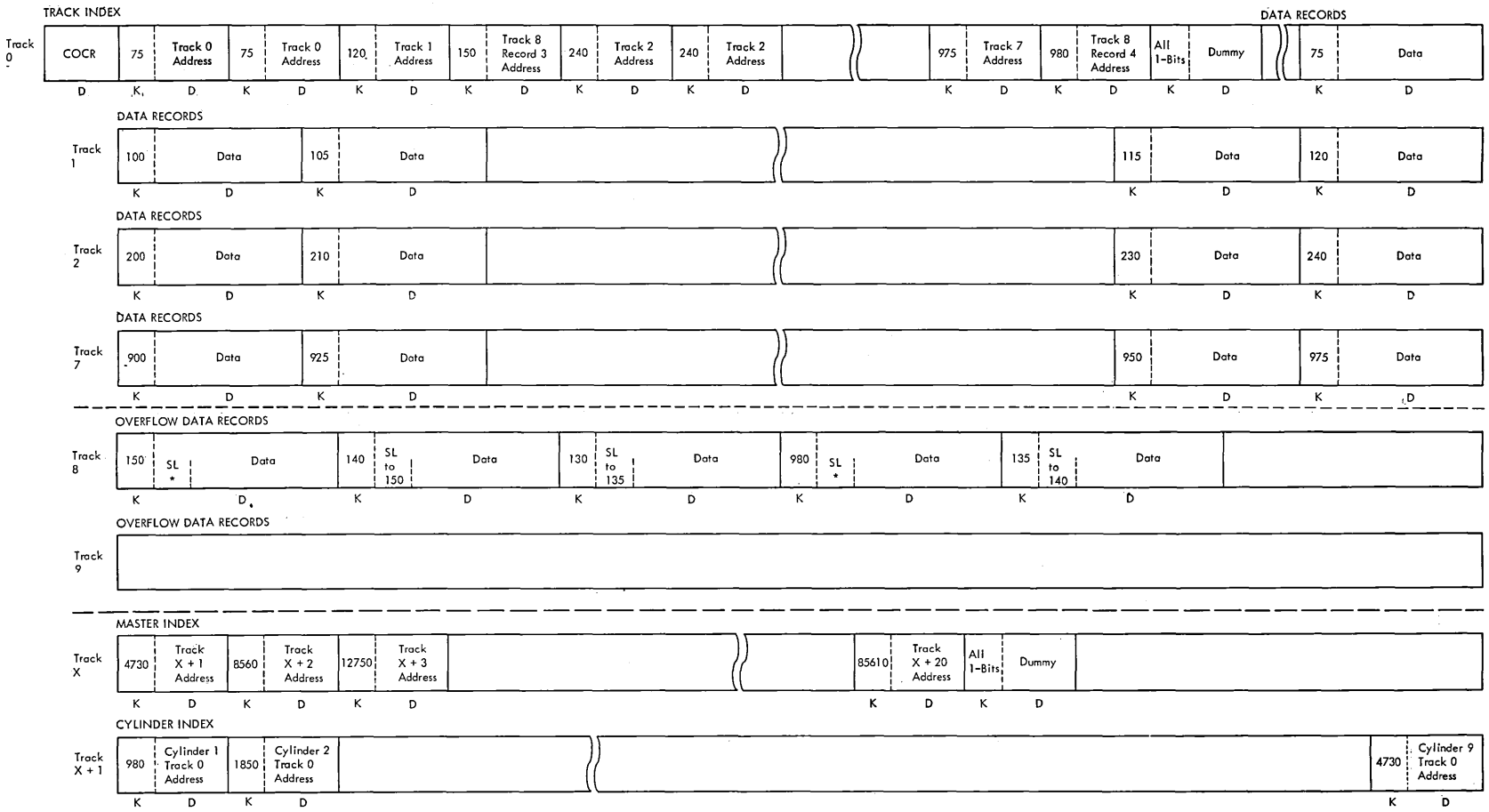


Figure 25. Schematic of a File on Disk, Organized by ISFMS

9. When the file was loaded, the last record on Cylinder 1 was Record 980, on Cylinder 2 Record 1850, and on Cylinder 9 Record 4730. This is reflected in the cylinder index.

10. When the file was loaded, the last entry on track X+1 (first track of Cylinder Index) was Record 4730, on track X+2 Record 8560, on track X+3 Record 12750, and on track X+20 Record 85610. This is reflected in the master index.

11. When cylinder overflow areas are used, the first record (Record 0) in the track index for a cylinder is the Cylinder Overflow Control Record (COCR). It contains the address of the last overflow record on the cylinder and the number of tracks remaining in the cylinder overflow area. When the number of remaining tracks is zero, overflow records are written in the independent overflow area.

MACRO INSTRUCTIONS TO LOAD OR EXTEND A DISK FILE BY ISFMS

The function of originally loading a file of presorted records onto disk and the function of extending the file by adding new presorted records beyond the previous high record are essentially the same. Both are considered a LOAD operation (specified by the DTFIS entry IOROUT), and they both use the same macro instructions in the problem program.

The areas of the disk packs used for the file are specified by Job Control XTENT cards. The areas are: the prime area where the data records are written, a cylinder index area where the user wants ISFMS to build the cylinder index, and a master index area if a master index is to be built (specified by the DTFIS entry MSTIND).

During the load operation, ISFMS builds the track, cylinder, and master (if specified) indices. If either the data records, the cylinder index, or the master index exceeds the area provided for it, ISFMS branches to a user's routine (specified by the DTFIS entries DTAREX, CYNDEX, and MANDEX, respectively). If the data area has been exceeded, an ENDFL instruction may be included in the DTAREX routine to prepare the file for closing. This permits the remaining records to be treated as extensions. To continue loading, the user must supply a new XTENT card for the area exceeded and restart the job.

Whenever any type of processing is being done for an Indexed Sequential file, all packs of a multipack file must be on-line.

Whenever an organized file is to be extended, the identical Job Control DLAB card that was originally used when the file was loaded must be used again. However, if an organized file is to be loaded on the same disk pack a second time, the user should prepare a new Job Control DLAB card to prevent his file from being treated as an extension of itself. In the new DLAB card, the user must change either the filename, creation date, or expiration date.

Name of First Control Section: Whenever the LOAD function is specified, the user must place a name in the START card in his source deck to define the first control section. This permits ISFMS to use an ENTRY statement to identify a linkage symbol defined in this control section and used in subsequent control sections. However, the user-written portion of the program will not be in the control section named by the START card. Instead, it will be in a control section named by ISFMS. The name assigned by ISFMS is FilenameU. Filename is the name of the last file for which LOAD is specified, unless sequential retrieval is specified for another file in the program (see Macro Instructions for Sequential Retrieval by ISFMS).

As a result of these conditions, the programmer must be sure to use the applicable name if he wishes to continue his first control section after it has been interrupted by a dummy section or a different control section. He must use the name assigned by ISFMS (rather than the START card) whenever LOAD is specified for any file in the source program.

Macro Instructions

Three different macro instructions are always required in the problem program to load original or extension records into the logical file on disk.

SETFL Macro

Name	Op	Operand
	SETFL	filename

The SETFL (set file load mode) macro instruction causes ISFMS to set up the file so that the load function can be performed. The symbolic name of the file to be loaded is the only parameter required in this instruction. This name is the same as that

specified in the DTFIS header entry for this file.

When a file is being created, ISFMS formats the track index, cylinder index, and master index (if used) with dummy entries. When a file is being extended, the SETFL macro simulates a restart condition so that the load function can proceed as if it were making its initial run.

WRITE Macro

Name	Op	Operand
	WRITE	filename,NEWKEY,IS

When a WRITE macro instruction with the parameter NEWKEY is issued in the problem program between a SETFL instruction and an ENDFL instruction (the third macro required for loading), it causes ISFMS to load a record onto disk. It requires three parameters. The first specifies the symbolic name of the file, as specified in the DTFIS header entry. The second parameter must be the word NEWKEY. The third parameter must be the letters IS to indicate processing by the indexed sequential system.

Before issuing the WRITE instruction for an unblocked record, the problem program must store the key of the record followed by the data in a work area (specified by DTFIS WORKL).

WORKL - Unblocked Records

Key	Data (With or without embedded key)

The ISFMS routines construct the I/O area (see Figure 18) by moving the data record to the data area, moving the key to the key area, and building the count area. Then they transfer the record to disk storage.

For blocked records, the problem program must store only the logical record in a work area (specified by DTFIS WORKL). The key must be part of the data record (location specified by DTFIS KEYLOC), but it must not be stored separately as it is for unblocked records.

WORKL - Blocked Records

Data (With embedded key)

The ISFMS routines move each data record to the I/O area. After the block of records in the data portion of the I/O area is completed, ISFMS moves the key of the highest record in the block into the key portion of the I/O area. Then ISFMS constructs the count portion and transfers the records to disk storage.

As records are transferred, ISFMS performs both a sequence check (to ensure that the records are in order by key) and a duplicate-record check. If an out-of-sequence record or a duplicate-record key is detected, ISFMS branches to the corresponding user's routine (specified by the DTFIS entries SQCHEX and DUPREX, respectively).

After each record is written, ISFMS makes the ID of that record available to the problem program. The ID is located in an 8-byte field labeled filenameH. In this case filename must be 5 characters long. For example, if the file name in the DTFIS header entry is PAYRD, the ID field is addressed by PAYRDH. By reference to this field, the ID of any selected records can be punched or printed for later use. This will be required if the user plans to retrieve records in sequential order starting with the ID of a particular record (see SETL Macro).

As records are loaded onto disk, ISFMS writes track-index records each time a track is filled, writes a cylinder-index record each time a cylinder is filled, and writes a master-index record (if DTFIS MSTIND is specified) each time a track of the cylinder index is filled. When a track index is completed, ISFMS writes a dummy record following the last index record. This is used in subsequent operations to indicate the end of the index and the beginning of data records.

ENDFL Macro

Name	Op	Operand
	ENDFL	filename

The ENDFL (end file load mode) macro instruction ends the mode initiated by the SETFL macro. The symbolic name of the file that has been loaded is the only parameter required in this instruction. This name is the same as the name specified in the DTFIS header entry and the SETFL instruction for this file.

The ENDFL macro performs a close-like operation for the file that has been loaded. It writes the last block of data records, if necessary, and then writes an

end-of-file record after the last data record. It also writes any index entries that are needed.

Note: At least one or more records must be loaded before the ENDFL macro instruction can be executed. If no records are loaded before the ENDFL macro is issued, the ENDFL macro returns control immediately to the user's program without writing an end-of-file record. The file then has to be reloaded with a new DLAB (disk label) card.

MACRO INSTRUCTIONS TO ADD RECORDS TO A FILE BY ISFMS

After a file has been organized on disk, new records can be added to the file. Each record is inserted in the proper place sequentially by key. This function is controlled by specifying ADD or ADDRTR in the DTFIS entry IOROUT.

The file may contain either blocked or unblocked records, as specified by the DTFIS entry RECFORM. When the file contains blocked records, the user must provide ISFMS with the location of the key field within each record. This is specified in the DTFIS entry KEYLOC. The records to be added, however, must be inserted one record at a time, and they must contain a key field in the same location as the records already in the file.

Whenever the addition of records is to follow sequential retrieval (IOROUT=ADDRTR), the sequential-retrieval instruction ESETL must be issued before the first record is added to the file (see ESETL Macro).

Macro Instruction

One macro instruction is available for use in the problem program, for adding records to a file.

WRITE Macro

Name	Op	Operand
	WRITE	filename,NEWKEY,IS

When a WRITE macro instruction with the parameter NEWKEY is issued in the problem program and the DTFIS entry IOROUT specifies ADD (or ADDRTR), ISFMS adds the record to the previously organized file. This is the same instruction, and requires the same parameters, as the WRITE instruction that is used for loading a file. The only difference is that DTFIS IOROUT specifies ADD (or ADDRTR) instead of LOAD.

For adding records, the problem program must store the key of the record followed by data in the work area specified by DTFIS WORKL, before issuing the WRITE instruction. ISFMS then constructs the I/O area and transfers the record to disk storage. As records are transferred, ISFMS checks for duplicate record keys and branches to the user's routine (specified by DTFIS DUPREX) if a duplication is found.

To insert the new record properly, ISFMS searches the indices to find the correct track for the record. If the correct track is not an overflow track, ISFMS then searches the track for the correct position sequentially. The record is inserted, all following records are shifted, and the highest record on the track is transferred to the appropriate overflow area. This is the cylinder overflow area if CYLOFL has been specified and if the area has not been filled. If the cylinder overflow area does not have space available, or if only an independent overflow area has been specified by an XTENT card, the overflow record is transferred to the independent overflow area. If the cylinder overflow area has been filled and an independent area has not been specified, however, there is no place to store the overflow record. ISFMS then branches to the user's routine specified by the DTFIS entry ADAREX. The user should specify an independent overflow area to store this and other overflow records. ISFMS also branches to the ADAREX routine if an independent overflow area is specified but has become filled. In either case the job should be restarted with a new XTENT card included for the independent overflow area.

Whenever records are to be inserted into a logical file of blocked records, ISFMS first locates the correct block on the track (after the proper track has been found). The block is determined by checking the key areas of the disk records. Each key area contains the key of the highest logical record in the corresponding block. Then ISFMS examines the key field within each logical record in the block to find the exact position to insert the record. After the record is inserted, the following logical records on the track are shifted and reblocked, and the key areas are adjusted. The last logical record on the track is moved to the overflow area.

If the proper track for a record is an overflow track, ISFMS writes the record, preceded by a sequence-link field in the data area of the record, and adjusts the appropriate linkages to maintain sequential order by key. Similar to the operation

described for the end record in the preceding paragraphs, ISFMS writes the new record in either the cylinder overflow area or the independent overflow area, or it branches to the user's routine if necessary.

If the new record is higher than all records presently in the file, ISFMS checks to determine if the last track containing data records is filled. If it is not, the new record is added, replacing the end-of-file record. The end-of-file record is written in the next record location on the same track, or on the following track in the prime data area. Another track must be available within the file limits. If the end-of-file record is the first record on a track, the new record is written in the appropriate overflow area. After each new record is inserted in its proper location, ISFMS adjusts all indices that are affected by the addition.

MACRO INSTRUCTIONS FOR RANDOM RETRIEVAL BY ISFMS

When a file has been organized by ISFMS, records can be retrieved in random order for processing and/or updating. Retrieval must be specified in the DTFIS entry IOROUT. Random processing must be specified in the DTFIS entry TYPEFLE, and updating (if used) must be specified in DTFIS UPDATE. If a multipack file is being processed, all packs must be on-line.

Because random reference to the file is by record key, the problem program must supply the key of the desired record to ISFMS. To do this the key must be stored in the main-storage key field specified by the DTFIS entry KEYARG. The specified key designates both the record to be retrieved and the record to be written back into the file in an updating operation.

Name of First Control Section: When DTF IOROUT=RETRVE and TYPEFLE=RANDOM are used, ISFMS names the first CSECT in the DTFIG macro IICSCT. If this DTF is used alone, the user's program will come under this CSECT name. If it is used in the same program with other DTFs, the name of the user's first control section will be determined as explained under Macro Instructions for Sequential Retrieval by ISFMS.

Macro Instructions

Two macro instructions are available for use in the problem program for retrieving and updating records randomly.

READ Macro

Name	Op	Operand
	READ	filename,KEY,IS

The READ instruction causes ISFMS to retrieve the specified record from the file. This instruction requires three parameters. The first parameter specifies the symbolic name of the file from which the record is to be transferred to main storage . This name is the same as the name specified in the DTFIS header entry for this file. The second parameter must be the word KEY. The third parameter must be the letters IS to indicate processing by the indexed sequential system.

To locate the record ISFMS searches the indices to determine the track on which the record is stored, and then searches the track for the specific record. When the record is found, ISFMS transfers it to the I/O area specified by the DTFIS entry IOAREAR. The IOCS routines also move the data portions of the record to the specified work area if the DTFIS entry WORKR is included in the file definition.

When records are blocked, ISFMS transfers the block that contains the specified record to the I/O area. It makes the individual data record available for processing either in the I/O area or the work area (if specified). For processing in the I/O area, ISFMS supplies the address of the record in the register specified by DTFIS IOREG.

If ISFMS does not find the specified record, it branches to the user's routine specified by the DTFIS entry RTRVEX.

WRITE Macro

Name	Op	Operand
	WRITE	filename,KEY,IS

The WRITE instruction with the parameter KEY is used for random updating. It causes ISFMS to transfer the specified record from main storage to disk storage. This instruction requires three parameters. The first parameter specifies the symbolic name of the file to which the record is to be transferred. This name is the same as the name specified in the DTFIS header entry and in the preceding READ instruction for this file. The second parameter must be the word KEY. The third parameter must be the letters IS.

ISFMS rewrites the record retrieved by the previcus read instruction for the same file. The key specified in the key field for the READ instruction determines where the record is written. The key need not be specified again, ahead of the WRITE instruction.

MACRO INSTRUCTIONS FOR SEQUENTIAL RETRIEVAL BY ISFMS

When a file has been organized by ISFMS, records can be retrieved in sequential order by key for processing and/or updating. Retrieval must be specified in the DTFIS entry IOROUT. Sequential processing must be specified in the DTFIS entry TYPEFLE, and updating (if used) must be specified in DTFIS UPDATE.

Although records are retrieved in order by key, sequential retrieval can start at a record in the file identified either by key or by the ID (identifier in the count area) of a record in the prime data area. Or, sequential retrieval can start at the beginning of the logical file. The user specifies, in the SETL macro, the type of reference he will use in the problem program.

Whenever the starting reference is by key and the file contains blocked records (RECFORM=FIXBLK), the user must also provide ISFMS with the position of the key field within the records. This is specified in the DTFIS entry KEYLOC. To search for a record, ISFMS first locates the correct block by the key in the key area of the disk record. (The key area contains the key of the highest record in the block.) Then, ISFMS examines the key field within each record in the block to find the specified record.

Whenever any type of processing is being done for an Indexed Sequential file, all packs of a multipack file must be on-line.

Name of First Control Section: When sequential retrieval is specified, the user must place a name in the START card in his source deck to define the first control section. This permits ISFMS to use an ENTRY statement to identify a linkage symbol defined in this control section and used in subsequent control sections. However, the user-written portion of the program will not be in the control section named by the START card. Instead, it will be in a control section named by ISFMS. The name assigned by ISFMS to the user's first control section is Filename3 whenever sequential retrieval alone (IOROUT=RETRVE and TYPEFLE=SEQNTL or RANSEQ) is specified for any file in the source program. Filename is the DTFIS header name of the first file (in input sequence) for which sequential retrieval is specified.

If sequential retrieval alone (IOROUT=RETRVE) is not specified for any file, the name assigned to the user's first control section depends upon the other functions (IOROUT) and/or types of processing (TYPEFLE) specified for the files in the source program. If either sequential add-retrieve (IOROUT=ADDRTR and TYPEFLE=SEQNTL or RANSEQ) or load (IOROUT=LOAD) is specified for any file, ISFMS names the user's first control section. If both are specified, for different files in the source program, the naming function that occurs last (in input sequence) determines the name that ISFMS assigns, as follows:

- If sequential add-retrieve (IOROUT=ADDRTR and TYPEFLE=SEQNTL or RANSEQ) occurs last, the control section name is Filename3. Filename is the DTFIS header name of the last file for which sequential add-retrieve has been specified.

- If the load function (IOROUT=LOAD) occurs last, the control section name is FilenameU. Filename is the DTFIS header name of the last file for which the load function has been specified.

If neither sequential retrieval, sequential add-retrieve, nor load is specified for any file in the source program, ISFMS does not name the user's first control section. In this case, the first control section is named by the name (if any) specified in the START card.

As a result of these conditions, the programmer must be sure to use the applicable name if he wishes to continue his first control section after it has been interrupted by a dummy section or a different control section. He must use the name assigned by ISFMS (rather than the START card) whenever sequential retrieval, sequential add-retrieve, or load is specified for any file in the source program.

Macro Instructions

Four macro instructions are available for use in the problem program for retrieving and updating records sequentially.

SETL Macro

Name	Op	Operand
	SETL	filename,BOF
	SETL	filename,KEY
	SETL	filename,idname

The SETL (set limit) macro instruction initiates the mode for sequential retrieval, and initializes the ISFMS routines to begin retrieval at the specified starting address. It requires two parameters. The first parameter specifies the symbolic name of the file (specified in the DTFIS header entry) from which records are to be retrieved.

The specification entered as the second parameter depends on the starting reference that is used for the file: beginning of the file, key, or ID.

BOF: If retrieval is to start at the beginning of the logical file, the letters BOF (beginning of file) must be entered as the second parameter.

KEY: If retrieval is to start at the record that contains a specific key (control information), the word KEY must be entered as the second parameter. In this case, the key of the desired record must be stored in the main-storage key field specified by the DTFIS entry KEYARG. The key must be supplied in this field before issuing the SETL instruction for the file. If ISFMS cannot find the starting record, it branches to the user's routine specified by the DTFIS entry RTRVEX.

idname: if retrieval is to start at the record that has a specific disk address (identifier - ID), the second parameter must specify the symbolic name of a field in main storage. The disk address of the first record to be retrieved must be stored in this field before the SETL instruction is issued for the file. This idname field must be eight bytes long and the user must supply the record identification (MBBCCHHR) as listed here. All numbers must be supplied in binary notation.

Byte	Ident.	Con-tents	Information
0	M	2-255	Number of the extent in which the starting record is located. The extents for the file must be numbered so that the first extent is number 2, the second extent number 3, etc.
1-2	B,B	0,0	Reserved for cell number, which relates to the IBM 2321 Data Cell Drive. These two bytes are always zero for 2311 disk-storage references.
3-4	C,C	0,1-199	Number of the cylinder (1-199) in which the record is located. The first byte is always zero, and the second byte specifies one of the 199 cylinders available for data records in a disk pack. These two bytes with the next two (HH) provide the track identification.
5-6	H,H	0,0-9	Number of the read/write head (0-9) that applies to the record. The first byte is always zero, and the second byte specifies one of the ten disk surfaces in a disk pack.
7	R	1-254	Sequential number of the record on the track.

ESETL Macro

Name	Op	Operand
	ESETL	filename

The ESETL (end set limit) macro instruction ends the sequential mode initiated by the SETL macro. The symbolic name of the file, which must be the same as the name specified in the DTFIS header entry and in SETL, is the only parameter required in this instruction.

If blocked records are specified, ESETL writes the last block back into the disk file in its previous location, if necessary.

When the program requires sequential retrieval followed by the addition of records to a file (IOROUT=ADDRTR), the ESETL macro instruction must be issued at the end of the sequential retrieval and before a WRITE instruction is issued for the first addition. If sequential retrieval is to be restarted after the

additions are completed, the key or ID of the last record retrieved must be saved. To return to sequential retrieval, the SETL macro instruction must be issued again.

GET Macro

Name	Op	Operand
	GET	filename,,IS
	GET	filename,workname,IS

The GET macro instruction causes ISFMS to retrieve the next record in sequence from the file. It can be written in either of two forms, depending on where the record is to be processed.

The first form is used if records are to be processed in the I/O area (specified by DTFIS IOAREAS). It requires three parameters, the second of which is blank and represented by a comma. The first parameter specifies the symbolic name of the file from which the record is to be retrieved. This is the same name as that specified in the DTFIS header entry and in the SETL macro instruction for this file. ISFMS transfers the record from this file to the I/O area, and the record is available for the execution of the next instruction in the problem program. The third parameter must be the letters IS to indicate processing by the indexed sequential system. ISFMS makes the data portion of each record available by supplying its address in the register specified by the DTFIS entry IOREG. When the unblocked records are specified, the key portion of each record is available at the area specified in the DTFIS entry IOAREAS.

The second form of the GET instruction is used if records are to be processed in a work area (DTFIS specifies WORKS). It requires three parameters. The first is the symbolic name of the file, the second is the symbolic name of the work area, and the third parameter is the letters IS. ISFMS transfers both the key of the record and the data to the specified work area for unblocked records. For blocked records, ISFMS transfers only the data portion of the record to the specified work area. The record is available for the execution of the next program instruction.

If blocked records and updating are specified in the file definition, each GET that transfers a block of records to main storage will, if necessary, also write the preceding block back into the disk file in its previous location. GET writes the preceding block if a PUT instruction has been issued for at least one of the records in the block. If no PUT instructions have been issued, updating is not required for this block and GET does not write the block.

PUT Macro

Name	Op	Operand
	PUT	filename,,IS
	PUT	filename,workname,IS

The PUT macro instruction is used for sequential updating of a disk file, and causes ISFMS to transfer records to the file in sequential order. It must be preceded by a GET instruction for that file. The PUT instruction may be written in either of two forms, depending on where records are processed.

The first form is used if records are processed in the I/O area (specified by DTFIS IOAREAS). It requires three parameters, the second of which is blank and represented by a comma. The first parameter specifies the symbolic name of the file to which the records are to be transferred. This is the same name as specified in the DTFIS header entry, the SETL instruction, and the GET instruction for this file. The third parameter must be the letters IS to indicate processing by the indexed sequential system.

The second form of the PUT instruction is used if records are processed in a work area. It requires three parameters. The first is the symbolic name of the file, the second is the symbolic name of the work area, and the third is the letters IS. The work-area name may be the same as that specified in the preceding GET for this file, but this is not required. ISFMS moves the record from the work area specified in the PUT instruction to the I/O area specified for the file in the DTFIS entry IOAREAS.

When unblocked records are specified, each PUT writes a record back onto the disk file in the same location from which it was retrieved by the preceding GET for this file. Thus, each PUT updates the last record that was retrieved from the file. If some records do not require updating, a series of GET instructions can be issued without intervening PUT instructions. Therefore, it is not necessary to rewrite unchanged records.

When blocked records are specified, PUT instructions do not transfer records to the disk file. Instead, each PUT indicates that the block is to be written after all the records in the block have been

processed. When processing for the block is complete and a GET is issued to read the next block into main storage , that GET also writes the completed block back into the file in its previous location. If a PUT instruction is not issued for any record in the block, GET does not write the completed block. At the end of the file the ESETL macro instruction writes the last block processed, if necessary.
NOTE: The user should insert an LTORG statement in his assembler deck whenever more than one control section is generated by the ISFMS macros (Load and/or Retrieve functions). This is to provide addressability for the generated literals in the imperative macros.

PROCESSING WITH STR DEVICES

Logical IOCS provides macro instructions for processing with STR (Synchronous Transmitter-Receiver) devices. These devices can be remotely attached to a System/360, Model 30, 40, 50, 65 or 75, through an IBM 2701 Data Adapter Unit, equipped with an IBM Synchronous Data Adapter, Type I.

Whenever STR macro instructions are used to transmit and receive data or furnish line control, each synchronous data adapter must be defined by the declarative macro DTFSN (Define The File for SyNchronous Transmitter-Receiver use). The operand entries for the DTFSN (and for the DTFRF macro) are described under File Definition Macros.

Before processing can be done with STR devices, the adapter must be initialized. Logical IOCS provides a unique imperative macro instruction, SOPEN (open STR adapter), to initialize the adapter for STR processing.

SOPEN Macro

The SOPEN (open STR adapter) macro instruction turns on the adapter, establishes the mode, and establishes synchronization. The SOPEN operands determine the line interfaces on the adapter, the data transmission rate, the data transmission mode, and the type of data checking to be performed. Where the dial option is present, SOPEN may also dial a number, monitor for ringing, and establish a connection.

SOPEN must be issued for a line before a READ, WRITE, or CNTRL macro is issued for that line.

Completion of a SOPEN macro is indicated in the traffic (or wait) bit and the unit exception bit in the channel command block (CCB). The WAIT macro or the WAITM macro should be used to check for completion of the SOPEN macro.

The lost data and unit exception (end-of-file) bits should be checked by the problem program upon completion of a SOPEN macro instruction. If SOPEN could not establish synchronism, and an operator reply of 4 was given, the lost data and unit exception bits will be on. If the remote terminal attempted to transmit before synchronism was established, the unit exception bit will not be on, indicating the SOPEN macro could not complete successfully.

SOPEN can be reissued with the same or different operands after a SCLOS macro to begin transmission or reception of data to the same or another device.

NAME	OPERATION	OPERAND	COMMENTS
	SOPEN	DTFNAME=dtfname	Symbolic name of DTFSN macro
		DIAL=IN OUT	Answering Calling
		°INTFAC=A # B BOTH	*Which line interface on the adapter to use. BOTH may only be specified when DIAL=IN.
		#INTLRC=NO YES	LRC check should be performed on record or group mark.
		INTLRCB=NO YES	When INTFAC=BOTH, INTLRCB provides for LRC check for interface B as INTLRC provides for interface A.
		#SPEED=X Y Z CLOCK	*data transmission rate expressed in characters/second.
		SPEEDB=X Y Z CLOCK	When INTFAC=BOTH, SPEEDB provides the data transmission rate for interface B, as SPEED provides for interface A.
		#MODE=FULL FOUR TWO	Full duplex Four wire half duplex Two wire half duplex (data transmission modes)
		MODEB=FULL FOUR TWO	When INTFAC=BOTH, MODEB determines the data transmission made for interface B, while MODE provides for interface A.
		NUMB=n	General register (2-11) loaded by problem program with address of DIALO macro. Must be supplied when DIAL=OUT.

Underlined choices will be assumed if operand is omitted.

°INTFAC must be "=A" if Dual Communications Interface feature is not present.

#Supplied by DIALO macro when SOPEN DIAL=OUT.

*For further reference see the section Communications: Synchronous Adapter in the publication IBM 2701 Data Adapter Unit: Principles of Operation (Form A22-6864).

Figure 26. SOPEN Macro

The SOPEN macro instruction is coded with the keyword operands as shown in Figure 26.

Where the problem program wishes to use the facilities of STR devices attached over a dial network (with the Automatic Call Feature installed), logical IOCS provides a declarative macro instruction, DIALO (dial out), to supply the dial address and various parameters required by the SOPEN macro.

NAME	OPERATION	OPERAND	COMMENTS
	DIALO		
		LENGTH=n	The total number of digits in the telephone number to be dialed (sum of "ACODE" and "DIGITS").
		ACODE=number	The area code, or any digits not included in "DIGITS".
		DIGITS=number	The telephone number to be dialed (maximum of 7 digits).
		#INTFAC=A B	*Which line interface on the adapter to use.
		#INTLRC=YES NO	LRC check should be performed on record or group mark.
		#SPEED=X Y Z CLOCK	* Data transmission rate expressed in characters/ second.
		#MODE=FULL FOUR TWO	Full duplex Four-wire half duplex Two-wire half duplex (data transmission modes)

Underlined choices will be assumed if operand is omitted.

* For further reference see the section, Communications: Synchronous Adapter, in the publication IBM 2701 Data Adapter Unit: Principles of Operation (Form A22-6864).

See SOPEN Macro.

Figure 27. DIALO Macro

DIALO Macro

The declarative DIALO (dial out) macro instruction generates the proper constants required by the SOPEN macro to dial and synchronize with the selected terminal.

The DIALO macro operands provide the actual number to be dialed and the number of digits in that number. Other operands specify the line interface on the adapter, the data transmission speed, and the data transmission mode. For a further explanation of DIALO operands see Figure 27.

Once the SOPEN and DIALO macros have established the proper connection between the STR devices, logical IOCS provides several imperative macro instructions for the actual processing (transmission, reception, and control) of data.

READ Macro

```
|Name|Op   |Operand                |
|----|-----|-----------------------|
|    |READ |dtfname,STR            |
```

The READ macro instruction is used to read one record from the STR device specified in the DTFSN (referenced in the READ macro by "dtfname").

The area into which the data will be read must be determined by the problem program before the READ is issued. The problem program must store the starting address of the designated area into the "abuck" portion of the expanded STR CCB (see Figure 42). The problem program must also store the length of the designated area in "lbuck" in the STR CCB.

Before a READ macro instruction is issued, the line must be SOPEN'ed, and a CNTRL macro with the operand PREP (prepare) must be issued and successfully completed. The problem program must be sure that the READ is complete (using the WAIT or WAITM

macro) before issuing another READ for the same adapter.

WRITE Macro

Name	Op	Operand
	WRITE	dtfname,STR

The WRITE Macro instruction is used to write one record to the STR device specified in the DTFSN macro (referenced in the WRITE macro as "dtfname").

The area from which the data will be written must be specified by the problem program before the WRITE is issued. The problem program must store the starting address of the designated area into the "abuck" portion of the expanded STR CCB (Figure 42). The problem program must also store the length of the designated area in "lbuck" in the CCB.

Before a WRITE macro instruction is issued, the line must be SOPEN'ed, and a CNTRL macro with the operand INQ (inquiry) must be issued and successfully completed. The problem program must be sure that the WRITE is complete (using the WAIT or WAITM macro) before issuing another WRITE for the same adapter.

CNTRL Macro

Name	Op	Operand
	CNTRL	dtfname,code,n

The CNTRL (control) macro instruction provides orders to the Data Adapter and the terminal. Orders apply to physical nondata operations of a unit and are peculiar to the unit involved.

CNTRL requires either two or three operands. The first operand (dtfname) must be the symbolic name of the DTFSN macro or line on which to perform the function specified in the code operand of the CNTRL macro. The second operand specifies the mnemonic codes (see Figure 14) for the operation to be performed:

EOF (End of Transmission) ends the transmission to an STR device. EOF may not be used during a read sequence (the time between a PREP issued and an EOF received).

INQ (Inquiry) sends INQ to the STR device. INQ must be successfully completed before a WRITE is issued. The INQ RCVD (inquiry received) bit on in the Expanded STR CCB (see Figure 42) should be checked. If INQ RCVD is ON, the remote terminal is attempting to transmit and the CNTRL INQ cannot complete successfully. INQ may be issued following a CNTRL PREP (when INQ is issued following a PREP, the PREP operation is discontinued). INQ may not be issued during a read sequence, or during a write sequence (the time between an INQ issued and EOF).

PREP (Prepare) monitors the line for receiving INQ, EOF, or TEL from the STR device. Before a READ is issued, PREP must be successfully completed. PREP may not be issued during a write sequence.

TEL (Alternate Mode) sends the TEL signal to an STR device. TEL may not be issued during a read or write sequence.

The third operand (n) specifies a count to be used if other than:

EOF - 2
TEL - 2
INQ - 10
PREP - Not applicable

This count is the number of times the particular signal will be sent until the correct reply to that signal is received. When the count is exceeded, an error condition results. An operator message is given.

The DTFSN must be SOPEN'ed before any CNTRL macro is issued.

If the problem program issues an EOF, INQ, PREP, or TEL out of sequence (during a read or write sequence), an operator message is issued, the STR lines are properly disabled, and the job is terminated with a dump.

The problem program must be sure that a CNTRL macro is complete (using the WAIT or WAITM macro) before issuing another imperative STR macro instruction. Exceptions to this are: SCLOS (close STR adapter), which may be issued at any time, and an INQ issued following a PREP, which has been explained previously.

CDCNV Macro

Name	Op	Operand
	CDCNV	type,startaddr,length

The CDCNV (code conversion) macro instruction provides for the conversion of the internal code of the transmitter (EBCDIC, BCD, or Binary) to the standard STR transmission code (fixed count four-out-of-eight [4/8]), or for the conversion of 4/8 code to the internal code of the receiver.

The first of three required operands, "type", is coded as A, B, C, D, E, or F, and specifies:

A - Convert for transmitting EBCDIC (256 character set) from System/360.

B - Convert for receiving EBCDIC (256 character set) from the terminal.

C - Convert for transmitting EBCDIC (256 character set) to a 1978 in column binary mode.

D - Convert for receiving EBCDIC (256 character set) from a 1978 in column binary mode.

E - Convert for transmitting BCD (56 character set) or binary (64 character set) to any STR device. All invalid characters are replaced with colon (1248....).

F - Convert for receiving BCD (56 character set) or binary (64 character set) from any STR device.

The second operand (startaddr) specifies the symbolic name of a full word containing the actual address of the leftmost byte of the field to be converted.

The third operand (length) specifies the symbolic name of a half word containing the number of bytes to be converted.

For types A and C, each EBCDIC character is converted to two 4/8 characters. Therefore, the length of the field provided for the converted characters must be twice the length of the area being converted.

For types B and D, two 4/8 characters are converted to one EBCDIC character. Therefore, the length of the converted field is one-half the length of the area to be converted.

Also, for types E and F, the BCD or substitute blank (2-8 punches) translates as "2-4-8-0" in 4/8 code while a blank (no punches) translates as "R-0-X-N" in 4/8 code. The CDCNV macro, types E and F, may be modified to transmit or receive the BCD or substitute blank. (See Appendix L.)

For a discussion of the WAIT and WAITM macros used for processing with STR devices, see PROCESSING RECORDS WITH PHYSICAL IOCS, under WAIT Macro and WAITM Macro.

SCLOS Macro

Name	Op	Operand
	SCLOS	dtfname

Logical IOCS also provides a unique macro for completion of STR processing. The SCLOS (close STR adapter) macro turns off the adapter. It may be issued at any time in the problem program. SCLOS must be issued when all transmission to the line specified in dtfname is completed, or when the problem program wishes to SOPEN the line with different operands. The problem program should SCLOS all lines before end-of-job.

Binary Synchronous Communication

Logical IOCS provides macro support routines designed to supply the facilities for sending and receiving data. It uses an IBM 2701 Data Adapter Unit equipped with an IBM Synchronous Data Adapter--Type II, connected by leased or dial line to a remote IBM System/360, Model 30, 40, 50, 65, 67 (working in 65 mode), or 75. The remote CPU is equipped with an IBM 2701 Data Adapter Unit with an SDA II or an IBM 2703 Transmission Control Unit with Binary Synchronous features.

BSC Line Control

Transmission is initiated in a point-to-point, CPU-to-CPU communications environment when one CPU successfully sends the Inquiry (ENQ) signal to the other CPU. For a leased line, BSC macro support provides the CNTRL Prepare (PRP) macro instruction to receive the ENQ and the CNTRL ENQ macro instruction to send the ENQ. (READ ENQ--Type TQ--also receives the ENQ signal.)

The ETX (ETB is also valid) control character is expected to be the last character of a message and, as such, indicates normal completion to a READ macro. Normally STX is the valid start-character of a text message. The SOH character is also valid. For transparent text, DLE STX and DLE ETX (or DLE ETB) are valid text-framing characters.

BSC support routines maintain a system of alternating acknowledgments (ACK-0 and ACK-1), positive responses to alternate text messages, for protection against message duplication or loss. The WRITE macro includes, following the actual WRITE

channel command, a READ command to receive the alternating acknowledgment. READ ENQ (type TQ) issues a READ command to receive a control character (ENQ) from the remote CPU. All other READ macro types issue a WRITE command to send the appropriate alternating acknowledgment for the last message (or control character) received, before issuing a READ command for this message. If the incorrect alternating acknowledgment is received, BSC error recovery retransmits the previous text message or control character until the correct alternating acknowledgment is received or until the retry count (RCOUNT in the DTFBS macro instruction) is exhausted. If the retry count has been exceeded and the correct alternating acknowledgment has not been received, the Message Format Error and Wrong ACK bits are posted to the BSC CCB.

BSC error routines perform other line-control analysis functions. The CNTRL macros (except CNTRL EOT on a leased line and CNTRL DSC) expect certain responses and include the channel commands to receive these responses.

Macro	Response Expected
CNTRL PRP	ENQ
CNTRL EOT (dial line)	DLE EOT, EOT, or ENQ
CNTRL WABT	ENQ
CNTRL ENQ	ACK-0

Invalid or unexpected responses detected by BSC error routines are posted to the CCB (unexpected response and I/O error) after the retry count is exhausted.

Refer to Appendix M, Part 2 for CCB bits to be checked after completion of BSC operations. Refer to Appendix M, Part 1 for a table containing a more precise description of each control character. Refer to Appendix M, Part 3 for a sample program illustrating how the problem program can use BSC support macro instructions for basic line control.

When BSC support macro instructions are used to transmit and receive data, the data adapter (SDA II) must be defined by the file-definition macro DTFBS (Define The File for Binary Synchronous Communication). This macro (and the DTFRF macro) are described under File Definition Macros.

Before any processing can be done within the CPU-to-CPU environment, the data adapter must be initialized. Logical IOCS provides the unique imperative macro instruction, BOPEN (open BSC adapter) to initialize the adapter.

BOPEN Macro

Name	Operation	Operand
	BOPEN	DTFNAME=dtfname, INTRFC=A, B DIAL=YES, NO CODE=A B

The BOPEN macro establishes the mode and, on a leased line, turns on the adapter (SDA II).

BOPEN must be issued for a line before any IDIAL macro or any READ, WRITE, or CNTRL macro. It may be reissued with the same or different parameters following the BCLOS macro instruction.

The first keyword operand specifies the symbolic name given to DTFBS for the line.

The second keyword operand (INTRFC=) specifies which adapter interface (A or B) is to be used. If this operand is omitted, INTRFC=A is assumed.

The third keyword operand (DIAL=) specifies whether the line is dial (DIAL=YES) or leased (DIAL=NO). If the DIAL parameter is omitted, DIAL=NO is assumed.

The fourth keyword operand (CODE=) specifies the code that is EBCDIC (A or B). This parameter applies when the dual code feature of the SDA II is present. If the CODE parameter is omitted, CODE=A is assumed.

If DIAL=YES is specified, the BOPEN macro must be followed by an IDIAL macro instruction to establish the connection. If DIAL=NO is specified, or assumed, BOPEN should be followed by a CNTRL operation (PRP or ENQ) or may be followed by a READ ENQ (Type TQ).

At the completion of the BOPEN macro, the problem program should check for normal completion (as in Appendix M, Part 2) before continuing.

IDIAL Macro

The IDIAL macro instruction performs the initial line control functions necessary for dial lines. It dials a number, monitors a line for "ringing" or handles ID-verification. IDIAL also reads or writes one text record from or to the remote CPU.

IDIAL must be issued for a dial line immediately following the BOPEN macro instruction and preceding any READ, WRITE, or CNTRL operation. IDIAL should not be used for a leased line.

The area address and length fields in the expanded BSC CCB must be loaded with the starting address and length (including the text-framing characters) of the data to be read or written before issuing the IDIAL macro.

The IDIAL macro is coded with the keyword operands as shown in Figure 28.

Name	Operation	Operand	Comments
	IDIAL	DTFNAME=dtfname	Symbolic name given to the DTFBS for this line.
		DIAL= CALL# / ANS* / MAN	This CPU is calling, answering, or establishing the connection with the remote CPU manually.
		°ID= NONE / SNDID / RCVID / BOTH	ID-verification: no ID-verification, send ID characters, receive ID characters, or both send and receive ID characters.
		OPTYPE= RD / WT / WTX	Read a text record. / Write a text record. / Write a record of transparent text.
		REG=n	General register (2-11) to be loaded by the problem program with the address of the IDLST. This parameter may be omitted only if there is no IDLST (i.e., DIAL=ANS/MAN, ID=NONE).

\# If DIAL=CALL, the Automatic Call Feature must be installed. OPTYPE must be either WT or WTX.

* If DIAL=ANS, OPTYPE must be RD.

° If the ID parameter is omitted, ID=NONE is assumed.

Figure 28. IDIAL Macro

For OPTYPE=WT (normal WRITE), the problem program normally should provide the text-framing characters, STX and ETX. For OPTYPE=WTX (transparent WRITE), the problem program must supply the start-character sequence, DLE STX; macro support provides the end-character sequence, DLE ETX. As with any WRITE macro, no line control characters should appear in the text unless a transparent WRITE is used.

ID-verification procedures, if included, allow two CPU's connected by dial line to identify themselves by exchanging sequences of up to 15 hexadecimal graphic characters. The problem program must provide these character sequences in an IDLST. The correct control (or response) character must be provided following the last ID-character in each ID-sequence and must be added to the count. See Figure 29 and Appendix M, Part 1. The IDLST may also contain the digits to be dialed (for DIAL=CALL), up to 15 digits (expressed as characters or as hexadecimal digits). Refer to Figure 30 for the format of the IDLST and for examples.

IDIAL OPERANDS	CONTROL (OR RESPONSE) CHARACTER TO BE INCLUDED AFTER THE LAST ID-CHARACTER *°	
	SNDID CHARACTERS	RCVID CHARACTERS
DIAL=CALL/MAN OPTYPE=WT/WTX	ENQ	ACK-0
DIAL=ANS/MAN OPTYPE=RD	ACK-0	ENQ

* Refer to Appendix M, Part 1 for the hexadecimal representation of the character and the length in bytes to be added to the count.

° The problem programmer must provide both SNDID and RCVID control (or response) characters if he includes any ID-verification (if he codes ID=SNDID/RCVID/BOTH).

Figure 29. ID-Character Sequence, Control Characters

IDLST	NUMBER OF DIAL DIGITS	DIGITS TO BE DIALED
	COUNT *	ID CHARACTERS TO BE SENT (SNDID)
	COUNT *	ID CHARACTERS EXPECTED (RCVID)
	RCV AREA - ID CHARACTERS RECEIVED WILL BE READ INTO THIS AREA AND CHECKED **	

* COUNT includes the number of ID-characters (up to 15) plus the length in bytes of the control (or response) character.

** The length of the RCV AREA should be equal to the number of ID-characters expected plus the length in bytes of the control (or response) character expected.

Examples:

(1) DIAL=CALL,ID=NONE

IDLST | X'4' | C'1234' | no. of digits - dial digits

[No other parameters are required.]

(2) DIAL=CALL,ID=SNDID

IDLST			
	X'4'	X'01020304'	no. of digits - dial digits
	X'4'	C'RAL' X'2D'	count - SNDID - ENQ
	X'2'	X'1070'	count - ACK-0
			RCV AREA

(3) DIAL=CALL,ID=RCVID

IDLST			
	X'4'	C'1234'	no. of digits - dial digits
	X'1'	X'2D'	count - ENQ
	X'5'	C'BOS' X'1070'	count - RCVID - ACK-0
			RCV AREA

(4) DIAL=CALL,ID=BOTH

IDLST			
	X'4'	C'1234'	no. of digits - dial digits
	X'4'	C'RAL' X'2D'	count - SNDID - ENQ
	X'5'	C'BOS' X'1070'	count - RCVID - ACK-0
			RCV AREA

(5) DIAL=ANS/MAN,ID=SNDID,OPTYPE=RD

IDLST			
	X'5'	C'BOS' X'1070'	count - SNDID - ACK-0
	X'1'	X'2D'	count - ENQ
			RCV AREA

(6) DIAL=AND/MAN,ID=RCVID,OPTYPE=RD

IDLST			
	X'2'	X'1070'	count - ACK-0
	X'4'	C'RAL' X'2D'	count - RCVID - ENQ
			RCV AREA

(7) DIAL=ANS/MAN,ID=BOTH,OPTYPE=RD

IDLST			
	X'5'	C'BOS' X'1070'	count - SNDID - ACK-0
	X'4'	C'RAL' X'2D'	count - RCVID - ENQ
			RCV AREA

Notes:

- For DIAL=MAN,OPTYPE=WT/WTX the IDLST form is the same as for the DIAL=CALL examples, except that no dial digits are required.
- For DIAL=ANS/MAN,ID=NONE no IDLST is required. (The REG=n parameter on the IDIAL macro instruction is also not required.) The IDLST will be provided by macro support.

Figure 30. IDLST Format and Examples

ID-checking is performed by the CPU that receives an ID-sequence. For example, if ID=SNDID is coded, the remote CPU receives and checks the ID-characters and control (or response) characters. If the specified ID-sequence(s) and responses are valid, the first text record is read or written.

The WAIT macro or WAITM macro should be used to check for the completion of the IDIAL macro instruction.

Upon completion of the IDIAL macro instruction, the problem program should check the normal completion bit in the BSC flag bytes of the CCB. If normal completion is not indicated, the program should check further in the "completion" and "received" BSC flag bytes. (See Appendix M, Part 2.)

Once the proper connection has been established (with BOPEN and, on a dial line, IDIAL), logical IOCS provides

imperative macro instructions for the actual processing (sending or receiving) of data.

READ Macro

Name	Operation	Operand
	READ	dtfname,BSC,type-code

The READ macro instruction provides five READ types, each of which causes the reading of one record from the remote CPU.

The five types are:

- Continue (TN). One record, of the length specified in the length field of the expanded BSC CCB, is read into the data area pointed to in the CCB.
- Continue with leading graphics (TG). The graphic characters contained in the parameter list pointed to by the CCB are written to the remote CPU. (See Figure 31 for the form of the parameter list.) One record, of the length specified in the CCB, is then read into the data area pointed to in the parameter list.
- Repeat (TP). A NAK control character is sent to the remote CPU to request retransmission of the last record. The record, of the length specified in the CCB, is read into the data area pointed to by the CCB.
- Repeat with leading graphics (TL). The graphic characters contained in the parameter list pointed to by the CCB are written to the remote CPU. The NAK control character is then sent to the remote CPU to request retransmission of the last record. The record, of the length specified in the CCB, is then read into the data area pointed to by the parameter list. See Figure 31.
- Inquiry (TQ). This READ-type may be used to read the ENQ control character. The ENQ signal, when received, is read into the response area of the CCB.

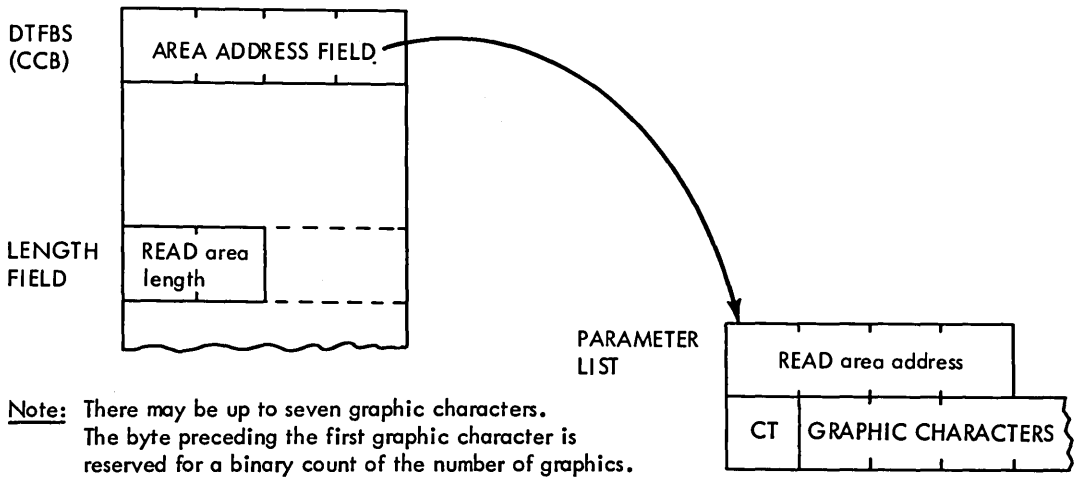
The third operand on the READ macro must contain one of the available type-codes:

$$\left\{\begin{matrix}\text{TN}\\ \text{TG}\\ \text{TP}\\ \text{TL}\\ \text{TQ}\end{matrix}\right\}$$

See the preceding description of each type.

Before issuing a READ macro, (except READ TQ), the area address field and the length field of the expanded BSC CCB must be properly loaded with the starting address and length (including the text-framing characters) of the data area (or the parameter list address and data area length the types TG or TL--READ's with leading graphics).

The line must be established (with BOPEN and, on a dial line, IDIAL) before the first READ macro instruction is issued. A CNTRL Prepare (PRP) should have been successfully completed before issuing the first READ on a leased line.



Note: There may be up to seven graphic characters. The byte preceding the first graphic character is reserved for a binary count of the number of graphics.

Figure 31. Parameter List for READ with Leading Graphics (Type-Code TG or TL)

The WAIT or WAITM macro should be used to check for the completion of the READ macro instruction.

Upon completion of the READ macro instruction, the problem program should check the normal completion bit in the BSC flag bytes of the CCB. If normal completion is not indicated, the program should check further in the "completion" and "received" BSC flag bytes. (See Appendix M, Part 2.) If normal completion is indicated, the problem program should also check the EOT received bit in the BSC flag bytes.

WRITE Macro

Name	Operation	Operand
	WRITE	dtfname,BSC,type-code

The WRITE macro provides five WRITE types, each of which causes the writing of one record to the remote CPU.

The five WRITE types are:

- Continue (TT). One record, of the length specified in the CCB, is written from the data area pointed to in the CCB.
- Transparent Text (TX) or Transparent Block (TXB). One record (or block) of the length specified in the CCB, is written from the data area pointed to in the CCB and the correct end-character sequence (DLE ETX or DLE ETB, respectively) is written.
- Conversational (TC). The problem program must set up a parameter list containing the length and starting address of the WRITE data area and the READ data area in that order. One record, of the length specified in this list, is written from the data area specified in the list. The response message (or control character) is then read into the specified READ data area. The parameter list is of the form indicated in Figure 32.
- Transparent Conversational (TV). One record, of the length specified in the parameter list pointed to in the CCB is written from the data area pointed to in the parameter list. The character sequence DLE ETX is written. The response message (or control character) is then read into the specified READ data area. Refer to Figure 32 for the correct form of the parameter list.

The third operand must contain one of the five available type-codes:

{ TT, TX, TXB, TC, TV }

See the preceding description of each type.

Before issuing a WRITE macro, the area address field and the length field of the expanded BSC CCB must be properly loaded with the starting address and length (including the text-framing characters) of the data area (or with the parameter list address and X'FFFF' for types TC or TV--conversational WRITE's). For

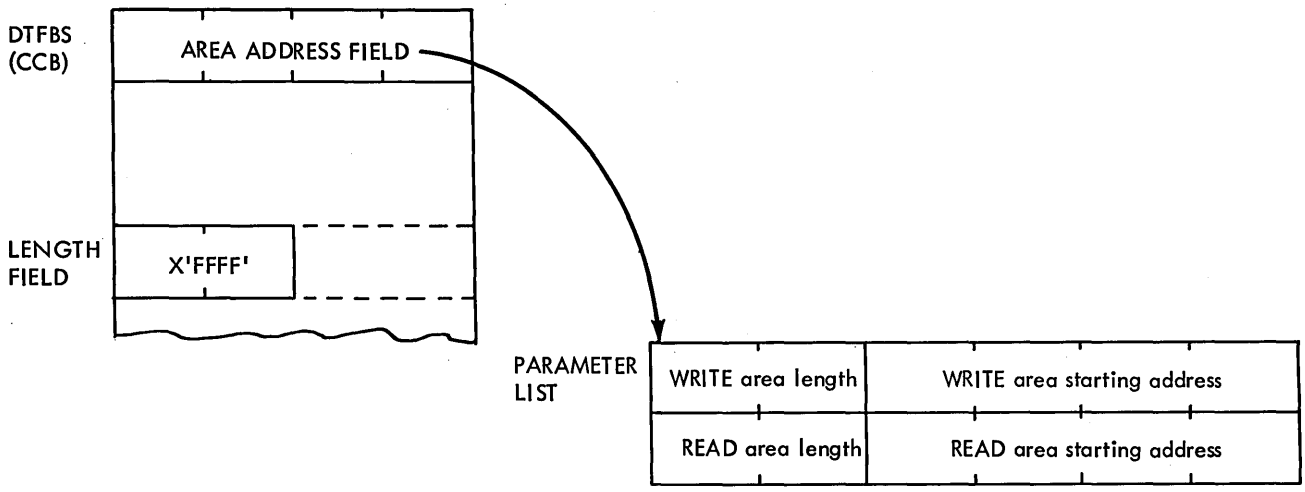


Figure 32. Parameter List for Conversational WRITE's (Type TC or TV)

WRITE-types TX, TXB, or TV, the DLE ETX (or DLE ETB) character sequence provided by macro support is not to be included in the length.

The line must be established (with BOPEN and, on a dial line, IDIAL) before the first WRITE macro instruction. A CNTRL ENQ should be issued before the first WRITE macro on a leased line.

The problem program normally should provide the text-framing characters, STX and ETX, for WRITE Continue (type TT) and WRITE Conversational (type TC). For transparent WRITE's (types TX, TXB, and TV), the problem program must supply the start-character sequence, DLE STX. Macro support provides the correct end-character sequence: DLE ETX for type TX or TV, DLE ETB for type TXB. No line control characters should appear in the text unless a transparent WRITE is used.

A conversational WRITE (type TC or TV) that reaches normal completion (a valid text message was received) should be followed by a READ Continue (type TN) to ensure that the system of alternating acknowledgments is correctly maintained. A conversational WRITE (type TC or TV) that receives graphics or some other response (not a valid text message) may be followed by another WRITE or CNTRL operation.

On a switched line, if an EOT is received in response to a text write, the transmitting CPU (which issues the WRITE macro) must issue a CNTRL EOT macro instruction to allow the receiving CPU to issue CNTRL ENQ and begin transmitting.

The WAIT or WAITM macro should be used to check for the completion of the WRITE macro instruction.

Upon completion of the WRITE macro instruction, the problem program should check the normal completion bit in the BSC flag bytes of the CCB. If normal completion is not indicated, the program should check further in the "completion" and "received" BSC flag bytes. (See Appendix M, Part 2.)

CNTRL Macro

Name	Operation	Operand
	CNTRL	dtfname,code

The CNTRL (control) macro instruction provides orders to the data adapter pertaining to line control.

CNTRL requires two operands. The first operand (dtfname) must be the symbolic name on the DTFBS macro. The second operand specifies one of the following mnemonic codes for the operation to be performed:

- PRP (Prepare) is used to monitor a non-switched line for activity. The operation completes when a signal of activity (normally an ENQ) is received from the remote CPU. CNTRL Prepare should be coded preceding the first READ macro and following BOPEN on a non-switched line. CNTRL Prepare should not be used for a dial line.
- EOT (End of Transmission) is used to send the End-Of-Transmission signal to the remote CPU. On a non-switched line, there is no provision for reading a response from the remote CPU. On a switched line, Message Format Error and the appropriate bit in the Received . Byte of the Expanded BSC CCB (EOT, DLE EOT, or ENQ) is posted.
- WABT (Wait Before Transmitting) sends the WAit-Before-Transmitting (WABT) sequence to the remote CPU and waits for a response, normally an ENQ. The problem program that requests the delay may reissue CNTRL WABT if not yet ready to receive when the ENQ is received.
- DSC (Disconnect) is used, on a switched line, to indicate to the remote CPU that the connection is being broken (the line is being disabled) at this CPU. CNTRL Disconnect must be followed by BCLOS to disable the line. CNTRL Disconnect should not be used on a leased line.
- ENQ (Inquiry) is used to bid for the line and initiate transmission by sending the ENQ control character. CNTRL ENQ normally precedes a WRITE macro. The expected response is ACK-9. When CNTRL ENQ elicits an outstanding acknowledgement from the remote CPU (i. e., when a WABT has been received), the response may be ACK-0, ACK-1, or NAK.

The line must be established (with BOPEN and, on a dial line, IDIAL) before issuing the CNTRL macro.

The problem program must be sure that a CNTRL macro is complete by using the WAIT or WAITM macro instruction immediately following the CNTRL macro instruction.

Upon completion of the CNTRL macro instruction, the problem program should check the normal completion bit in the BSC flag bytes of the CCB. If normal completion is not indicated, the problem program should check further in the

"completion" and "received" BSC flag bytes. (See Appendix M, Part 2).

BCLOS Macro

Logical IOCS provides a unique macro instruction for completion of BSC processing. The BCLOS macro turns off the data adapter (SDA II) and clears the BSC flag bytes. Before disabling the adapter, BCLOS halts I/O on any command outstanding to the SDA II. BCLOS must be issued when all transmission on the line specified in dtfname is completed or when the problem program wishes to use BOPEN (and IDIAL) again with different parameters.

Name	Operand	Operand
	BCLOS	dtfname

The problem program should check for the completion of BCLOS by using the WAIT or WAITM macro.

ERRPT Macro

When all data transmission is completed and end-of-job is reached, the problem program using BSC support macros must issue the ERRPT macro. ERRPT displays the error statistics:

- Data Check
- Lost Data
- Intervention Required
- Time Out
- Unit Check

and the count of total transmissions received for this job. These hexadecimal counts illustrate errors on the line or in the modern equipment. Regular display of the counts ensures maximum throughput for the problem program through early detection of frequently occurring errors.

ERRPT also performs functions essential to the proper termination of a BSC job. ERRPT removes the CCB from the CCB table (BTAB) kept in the supervisor. The CCB is entered in the table when a BOPEN macro is issued for the CCB, and is only deleted by ERRPT. Therefore, to avoid filling up the CCB table, ERRPT should be issued following the BCLOS macro for this CCB.

Name	Operation	Operand
	ERRPT	dtfname

The ERRPT macro should follow the BCLOS macro, and, at end-of-job, precede the EOJ macro.

For more detailed information on the format of these messages, refer to IBM System/360 Basic Operating System, Operator Messages, Form C24-5024.

For a discussion of the WAIT and WAITM macros used for processing with BSC support, see Processing Records with Physical IOCS, under WAIT Macro and WAITM Macro.

PROCESSING RECORDS WITH PHYSICAL IOCS

Records can be transferred to or from an input/output device by issuing physical IOCS macro instructions. These instructions relate directly to the physical IOCS routines and bypass all logical IOCS routines. Thus routines for such functions as blocking or deblocking records, performing programmed wrong-length-record checks, switching I/O areas when two areas are used, and setting up Channel Command Words (CCW) are eliminated. Any of these functions that are required for a problem program must be provided by the user in his own programming.

Physical IOCS routines control the transfer of data to or from the external device. These routines are:

Start I/O

Interruption

Channel Scheduler

Device Error

Thus, physical IOCS macro instructions provide the user with the capability of obtaining data and performing non-data operations in I/O devices, by issuing only the I/O commands that he requests. For example, if he is handling only physical records, he does not need the IOCS routines for blocking and deblocking logical records. He can write his own routines to handle the characteristics of his data file logically.

Three macro instructions are available to the programmer for direct communication with physical IOCS: CCB (Command Control Block), EXCP (Execute Channel Program), and WAIT. These are explained in the following sections. Whenever physical IOCS macro instructions are used, the programmer must construct the Channel Command Words (CCW) for his input/output operations. He uses the assembler-instruction CCW statement for this. However, when using physical IOCS for 7-track tapes, the user need not write CCW's for setting the mode of the tape. Physical IOCS automatically performs this

function. He must also recognize and bypass checkpoint records if they are interspersed with data records on an input tape.

CCB Macro

Name	Op	Operand
blockname	CCB	SYSnnn,command-list-name,X'yyyy'

The CCB (Command Control Block) macro instruction must be issued once in the problem program for each I/O device that is controlled by physical IOCS macro instructions. It causes a command control block (Figure 33) to be created. This block is necessary to communicate information to physical IOCS so that it can perform desired operations (for example, start I/O). The command control block also receives status information after an operation, and makes this available for use by the problem program.

The CCB instruction must be labeled (Blockname) with a symbolic name. This name must be the operand in the EXCP and WAIT instructions, which must refer to the command control block.

Two operands are required in this CCB instruction. A third operand is optional. The first operand specifies the symbolic unit (SYSnnn) for the actual I/O unit with which this control block will be associated. The name may be SYSRDR, SYSLST, SYSIPT, SYSOPT, SYSLOG, SYS000-SYS254. The actual I/O unit is assigned to the symbolic unit by a Job Control ASSGN card, or by the SYMUN macro instruction.

The second operand (command-list-name) specifies the symbolic name of the first CCW to be used with this CCB. This name must be the same as the name specified in the assembler CCW statement that constructs the channel command word.

The third operand (X'yyyy') may be used to set the bits of bytes 2 and 3 at assembly time. After the user determines which bits he wishes to set on, and which off, he enters the hexadecimal representation of the binary value that he wishes. The hexadecimal value must be preceded by X and enclosed in single quotes. For example, to set on byte 2, bit 6, he would enter X'0400'.

Only the last five bits of byte 2 are used by the problem program to communicate with physical IOCS, as shown in Figure 33. However, the user may also wish to set (at assembly time) some bits that IOCS normally sets during program execution. For example, if the user sets bit 6 of byte 3 on at assembly time, he can cause the program to act as if the channel 9 overflow condition has occurred when he begins executing his program.

From the specifications in this CCB instruction, the macro sets up an 8-byte command-control block (Figure 33) as follows:

Bytes	Contents
0-1	The first two bytes are used for a chain field that physical IOCS uses for channel queueing. After a record has been transferred, IOCS places the residual count in these two bytes. The problem program can use this to check the length of the record that was transferred.
2-3	The next two bytes are used for transmission of information between physical IOCS and the problem program. (For example, the problem program can test byte 2, bit 1 to determine if the I/O device detected a wrong-length record when data was transferred.) All bits are set to a 0 (off) when the problem program is assembled unless the third operand is included in the CCB macro instruction. If the third operand is included, all bits set by IOCS should be assembled as zeros (off). During execution, each bit may be set at 1 (on) by the problem program or by a condition detected by physical IOCS. In byte 2, bits 3 and 5-7 are turned on by the problem program. Any bits that are turned on, during program execution, by physical IOCS are reset by IOCS the next time an EXCP macro using the same CCB is executed. The condition indicated

by the setting of each bit is shown in Figure 34.

4-5 Bit 0 of byte 4 is used to indicate that a program-controlled interruption has occurred. Like the bits in bytes 2 and 3 (see Figures 33 and 34), this is used to transfer information from physical IOCS to the problem program. The other bits of bytes 4 and 5 are a hexadecimal representation of the symbolic unit for the I/O device, as specified in the first operand of this CCB instruction.

6-7 The last two bytes contain the address of the CCW (or first address of a chain of CCW's) associated with this CCB and specified symbolically in the second operand.

For a description of the CCB, expanded for STR use, see: File Definition Macros, Processing with STR Devices (DTFSN, DTFRF). For description of the expanded BSC CCB, see: File Definition Macros, Binary Synchronous Communication (DTFBS, DTFRF).

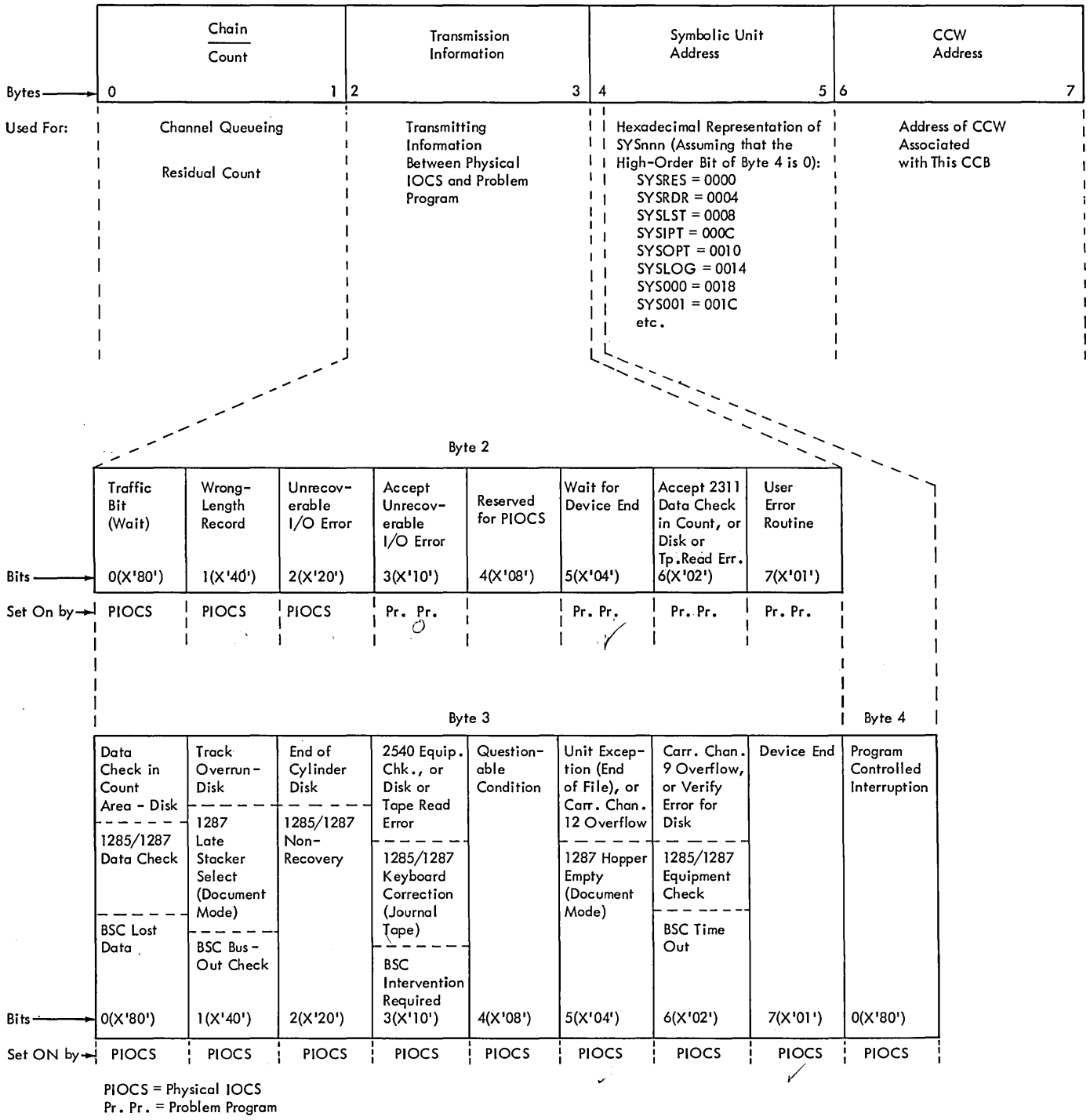


Figure 33. Command Control Block (CCB)

BYTE	BIT	CONDITION INDICATED	
		1 (ON)	0 (OFF)
2	0 - Traffic Bit (Wait)	I/O Completed (Channel End)	I/O Requested and Not Completed
	1 - Wrong-Length Record	Bit 41 in CSW is ON	- -
	2 - Unrecoverable I/O Error	I/O Not Executed	- -
	3 - Accept Unrecoverable I/O (Bit 2 ON)	Return to User after Physical IOCS Attempts to Correct I/O Error	Terminate Job when Unrecoverable Error Detected
	4 - Reserved for PIOCS	- -	- -
	5 - Wait for Device End	Printer to be Tested for Carriage Channel 9 or 12 overflow, or Punch to be tested for errors, or the user has issued a CCB requesting physical IOCS to post byte 3, bit 7 at Device End.	- -
	6 - Accept Data Check in Count, or Disk or Tape Read Error. (Bit 3 of Byte 3 ON)	Return to User After Physical IOCS Attempts to Correct the Disk or Tape Error.	Terminates the Job if the Read Cannot be Accomplished
	7 - User Error Routine	User will Handle Unit Check (Test Bit 2)	Physical IOCS Error Routine
3	0 - Data Check in Count Area - Disk	Yes	No
	- Data Check: 1285 or 1287	Yes	No
	- BSC Lost Data	Yes	No
	1 - Track Overrun - Disk	Yes	No
	- Late Stacker Select 1287 Document Mode	Yes	No
	- BSC Bus-out Check	Yes	No
	2 - End of Cylinder - Disk	Yes	No
	- 1285/1287 Non-Recovery Document Jam or Torn Tape	Yes	No
	3 - 2540 Equipment Check, or Tape or Disk Read Error	Yes	No
	- 1285/1287 Keyboard Correction Journal Tape Mode	Yes	No
	- BSC Intervention Required	Yes	No
	4 - Questionable Condition	Card: Unusual Command Sequence Tape: Converter Check Disk: No Record Found	- -
	5 - Carriage Channel 12 Overflow *#, or Unit Exception (End of File)	Yes	No
	- Hopper Empty 1287 Document Mode	Yes	No
	6 - Carriage Channel 9 Overflow *#, or Verify Error for Disk	Yes	No
	- 1285/1287 Equipment Check	Yes	No
	- BSC Time Out	Yes	No
→	7 - Device End	Has Occurred*	- -
4	0 - Program-Controlled Interruption	PCI Bit in PSW is ON	- -

* Set ON Only if Byte 2, Bit 5 is ON
\# DVE needed in IOCFG Macro

Figure 34. Conditions Indicated by CCB Bytes 2, 3, and 4

EXCP Macro

Name	Op	Operand
	EXCP	blockname

The EXCP (execute channel program) macro instruction requests physical IOCS to start an input/output operation for a particular I/O device. The symbolic name (blockname) of the CCB established for the device is the only operand required in this instruction.

Physical IOCS determines the device concerned, from the command control block specified by blockname, and either starts the device or places the command control block (CCB) in a channel queue. Program control is then returned to the problem program. If the CCB is in a queue, the actual transfer of data will be started at some later time, when the CCB reaches the top of the queue.

WAIT Macro

Name	Op	Operand
	WAIT	blockname

This WAIT or WAITM macro instruction is issued whenever the program requires that an I/O operation, started by an EXCP instruction, be completed before execution of the problem program continues. For example, the transfer of data (a physical record) to main storage must be completed before that data can be added, moved to another area of main storage , or otherwise processed. When this WAIT instruction is executed, the program enters a waiting loop until the related CCB indicates that the associated input/output operation is finished. Then programming automatically continues, and the data can be processed. The WAIT (or WAITM) macro should be issued to check for the completion of BSC support macros.

The symbolic name (blockname) of the CCB established for the I/O device is the only operand required in this instruction. This is also the same name as that specified in the EXCP instruction for this device. When using STR macro instructions, the symbolic name (blockname) of the CCB is referred to as "dtfnameB", where dtfname is the symbolic name of the DTFSN macro. When using BSC macro instructions, the symbolic name (blockname) of the CCB is referred to as "dtfnameD," where dtfname is the symbolic name of the DTFBS macro.

WAITM Macro

Name	Op	Operand
	WAITM	blockname1, blockname2....
		blocknamen, reg

This macro is used with both physical IOCS and logical IOCS for STR devices or for BSC support. The WAITM macro instruction allows the problem program to wait on the completion of one of several specified I/O operations. When the WAITM macro is executed, the problem program enters a waiting loop until the traffic bit in any one of the specified CCB's indicates that the associated I/O operation is completed.

If the problem program is using physical IOCS, "blockname" is the symbolic name of the CCB. For STR devices an expanded CCB is generated by the DTFSN macro. The blockname for a DTFSN CCB must be referenced in the operand field of a WAITM (or WAIT) macro as "dtfnameB" (where dtfname is the symbolic name of the DTFSN). The maximum number of blocknames that can be specified is 17. For BSC support, the expanded CCB is generated by the DTFBS macro. The blockname for a BSC CCB must be referenced in the operand field of a WAITM (or WAIT) macro as "dtfnameD" (where dtfname is the symbolic name of the DTFBS macro).

The last operand (reg) is a general register number (2-11). The address of the associated CCB (dtfnameB) is loaded into this register upon completion of an I/O operation.

The WAITM macro must be issued to check for completion of a SOPEN, READ, WRITE, or CNTRL macro, when the problem program uses these macros with respect to STR I/O operations. WAITM (or WAIT) should also be issued to check for the completion of BSC support macros.

CHNG Macro

Name	Op	Operand
	CHNG	SYSnnn

This macro instruction is used with both logical and physical IOCS. It is described in the group included under Processing Records Consecutively.

WRITING CHECKPOINT RECORDS

When a problem program is expected to run for an extended period of time, provision should be made for taking checkpoint records periodically during the run. These records contain the status of the job and the system at the time records are written. Thus, they provide a means of restarting at some midway point rather than at the beginning of the entire job, if processing must be terminated for any reason before the normal end of job. For example, a job of higher priority may require immediate processing, or some malfunction such as a power failure may occur, and cause such an interruption.

If checkpoint records are written periodically, operation can be restarted by using the last set of checkpoint records written previous to the interruption. Therefore the records must contain everything needed to re-initialize the system when processing is restarted. Checkpoint records can be written by issuing a CHKPT macro instruction in the problem program. Restarting jobs, for which checkpoint records have been written by use of the CHKPT macro, is performed by the Job Control program. Job Control is described in the Programmer's Guides, as listed on the front cover of this publication.

Each time CHKPT is executed, several records are written on disk or tape. If the object program is executed in a disk-resident system, the checkpoint records may be written in the checkpoint area of the system disk pack or on tape. When they are written on disk, each set of checkpoint records replaces the previous set so that the most recent is always available for restarting the job. When checkpoint records are written on tape, an additional set is written each time CHKPT is executed. Thus all the sets of checkpoint records for a particular run are saved. Each set should be identified so that the set to be used for restarting can be identified. If a disk-resident system is not used for execution of the object program, the checkpoint records must be written on tape. The disk/tape checkpoint records contain information such as:

- Header information: ///CHKPT//nnnnxxxx where nnnn is information used by the restart program, and xxxx is the user's identification (usually a number) of the set of checkpoint records. Before each set is written, this identification should be changed by the user. On the restart, this identifies the set of checkpoint records to be used. This record is not written when checkpoint records are written on the system pack.
- The contents of the general registers are saved. The contents of the floating-point registers are not saved.
- The contents of the Supervisor communication region, except the first 10 bytes.
- A table of restart information, such as the starting point in the problem program and the positions of the input/output data tapes within the logical file.
- The problem program and data in process at this time. This includes all main storage above the supervisor, depending on the size specified in the configuration byte (9) of the communication region.
- Trailer information, when checkpoint records are written on tape. The trailer is identical to the header label.

CHKPT Macro

Op	Operand
CHKPT	n,restart-name,SYSnnn,DISK

The CHKPT (checkpoint) macro instruction causes checkpoint records to be written. The instruction requires either three or four operands, depending on whether the object program is to be executed in a disk-resident system.

The first operand specifies the number (n) of tape reels of input/output data that will have to be properly positioned when the job is restarted. The output tape used for writing the checkpoint records must not be included in this count. For all other tape drives, the macro routines save both the address of the symbolic unit for the drive and the block count that has been accumulated at the time the checkpoint records are written. Block count is the number of blocks of data (physical tape records, from gap to gap) that have been read or written. When the job is restarted, the block count is used to position the tape properly to continue with the data yet to be processed. If the object program does not include the processing of any input/output data that is recorded on tape, the comma must be entered first in the operand field of the CHKPT macro instruction.

The second operand (Restart-name) specifies the symbolic name of the problem-program statement at which programming is to restart if an interruption occurs and processing must be continued at some later time.

The third operand specifies the symbolic unit (SYSnnn) for the tape drive on which the checkpoint records are to be written, if they are to be recorded on tape rather than disk. When an interrupted job is restarted, this same symbolic unit must be specified in the Job Control RSTRT card. If the checkpoint records are to be written on disk, SYSRES must be specified.

When checkpoint records are written on tape, they may be written on a separate tape drive, or they may be interspersed with data records in an output tape file. When the symbolic unit (SYSnnn) is assigned to the same drive used as an output file defined by logical IOCS, checkpoint records are written on this drive regardless of whether an alternate drive is assigned by the DTFSR entry ALTTAPE. The CHKPT macro will not write checkpoint records on the alternate drive. If checkpoint records are interspersed with data records in an output tape file, they can be bypassed on input by use of the DTFSR entry CKPTREC.

When checkpoint records are written on 7-track tape, the Data Conversion special feature must be used.

The fourth operand (DISK) indicates that checkpoint records are to be taken for an object program executed in a disk-resident system. Otherwise, it is omitted and the macro instruction contains only three operands.

Issuing a CHKPT macro instruction in the problem program causes the macro-library checkpoint routines to be assembled at the same time as the problem program. If the user's object program is to be executed in a disk-resident system (DISK specified as the fourth parameter in the CHKPT macro instruction), the checkpoint routines must be stored in the core-image library on the system-residence disk pack. They are transferred from system residence to the Supervisor transient area in main storage whenever the CHKPT macro is to be executed. If a disk-resident system is not to be used for execution of the object program (DISK not specified in this instruction), the checkpoint routines are assembled with the user's problem program. In either case (disk-resident system, or not) the CHKPT instruction, if used, must be issued only once so that the routines are assembled only once per program assembly.

The checkpoint routines use registers 1, 14, and 15. It is the user's responsibility to save the contents of these registers if the problem program also uses them.

When the object program is to be executed in a disk-resident system, issuing the CHKPT macro instruction in the source program causes a supervisor-call instruction with code 1, and a statement with the program name SYSCPT or SYSCPD, to be assembled (see Supervisor-Communication Macros: FETCH). During execution, the supervisor-call interruption routine analyzes the code and fetches the checkpoint routines from residence to the transient area. When a disk-resident system is not used for program execution, checkpoint routines are assembled in line.

Included with the assembled checkpoint routines (for a disk-resident system, or not) is a table of restart information. This table may contain such information as the checkpoint identification, the restart address, and information about the tape drives used by the problem program. For example, issuing a CHKPT instruction such as:

```
        CHKPT   3,STHERE,SYS004
```

causes this table to be assembled:

Statements			Notes
SYSCHKPT	DC	CL4' ' (4blanks)	1
	DC	Y(STHERE)	2
	DC	3FL4'0'	3

Notes:

1. This statement provides four bytes for the checkpoint identification. It is the user's responsibility to move updated identification to these four positions before each set of checkpoint records is written. For example, he might move CR01 (Checkpoint Record 1) to SYSCHKPT for the first set, CR02 for the second, etc. Each time checkpoint records are written, this is part of the recorded table of restart information. Then when the job is restarted after an interruption, the programmer indicates which set of checkpoint records is to be used by specifying the identification in the Job Control RSTRT card. The tape containing the checkpoint records is searched, and the identified set of records is read from the tape.

2. The symbolic restart-name, STHERE (start here) in this case, is assembled in this macro-library statement. If the user wants to restart at a different location than that specified

in his macro instruction, he can move the address of that location to SYSCHKPT+4

3. The DC statement defines a full word (four bytes) constant for each input/output data file. In this case, three constants are assembled because 3is specified as the first operand in the CHKPT instruction. These constants provide an area to store information for properly positioning the input/output data tapes when the job is restarted after an interruption. The first two bytes of each statement contain the address of the symbolic unit for the corresponding tape drive. The second two bytes contain the block count for the corresponding file. Usually when logical IOCS is used for a file of records, the user does not have to refer to these fields. The IOCS routines move the symbolic unit address before the checkpoint records are written. They also increment the block count (if DTFSR CHECKPT specified) whenever a GET or PUT requires an actual I/O operation. If the user (instead of IOCS) is supplying information for certain files, he must not use those 4-byte fields that are specified for files maintained by logical IOCS (see DTFSR CHECKPT).

The user must supply both the address and the block count whenever:

- a file with nonstandard labels is read backwards.
- an unlabeled file is read backwards.
- a file is processed by physical IOCS macro instructions.

The block count that he supplies must be relative to the beginning of the file, regardless of whether the file is read forward or backwards.

To supply the address information, the user can move the addresses of the symbolic units from the Command Control Blocks (CCB) associated with the files (see Processing Records with Physical IOCS: CCB). He would have issued three CCB's, one for each file, perhaps as follows:

```
BLOCK1 CCB  SYS001,CCW1
BLOCK2 CCB  SYS002,CCW2
BLOCK3 CCB  SYS003,CCW3
```

To move the addresses in this case, he could write these statements:

```
MVC    SYSCHKPT+6(2),BLOCK1+4
MVC    SYSCHKPT+10(2),BLOCK2+4
MVC    SYSCHKPT+14(2),BLOCK3+4
```

For the block count, the user must increment (or decrement, on a backspace order) the count each time an EXCP macro that results in tape movement is executed. The instructions to add to (or subtract from) the block count must precede the EXCP instruction in the user's program. For the first file, they might be:

```
LH     R,SYSCHKPT+8
LA     R,1(R)
STH    R,SYSCHKPT+8
EXCP   BLOCK1
```

A block count must not be supplied in the restart table for the output file that contains both data and checkpoint records, however. In this case the tape is positioned, on restart, at the first data record following the set of checkpoint records that are used for restarting.

COMPLETION

After all the records for a logical output file have been processed (end-of-file), that file must be deactivated by an instruction in the problem program to close the file. When the end of a logical input file in an I/O unit other than disk or tape is sensed, IOCS immediately branches to the user's end-of-file routine (specified by DTFSR EOFADDR) where the instruction to close the file can be issued. When the end of a disk or tape input file is sensed, IOCS checks standard trailer labels (if any), makes provision for user-checking of user labels, and then branches to the user's end-of-file routine (specified by EOFADDR) where the file may be closed. A CLOSE macro instruction is available to the programmer for closing each input and output file.

An end-of-volume condition (EOV), rather than an end-of-file condition (EOF), can occur during the processing of records in a logical file on disk or tape. An EOV condition means that the processing of all the records on one volume (disk pack or tape reel) has been completed, but that more records for the same logical file are recorded on another volume. When this occurs, IOCS checks or writes standard labels (if any) on the completed volume (trailer labels) and on the next volume (header labels), makes provision for user-processing of user-standard labels on

both volumes, and then makes the data records on the next volume available for processing. Because IOCS detects the end-of-volume condition and utilizes many of the routines established for opening and closing files, no problem-program instructions are required specifically for an EOV condition. However, if the program requires that the processing of tape records on one volume be ended before the actual end of the volume is reached, an end-of-volume condition can be forced. An FEOV (forced end-of-volume) macro instruction is provided for this condition in tape files.

When an EOV or FEOV condition is detected in a logical file on tape, IOCS increases the volume sequence number (in storage) by 1. Thus, this number is updated for checking/writing the header label on the next reel. (For a description of tape labels, see the Programmer's Guide.) IOCS also updates the active drive number if an alternate tape drive has been specified (see DTFSR ALTTAPE).

The specific functions that occur on an EOF or EOV condition for a disk or tape file vary with the type of operation (input or output) and with the use of file labels. These functions are discussed in the following sections.

Disk Input File

When records in a logical input file on disk are processed in consecutive order (specified by DTFSR) or in sequential order by key (specified by DTFIS), IOCS detects an end-of-file condition. The end of the input file is determined either by the ending address of the last extent specified for the file in Job Control XTENT cards, or by an end-of-file record read from the data file. With sequential processing (DTFIS), IOCS immediately branches to the user's end-of-file routine (specified by EOFADDR).

When records are processed in consecutive order (DTFSR), the file may contain user trailer labels. In this case IOCS branches first to the user's label routine (specified by LABADDR) where the user may check his trailer labels. Up to eight trailer labels can be read and checked. They are written on the first track of the first extent specified for the file on each pack. The trailer labels follow the user header labels for the pack, and they are identified by UTL0, UTL1, - - - UTL7. When IOCS branches to the user's label routine, it also reads the trailer label and makes it available to the user for checking. IOCS sets up a label area and supplies the address of the area to the user in Register 1. After each label is checked, the user returns to IOCS by use of the LBRET macro. After all trailer labels have been checked, IOCS branches to the user's end-of-file routine (specified by EOFADDR).

IOCS detects end-of-volume conditions in a disk input file. The end of a volume is recognized when all extents on one volume have been processed but Job Control XTENT cards have specified additional extents on another pack. At the end of a volume, IOCS allows the user to check his trailer labels (if any), the same as at the end of a file. IOCS then checks the standard header labels on the next volume, allows the user to check any user header labels by branching to the address specified by LABADDR, and makes the first record in the first extent available for processing.

Disk Output File

When disk records are processed consecutively (DTFSR) or loaded sequentially by key (DTFIS), and when all records for the logical file have been completed, the CLOSE instruction is issued and normal EOF procedures are initiated (see CLOSE Macro). If the end of the last extent specified for the file is reached before CLOSE is issued, IOCS assumes an error condition.

End-of-volume conditions in a disk output file are detected in the same way as in a disk input file. At the end of a volume IOCS allows the user to write his trailer labels (if any), the same as at the end of a file (see CLOSE Macro). IOCS then writes standard header labels on the next volume, allows the user to write any user header labels by branching to the user's label routine (specified by LABADDR), and permits the processing of output data records to continue.

Tape Input File

When logical IOCS senses a tape mark on a tape input file, either an end-of-file or end-of-volume condition exists. The EOF/EOV condition is determined by IOCS or by the user (depending on the type of labels used for the file) and the appropriate functions are performed.

If standard labels are specified, IOCS immediately reads and checks the standard trailer label. If user labels are also present and are to be checked (specified by DTFSR LABADDR), the user's routine is then entered for each user label that is read (see OPEN Macro). After all labels have been checked, the rewind option is executed, as specified in DTFSR REWIND.

When the standard trailer label is checked, either an EOV or EOF condition is

sensed. When an EOV identifier is sensed, IOCS switches to the alternate tape drive (if one is specified in the DTFSR entry ALTTAPE) after user labels have been checked if specified. If an alternate drive is not specified, the operator is notified to change the tape reels and the system enters the wait state. When the operator has mounted the new reel and pressed either the request key (on the 1052) or the interruption key (on the console), processing resumes. IOCS checks the header label(s) if checking is specified, and normal processing continues. If an input file is processed by physical IOCS (DTFPH specified), the user must issue an OPEN instruction for the new reel. Then IOCS checks the header label and processing continues.

When an EOF condition is sensed, IOCS branches to the programmer's end-of-file routine, specified by the DTFSR entry EOFADDR.

If the tape input file has nonstandard labels, IOCS immediately branches to the user's label routine (specified by DTFSR LABADDR) when the tape mark is sensed. In his routine, the programmer must use physical IOCS macro instructions to read his label(s). Furthermore he must determine the EOF/EOV condition and indicate this to IOCS by loading either EF (end-of-file) or EV (end-of-volume) in the two low-order bytes of Register 0. On an EF condition, IOCS branches to the users end-of-file address (specified by DTFSR EOFADDR) when the problem program returns to IOCS at the end of the label routine. On an EV condition, IOCS initiates the end-of-volume procedures to close the completed volume and open the next volume for processing.

If a tape file is not labeled (DTFSR FILABL=NO) or contains labels that are not to be checked (DTFSR FILABL=NSTD), IOCS branches to the end-of-file address when the tape mark following the last data record is sensed. If an end-of-volume condition exists instead of an end-of-file condition, the user may indicate this by issuing an FEOV macro instruction in his end-of-file routine. If an alternate tape drive is specified by the DTFSR entry ALTTAPE, IOCS switches to the alternate drive and processing resumes. If an alternate tape drive is not specified, the operator is notified to change the tape reels, and the system enters the wait state. When the new tape reel has been mounted, the operator must press either the request key on the 1052 or the interruption key on the console to resume processing.

Whenever an input tape is read backwards (BACK specified in DTFSR READ), an end-of-file condition always exists when the file header label is reached. That is, backwards reading is confined to one volume. Therefore, with standard labels, the input/output routines check only the block count, which was stored from the trailer label, and then branch to the specified end-of-file routine. When physical IOCS macros are used to read records backwards, labels cannot be checked (DTFPH must not be specified). For tape files with nonstandard labels, IOCS branches to the user's label routine specified by DTFSR LABADDR where he may check the header label. He must use physical IOCS macro instructions to read the label(s) for checking.

Tape Output File

When an end-of-reel reflective marker is sensed on an output tape, logical IOCS prepares for closing the file by ensuring that all records have been written on the tape. If the programmer issues another PUT, indicating that more records are to be written on this output file, normal end-of-volume (EOV) procedures are initiated. If the programmer issues a CLOSE, the EOF procedures are initiated.

The programmer should be aware that, under certain conditions, a truncated block of records may be written at an EOV or EOF condition, even though the file is defined as having fixed-length blocked records. When this file is used for input, the System/360 logical IOCS will recognize and handle these short blocks without the programmer being concerned or aware of this condition.

Labeling procedures for the EOV condition closely follow those described under CLOSE Macro. The label is coded EOV rather than EOF, and only one tape mark is written after the label set, or after the data if standard labels are not used.

Forced End-of-Volume: Tape Files

In some cases a programmer may need to force an end-of-volume condition at a point other than the normal tape mark (input) or reflective marker (output). He may want to discontinue reading or writing the records on the present volume, and continue with those records for this same logical file that are recorded on the next volume. This may be necessary because of some major change in the category of records or in the processing requirements. An FEOV (forced end-of-volume) macro instruction is available to the programmer for this function.

CLOSE Macro

Op	Operand
CLOSE	filename
CLOSE	filename1,filename2,filename3,---

The CLOSE instruction is used to deactivate any file that was previously opened in any input/output unit in the system: card reader, card punch, magnetic tape unit, disk drive, paper tape reader, printer, optical roll reader, or display unit. The symbolic name of the logical file, assigned in the DTFSR, DTFDA, DTFIS, or DTFPH header entry, is required in this instruction. As many as 16 files may be closed by one instruction, by entering additional filename parameters. CLOSE is required whenever logical IOCS macro instructions are used to transfer data. When physical IOCS is used, CLOSE is required only if standard labels on magnetic tape are to be written. A file may be closed at any time by issuing this macro instruction.

Reopening a Closed File: If further processing of a closed file is required at some later time in the program, the file must be opened again. If a file of disk records is reopened after a CLOSE, the label processing and extents made available depend on the type of processing that is specified for the file. When an input file is processed in consecutive order (DTFSR specified), IOCS checks the label(s) on the first pack and makes the first extent available, the same as at the original OPEN. When a file is processed by physical IOCS with SINGLE specified in DTFPH MOUNTD, IOCS opens the next extent specified by the user's XTENT cards. When a file is processed by the direct access method (DTFDA specified), by the indexed sequential system (DTFIS specified), or by physical IOCS with ALL specified in DTFPH MOUNTD, all label processing is repeated and all extents are again made available.

If a file of tape records is closed, the tape is positioned in accordance with the REWIND specification. Therefore to resume processing of tape records at the point where CLOSE occurred, NORWD should be specified in DTFSR REWIND. When OPEN is issued later for additional records on that reel, the first record read must be a file label if standard labels are specified for the tape file being opened. If the tape file being opened is unlabeled or contains nonstandard labels, it is the user's responsibility to identify the first record read as a data record or a file label. When a file being reopened is a multireel file with standard labels, IOCS expects that the reel available for the OPEN is the same reel, on the same drive, that was in process when CLOSE was executed. If not, a message is issued.

In addition to the registers used by logical IOCS, CLOSE also uses register 5. The programmer may use register 5 because the CLOSE macro routine saves and restores this register. However, if the programmer plans to use register 5 as a base register, he should be aware that register 5 is dropped at the end of the CLOSE routine.

Disk Files

When disk records are processed in random order (specified by DTFDA or DTFIS), the CLOSE instruction is issued in the problem program to deactivate the file after all records have been processed.

When records in a disk input file are processed in consecutive order (DTFSR) or in sequential order (DTFIS), the CLOSE instruction is generally issued in the user's end-of-file routine (specified by EOFADDR) to deactivate the file. IOCS branches to this routine when it detects an end-of-file condition (see Completion).

When records in a disk output are processed in consecutive order (DTFSR) or loaded in sequential order (DTFIS), the CLOSE instruction is issued after all records for the file have been processed. It writes any record, or block of records, that has not already been written. If a record block is partially filled, it is truncated; that is, a short block is written on disk. CLOSE causes one or more functions to be performed before it deactivates the file. It always writes an end-of-file trailer record after the last data record in the file. If records are processed in consecutive order, user trailer labels may be written if the DTFSR entry LABADDR is included in the file definition.

Up to eight trailer labels can be written on the first track of the first extent specified for the file on each pack. They follow the user-standard header labels for the pack and are identified by UTL0, UTL1,---UTL7. For this operation, IOCS branches to the user's label routine, sets up a label area, and supplies the address of the area in Register 1. In his routine the user constructs the trailer label and then returns control to IOCS by use of the LBRET macro. IOCS then writes the trailer label. Similar to writing user header labels, these steps are repeated until eight trailer labels have been written or until the user indicates that he does not require any more labels, whichever occurs first (see OPEN Macro: Disk Output File, Writing Additional User-Standard Header

Labels). After the last trailer label is written, CLOSE deactivates the file.

Tape Input File

When an input file recorded on magnetic tape is processed, CLOSE is generally issued in the user's end-of-file routine. It initiates rewind procedures for the tape as specified in the DTFSR entry REWIND. It then deactivates the file.

If CLOSE is issued for any tape input file before the end of the data is reached, the tape is rewound as specified by the DTFSR entry REWIND, and the file is deactivated. No labels are read or checked.

Tape Output File

For a magnetic tape output file, CLOSE is issued when all records for the file have been processed. It writes any record, or block of records, that has not already been written. If a record block is partially filled, it is truncated; that is, a short block is written on the tape. Following the last record, a tape mark is written. If labels are not specified, a second tape mark is written and the tape is rewound as specified in DTFSR REWIND.

When standard labels are specified (STD in DTFSR FILABL or OUTPUT in DTFPH TYPEFLE), CLOSE causes the file trailer label to be completely written after the tape mark. The EOF1 indication, the block count accumulated during the run, and the header-label information (with HDR1 replaced by EOF1) are included in the trailer label.

IOCS accumulates the block count for the trailer label whenever logical IOCS (DTFSR) is used for an output file. When physical IOCS (DTFPH) is used, however, the problem program must accumulate the block count, if desired, and supply it to IOCS for inclusion in the standard trailer label. For this, the count (in binary form) must be moved to the 4-byte field that is labeled filenameB. For example, if filename specified in the DTFPH header entry is DETLOUT, the block count field is addressed by DETLOUTB; if filename is DETL, the field is addressed by DETLB. The user must define this address as an address constant.

If checkpoint records are interspersed with data records on an output tape, the block count accumulated by logical IOCS does not include a count of the checkpoint records. Only data records are counted. Similarly if physical IOCS is used, the problem program must omit checkpoint records and count only data records.

If user-standard labels (UTL) are to follow the standard trailer, the CLOSE routine branches to the user's routine (identified by DTFSR or DTFPH LABADDR) after the standard label has been written. Upon entry to the user's routine, IOCS supplies Code F in the low-order byte of Register 0 to indicate that an end-of-file trailer label should be built (see DTFSR LABADDR). In his routine the programmer can build a maximum of eight user-standard labels, which the CLOSE routine writes for him. After building each user-standard label, he must return to the CLOSE routine by use of the LBRET macro.

After all trailer labels are written, the CLOSE routines write two tape marks, execute the rewind option, and deactivate the file.

For the proper procedures to handle user-standard labels and/or nonstandard labels, see OPEN Macro: Tape Output File.

Other Files

When the last paper tape or card input record has been read, IOCS branches to the user's end-of-file routine where CLOSE is generally issued.

When a printer or card output file is completed, CLOSE must be issued for that file. Any record in the output area that has not been printed, displayed, or punched is transferred to the output file before the file is deactivated.

For a discussion of the SCLOS macro, for the STR adapter, see Processing with STR Devices. For a discussion of the BCLOS macro, for the BSC adapter, see Binary Synchronous Communication.

LBRET Macro

Name	Op	Operand
	LBRET	1
	LBRET	2

The LBRET (label return) macro instruction is issued at the end of the user's label routine to return to IOCS. This macro is described under Initialization: LBRET Macro.

FEOV Macro: Tape Records

Name	Op	Operand
	FEOV	filename

The FEOV (forced end-of-volume) macro instruction is used for either input or output files on tape to force an end-of-volume condition when neither a tape mark nor a reflective marker has been sensed. This indicates that processing of records on one volume is considered finished, but that more records for the same logical file are to be read from or written on the following volume.

The FEOV macro cannot be issued in the user's EOF routine when the FEOV refers to the same file that caused entry to the EOF routine, except for an unlabeled file or for a nonstandard labeled file with no user label address specified.

The symbolic name of the file, specified in the DTFSR or DTFPH header entry, is the only parameter required in this instruction.

When logical IOCS macro instructions are used for a file (DTFSR specified), FEOV initiates the same functions that occur at a normal end-of-volume condition, except trailer-label checking. For an input tape, it immediately rewinds the tape as specified by DTFSR REWIND and provides for a volume change as specified by DTFSR ALTTAPE. Trailer labels are not checked. FEOV then checks the standard header label on the new volume, and provides for user-checking of any additional user-standard header labels if DTFSR LABADDR is specified. If nonstandard labels are specified (DTFSR FILABL=NSTD), FEOV provides for user-checking if desired. For an output tape, FEOV writes the last block of records if necessary (this may be a short block) and writes a tape mark. Then it writes the standard trailer label and additional user-standard labels (if any), writes one tape mark, provides for a volume change, and writes the file header label(s) on the new volume, as specified in the DTFSR entries REWIND, ALTTAPE, FILABL, and LABADDR. If nonstandard labels are specified, FEOV provides for user-writing of trailer labels (completed volume) and header labels (new volume), if desired.

When physical IOCS macro instructions are used and DTFPH is specified for standard label processing, FEOV may be issued for an output file only. In this case FEOV writes the standard trailer label, and any additional user-standard trailer labels if DTFPH LABADDR is specified. When the new volume is mounted and ready for writing, IOCS writes the standard header label and additional user-standard header labels, if any.

FILE DEFINITION MACROS

Whenever logical IOCS macro instructions (GET, PUT, READ, WRITE, etc) are used in a program to control the input/output of records in a file, that file must be defined by a declarative macro instruction. The parameters of the macro instruction are punched in a set of entry cards. Each parameter uses the keyword format (see Macro Instruction Format). In addition to describing the file, the parameter entry cards indicate the type of processing for the file, and specify symbolic names of main-storage areas and routines used by the file.

When physical IOCS macro instructions (EXCP, WAIT, etc) are used for a file, a declarative macro instruction is required only if disk or tape files with standard labels are to be processed. No other files require definition.

The file definition macros for all files in a problem program must be assembled with that problem program. The logical IOCS routines that the problem program will require during its execution are assembled from the specifications in the file definitions. A separate set of parameter entry cards is included for the definition of each logical file, and the sets may be placed in any order. If the problem program is to be executed in a disk-resident system, a begin-definition card (DTFBG macro statement) must precede the first set of file definition cards. In all cases, regardless of where the program will be executed, an end-of-definition card (DTFEN macro statement) must follow the last set. At program assembly time, the entire group of file definition cards (punched in assembler-card format) is placed in the card deck immediately after the START card and ahead of any of the user's source program. The user's source program starts after the end-of-definition card (DTFEN).

Five different file-definition declarative macros (DTFSR, DTFDA, DTFRF, DTFSN, and DTFIS) are available for defining files processed by logical IOCS, and one macro (DTFPH) for files processed by physical IOCS. For logical IOCS operations, the file-definition macro to use for a file depends on the type of processing that will be performed for that file:

Consecutive Processing. This applies to input/output files in serial devices, or on 2311 disk when records are processed consecutively. The DTFSR (Define The File in a Serial-type device) macro is used.

Direct Access Method. Whenever a logical disk file is to be processed randomly, the direct-access-method macro DTFDA is used.

Indexed Sequential System. When a logical disk file is to be organized or processed by the indexed sequential file management system (ISFMS), the DTFIS macro is used.

Processing with STR devices. Where STR devices are used, the adapter must be defined by the macro, DTFSN.

Binary Synchronous Communication. Where BSC support macros are to be used, the data adapter (SDA II) must be defined by the DTFBS macro.

The first card of a file-definition macro instruction is called a header card, and the continuation cards are called detail entry cards. The header card is punched with:

- The symbolic name of the file in the name field. This is the name that must be specified in any logical IOCS macro instructions that refer to this file in the user's program. The symbolic file name may be up to seven characters long in a DTFSR, DTFDA, or DTFPH macro header card. In a DTFIS macro header card, the name may be up to five characters long.

  DTFIS filenames should be unique to each other if the program phases are to be stored permanently in the core image library. This will prevent one phase from being overlaid by another with the same filename. (For further information on the core image library, see the Programmer's Guide listed on the front cover of this publication.)

- The macro mnemonic operation code in the operation field. This is DTFSR, DTFDA, DTFIS, or DTFPH, depending on the type of processing to be performed for the file.

- Keyword entries in the operand field, if desired.

- A continuation punch in column 72, if detail cards are necessary.

The detail cards follow the header card, and they may be arranged in any order. Each detail card is blank in the name and operation fields and is punched with one or more operands. These operands must be the keyword type of macro parameter. That is, they are expressed as equal conditions, for example, DEVICE=DISK11. All detail cards except the final one used must be punched with a comma immediately following the last operand and with a continuation punch in column 72. They may contain comments if a space is left after the comma following the last operand. The final card may contain comments if a space is left after the last operand. When a particular detail entry does not apply to a file, that operand must be omitted.

The begin-definition card preceding the first set of DTF cards (when a disk-resident system will be used for program execution) must be punched with DTFBG in the operation field and DISK in the operand field. The name field is blank.

The end-of-definition card following the last set of DTF cards must be punched with DTFEN in the operation field. The name field is blank. The operand field may be blank, or it may be punched with one operand if the object program is not to be executed in a disk system. In this case the operand OVLAY may be specified to reduce the amount of main storage used for the program. When the object program is executed (in a system that does not utilize a disk-resident supervisor), OVLAY causes the OPEN routine to be overlaid by the user's problem program, and the CLOSE routine to overlay the user's program. All files should be opened by one OPEN macro instruction, and all files should be closed by one CLOSE macro instruction. In this case, CLOSE should be the last instruction before EOJ in the user's program.

The reference card, with DTFRF in the operation field, must precede the first DTFSN card for STR processing or the first DTFBS card for BSC processing. Where other DTF macros are present, DTFRF must follow the DTFEN card.

The detail parameter entries for each of the four declarative macros are described in the following pages. They are grouped by type of processing.

CONSECUTIVE PROCESSING (DTFSR)

The following lists show the DTFSR detail entries that apply to each type of file when records are processed consecutively. These entries are explained in the following text and shown in Figure 35.

2311 Disk Drive	2400 Tape Unit	1442 2501 2520 2540 Reader	1442 2520 2540 Punch
BLKSIZE	ALTTAPE	BLKSIZE	BLKSIZE
CONTROL	BLKSIZE	CONTROL	CONTROL
DEVICE	CHECKPT	DEVADDR	CRDERR
EOFADDR	CKPTREC	DEVICE	CTLCHR
ERROPT	CONTROL	EOFADDR	DEVADDR
IOAREA1	DEVADDR	INAREA	DEVICE
IOAREA2	DEVICE	INBLKSZ	IOAREA1
IOREG	EOFADDR	IOAREA1	IOAREA2
LABADDR	ERROPT	IOAREA2	IOREG
RECFORM	FILABL	IOREG	RECFORM
RECSIZE	IOAREA1	OUAREA	RECSIZE
TRUNCS	IOAREA2	OUBLKSZ	TYPEFLE
TYPEFLE	IOREG	RECFORM	WORKA
UPDATE	LABADDR	TYPEFLE	
VARBLD	READ	WORKA	
VERIFY	RECFORM		
WLRERR	RECSIZE		
WORKA	REWIND		
	TPMARK		
	TYPEFLE		
	VARBLD		
	WLRERR		
	WORKA		

1403 1404 1443 1445 Printer	1052 Printer-Keyboard	1285/1287 Optical Reader	2671 Paper Tape Reader
BLKSIZE	BLKSIZE	BLKSIZE	BLKSIZE
CONTROL	DEVADDR	CONTROL	DEVADDR
CTLCHR	DEVICE	COREXIT	DEVICE
DEVADDR	IOAREA1	DEVADDR	EOFADDR
DEVICE	IOREG	DEVICE	IOAREA1
IOAREA1	RECFORM	EOFADDR	IOAREA2
IOAREA2	RECSIZE	HEADER	IOREG
IOREG	TYPEFLE	IOAREA1	RECFORM
PRINTOV	WORKA	IOAREA2	RECSIZE
RECFORM		OFFLINE	TRANS
RECSIZE		IOREG	TYPEFLE
TYPEFLE		RECFORM	WORKA
UCS		RECSIZE	
WORKA		TYPEFLE	
		WORKA	

ALTTAPE=SYSnnn

This entry specifies the symbolic unit (SYSnnn) for a tape drive that will be used as an alternate when a tape file has two or more reels (volumes) of data.

The actual drive is assigned to this symbolic unit either at job-execution (load) time by the Job Control ASSGN card, or at system-generation time by the SYMUN macro instruction.

The symbolic unit for the first drive used for this file is specified in the DTFSR entry DEVADDR=SYSnnn. If DEVADDR is assigned to the same unit used for writing out checkpoint records, the user should be aware that checkpoint records may not be written on the alternate tape drive. (See the section CHKPT Macro.)

Whenever an alternate drive is specified and an end-of-volume condition (EOV, EV, or FEOV) occurs for a multi-reel file, the IOCS OPEN/CLOSE routines update the number of the drive actively in use. Thus, after an EOV (or FEOV) on reel 1, 3, 5, etc., IOCS expects the next reel to be mounted on the device assigned to ALTTAPE. After an EOV (or FEOV) on reel 2, 4, 6, etc., IOCS expects the next reel to be mounted on the device assigned to DEVADDR.

NAME	OPERATION	OPERAND#	APPLIES TO									MUST BE INCLUDED	REMARKS*
			2311 DISK DRIVE	2400 MAGNETIC TAPE UNIT	1442/2501/2520/ 2540 READER	1442/2520/2540 PUNCH	1403/1404/1443/ 1445 PRINTER	1052 PRINTER-KEYBOARD	2671 PAPER TAPE READER	1285 OPTICAL READER	1287 OPTICAL READER		
Filename†	DTFSR†		X	X	X	X	X	X	X	X	X	Each File	Header Card. Specify Symbolic File Name.
		ALTTAPE=SYSnnn		X								Multi-Volume File Using Two Tape Drives	Symbolic Unit for Alternate Tape Drive.
		BLKSIZE=n	X	X	X	X	X	X	X	X	X	Each File Except Combined File with Separate I/O Areas	Length of I/O Area. n = Maximum Number of Characters.
		CHECKPT=n		X								CHKPT Macro Used	
		CKPTREC=YES		X								Bypass Checkpoint Records on Input	Applies to Input File Only.
		CONTROL=YES	X	X	X	X	X			X	X	CNTRL Macro Used	Does Not Apply to 2501. CTLCHR Must Be Omitted.
		COREXIT=Name								X	X	For Correction Routines	Symbolic Name of Correction Routine
		CRDERR=RETRY				X						Punch Again on Error Condition	Applies to 2520 and 2540.
		CTLCHR=YES				X	X					Logical Records Have Control Character in First Position	Each Record Must Contain a Control Character. CONTROL Must Be Omitted and the CNTRL Macro Must Not Be Used for This File
		†DEVADDR=SYSnnn		X	X	X	X	X	X	X	X	Each File Except Disk	Symbolic Unit for the I/O Device Used for the File.
		DEVICE=DISK11	X									Disk Input/Output File	Include for Each File, and Specify Proper Name after DEVICE=.
		TAPE		X								Tape Input/Output File	
		READ01			X							2501 Input File	
		READ20			X	X						2520 Input/Output File	
		READ40			X	X						2540 Input/Output File	
		READ42			X	X						1442 Input/Output File	
		PRINTER					X					1403, 1404, 1443, 1445 Output	
		CONSOLE						X				1052 Input/Output	
		PTAPERD							X			2671 Input File	
		READ85								X		1285 Input File	
		READ87T									X	1287 Tape Input	
		READ87D									X	1287 Document Input	
		EOFADDR=Name	X	X	X				X	X	X	Input File	Symbolic Name of End-of-File Routine
		ERROPT=IGNORE	X	X								Process Error Records	Applies Only to Disk or Tape Input. Prevents Job Termination on Error Condition. Enter Desired Specification after ERROPT=.
		SKIP										Skip over Error Records	
		Name										User Routine for Error Records	
		FILABL=STD		X								Check or Write Standard Labels	Include for Tape Input/Output and Specify Desired Operation after FILABL=.
		NSTD										File Contains Nonstandard Labels	
		NO										Unlabelled File	
		HEADER=YES								X	X	Header record will be read by Open	If this entry is omitted, OPEN assumes no header record
		INAREA=Name			X							Separate Areas for Input and Output for a Combined File	Applies Only to 1442.
		INBLKSZ=n			X							Separate Areas for Input and Output for a Combined File	Applies Only to 1442. Length of INAREA.
		IOAREA1=Name	X	X	X	X	X	X	X	X	X	Each File Except Combined File with Separate I/O Areas	Symbolic Names of Input/Output Area. Same as Used in DS.
		IOAREA2=Name	X	X	X	X	X		X	X	X	Two I/O Areas Used. Not Valid if DEVICE=READ87D	Symbolic Name of Second Input/Output Area. Same as Used in DS.
		IOREG=n	X	X	X	X	X	X	X	X	X	Process Blocked Records in I/O Areas; Process, in the Input Area, Variable-Length Records Read Backwards; Process in Two I/O Areas; or Process Undefined Records On 1285	n = Number of General Purpose Register 2-11. Omit WORKA=YES.
		LABADDR=Name	X	X								Check/Build Additional User-Standard Labels, or Process Nonstandard Labels	Symbolic Name of User's Label Routine.

Figure 35. DTFSR Entries (Part 1 of 2)

NAME	OPERATION	OPERAND#	APPLIES TO									MUST BE INCLUDED	REMARKS*
			2311 DISK DRIVE	2400 MAGNETIC TAPE UNIT	1442/2501/2520/2540 READER	1442/2520/2540 PUNCH	1403/1404/1443/1445 PRINTER	1052 PRINTER-KEYBOARD	2671 PAPER TAPE READER	1285 OPTICAL READER	1287 OPTICAL READER		
Filename†	DTFSR†	OFFLINE=YES								X	X	Allow program controlled correction on reject character	If this entry is omitted, reject characters are retried 9 times and keyboard correction allowed on tenth retry.
		OUAREA=NAME			X							Separate Areas for Input and Output for a Combined File	Applies only to 1442.
		OUBLKSZ=n			X							Separate Areas for Input and Output for a Combined File	Applies Only to 1442. Length of OUAREA.
		PRINTOV=YES					X					PRTOV Macro Used	
		READ=FORWARD		X									If This Entry Omitted, IOCS Assumes FORWARD.
		BACK										Read Tape Backwards	Does not Apply to Variable-Length Blocked Records.
		RECFORM=FIXUNB	X	X	X	X	X	X	X	X	X		Specify as Needed for Fixed-Length Unblocked Records. If This Entry Omitted, IOCS Assumes FIXUNB.
		FIXBLK	X	X								Fixed-Length Blocked Records	
		VARUNB	X	X		X	X					Variable-Length Unblocked Records	Disk or Tape Records Require Record-Length Field.
		VARBLK	X	X								Variable-Length Blocked Records	Blocks Require Block-Length Field. Records Require Record-Length Field.
		UNDEF	X	X		X	X	X	X	X	X	Undefined Records	
		RECSIZE=n	X	X		X	X	X	X	X	X	Fixed-Length Blocked Records Undefined Records	n = Number of Characters in Record. n = Number of a Register 2-11.
		REWIND=UNLOAD		X								Unload on CLOSE or End-of-Volume	Omit to Rewind Only, at OPEN, CLOSE, or End-of-Volume.
		NORWD										Prevent Rewinding	
		TPMARK=NO		X								Prevent a Tape Mark from Being Written Ahead of Data Records	Applies to Unlabeled Tape Files and Files with Non-Standard Labels.
		TRANS-Name							X			Tape Punched with Code Other than EBCDIC and IOCS is to Perform Translation	Symbolic Name of Code-Translation Table.
		TRUNCS=YES	X									Fixed-Length Blocked Records with Short Blocks	Include for Output if TRUNC Macro Used. Include for Input if TRUNC Macro Was Used When File Was Created.
		†TYPEFLE=INPUT	X	X	X			X	X	X	X	Each file	Specify Proper Type after TYPEFLE=. CMBND Applies to 1442, or 2520 or to 2540 if Punch-Feed-Read Special Feature is installed.
		OUTPUT	X	X		X	X	X					
		CMBND			X								
		UCS=NO					X						Applies only to 1403. CONTROL=YES Required.
		YES					X					Ignore Command Rejects on 1403 Printer without UCS Feature	
		UPDATE=YES	X									PUT Used For a Disk Input File	
		VARBLD=n	X	X								Variable-Length Blocked Records Built in Output Area	n = Number of a Register 2-11.
		VERIFY=YES	X									Check Record Written on Disk	
		WLRERR=Name	X	X								User Routine for Wrong-Length Records	Symbolic Name of Wrong-Length-Record Routine. If Omitted, Error Handled As in ERROPT, or Job Terminated.
		WORKA=YES	X	X	X	X	X	X	X	X	X	GET or PUT Specifies a Work Area. Not Valid if DEVICE=READ87D.	Omit IOREG=n.

† Must be included. Other entries are included when applicable.

When two or more choices are shown, select only the appropriate one and enter it after the = sign.

* The header and each detail card except the last one used in a file set must contain a continuation punch in column 72. Each detail card except the last one used must also contain a comma after the last operand.

In all entries: Solid caps must be entered as shown (For Example, CONTROL=YES).

Lower-case letters are to be replaced by programmer's symbolic name or a number (For example, Filename in header card, or BLKSIZE=n where n is replaced).

n is a decimal self-defining value.

Figure 35. DTFSR Entries (Part 2 of 2)

BLKSIZE=n

This entry indicates the size of the input, or output, area specified by IOAREA1. BLKSIZE specifies the maximum number (n) of characters that will be transferred to, or from, the area at any one time. When variable-length records are read, or written, the area must be large enough to accommodate the largest block of records, or the longest single record if the records are unblocked.

If card-punch or printer-output records include control characters (DTFSR CTLCHR specified) and/or record-length fields for variable-length records (RECFORM=VARUNB), the BLKSIZE specification must include the extra bytes allotted in the main-storage output area.

If two input, or output, areas are used for a file (IOAREA1 and IOAREA2), the size of only one area is specified in this entry.

IOCS uses this specification to:

- Construct the count field of the CCW for an input file.
- Construct the count field of the CCW for an output file of fixed-length records.
- Check physical record length for a file of fixed-length blocked input records.
- Determine if the space remaining in the output area is large enough to accommodate the next variable-length output record.

CHECKPT=n

This entry is required for a tape input file whenever the CHKPT (checkpoint) macro instruction is included in the problem program. It is required for a tape output file of data records when the CHKPT instruction is used, unless the checkpoint records are also to be written on this file. Whenever an output file is to contain checkpoint records interspersed with data records, this entry must be omitted.

When the macro-library checkpoint routines are assembled, a table of restart information is included (see CHKPT Macro). This table contains a 4-byte field (full word) for each tape drive used by the problem program and specified by the first parameter of the CHKPT instruction. Each of these 4-byte fields contains information for properly positioning the data file when the job is restarted after an interruption. Part of the repositioning information is the block count for the file, which is contained in two of the four bytes. The block count indicates the number of physical tape records that have been read (or written), and thus it must be incremented whenever an I/O operation occurs during program execution. IOCS can update the block count for a particular file if the 4-byte field related to the file is identified. Therefore the user specifies in the CHECKPT entry the number (n) of the 4-byte field that IOCS is to use for this file. For example, if three input/output tape files are used in a job, he could plan that the three 4-byte fields would be identified by the numbers 1, 2, and 3. Then in the file definition for each file, he specifies (by CHECKPT=n) one of the three numbers (1, 2, or 3).

This entry is omitted for an output file containing both data and checkpoint records because, on restart, the tape is positioned by the Restart program at the first data record following the checkpoint records that are used for restarting.

CKPTREC=YES

This entry is required if a tape input file will contain checkpoint records interspersed among the data records. With this entry, IOCS recognizes the checkpoint records and bypasses them.

CONTROL=YES

This entry must be included if a CNTRL macro instruction will be issued for this file. A control command issues orders to the I/O device to perform non-data operations such as card or document stacker selection, carriage skipping, line marking, tape rewinding, etc.

From this specification IOCS generates a CCW for control commands and also recognizes a CNTRL macro. If the CNTRL macro is not used in the program, then DTFSR entry CONTROL=YES must be omitted.

When CONTROL is included, the DTFSR entry CTLCHR must not be included.

COREXIT=Name

COREXIT provides an exit to the user's error correction routine for the 1285 or 1287 Optical Reader. After a GET, WAITF, or CNTRL macro (to increment or eject and/or stacker select a document) is executed and an error occurs, the error correction routine is entered, and an indication of the reason for the entry is provided in Filename+17.

Filename+17 contains the following hexadecimal bits indicating the conditions that occurred during the last line or field read. Filename+17 should also be tested for appropriate error indication(s) after issuing the optical reader macros DSPLY, RESCN, RDLNE, CNTRL READKB, and CNTRL MARK. More than one error condition may be present.

X'01' A data check has occurred. (Five read attempts for journal tape processing or three read attempts for document processing were made.)

X'02' The operator corrected one or more characters from the keyboard. (Journal tape processing only).

X'04' A wrong length record condition has occurred. (Ten read attempts were made.) Not applicable for undefined records.

X'08' An equipment check resulted in an incomplete read. (Ten read attempts were made.)

X'10' A non-recovery error occurred.

X'20' A stacker-select command was given after the allotted time had elapsed and the document had been put in the reject pocket (Document processing only).

X'40' The document scanner was unable to locate the reference mark. (Ten attempts were made.)

Filename+17 can be interrogated by the user to determine the reason for entry to the error-correction routine. This may be done by setting up an address constant for filename. This also permits the user to reference the DTF area at any place in his program. Choice of action in the user's error-correction routine will be determined by the particular application involved.

If the user issues the CNTRL, DSPLY, RDLNE, or RESCN macros in his error routine within COREXIT, he must first save and later restore registers 14 and 15. All exits from COREXIT (except as noted) must be to the address in register 14. This address returns the user to the point in the program from which the branch to COREXIT occurred. If the command chain bit is on in the READ CCW for which the error occurred, IOCS completes the chain upon this return from the COREXIT routine.

When processing journal tapes, a nonrecovery error (torn tape, tape jam, etc.) normally requires complete reprocessing of the tape. In this case, the user must not branch tc the address in register 14 from the COREXIT routine. Following a nonrecovery error, the Optical Reader file must be closed, the condition causing the nonrecovery must be cleared, and the file must be reopened before processing can continue.

When processing documents, a nonrecovery error (indicating that a jam occurred during a document incrementation operation or a scanner control failure has occurred, etc.), requires that the document be removed, either manually or by nonprocess runout. The user program should branch to read the next document. Also, if the 1287 scanner is unable to locate the document reference mark, the document cannot be processed. The document must be ejected and stacker selected before attempting to read the following document or looping will occur. In any of these cases, the user must not branch to the address in register 14 from the COREXIT routine. The user should ignore any output resulting from the document in any case.

Eight binary counters are used to accumulate totals of certain 1285 and 1287 error conditions. These counters each occupy four bytes, starting at Filename+20. Filename is the same, specified in the DTF header entry. The error counters are:

Counter	Address	Contents
1	Filename+20	Incomplete read (equipment check)
2	Filename+24	Incomplete read uncorrectable after ten read attempts.
3	Filename+28	Wrong length records (not applicable for undefined records).
4	Filename+32	Wrong length records uncorrectable after ten read attempts (not applicable for undefined records).
5	Filename+36	Keyboard corrections (journal tape only).
6	Filename+40	Journal tape lines, including retried lines, or document fields including retried fields, in which data checks are present.
7	Filename+44	Lines marked (journal tape only).
8	Filename+48	Count of total lines read from journal

tape or the number of CCW chains executed during document processing.

The user may print out the contents of these counters for analysis, at end-of-file, or at end-of-job, or he may ignore the counters. (Binary contents of the counters should be converted to a printable format.)

Note: The user cannot issue a GET, READ, or WAITF macro in his error correction routine. In this routine, records must not be processed. The record that caused the exit to the error routine will be available for processing upon return to the user's mainline program. Any processing included in the error routine would be duplicated after return to the mainline program.

CRDERR=RETRY

This entry applies only to a card output file in the IBM 2520 or 2540. It specifies the operation to be performed if an error is detected.

Normally if a punching error occurs, it is ignored and operation continues. The error card is stacked in pocket 1 (2520) or pocket P1 (2540). Correct cards are stacked in pocket 2 (2520) or pocket P2 (2540). If this CRDERR entry is included to specify retrying, however, IOCS also notifies the operator and then enters the wait state when an error condition occurs. The operator can either terminate the job or instruct IOCS to repunch the card.

From this specification, IOCS generates a retry routine for the 2520 or 2540. IOCS also generates a save area for the card punch record read from the 2540.

CTLCHR=YES

The CTLCHR (control character) entry applies only to printer and punch output files. It is included if each logical record to be written or punched contains a control character (carriage control or stacker selection) in the record itself, in the main-storage output area. For fixed-length or undefined records, the control character must be the first character. For variable-length records, it is the first character after the record-length field. The control character codes are the same as the command codes (including the modifier bytes) used for a punch command or a print command (delayed only).

With this entry, the IOCS routines cause the control-character-specified printer or card punch order to be issued to the I/O device. Printing or punching begins with the second character in the record.

When this CTLCHR entry is not included, any control functions desired must be performed by the CNTRL macro.

DEVADDR=SYSnnn

This entry specifies the symbolic unit (SYSnnn) to be associated with this logical file. The symbolic unit represents an actual I/O device address. The symbolic unit may be:

SYSRDR for system control-card reader

SYSLST for system printer

SYSIPT for main system input device

SYSOPT for main system output device

SYSLOG for control-card logging device

SYS000-SYS254 for other devices in the system.

This is generally a unique number for each logical file (except files on disk).

The symbolic unit (SYSnnn) is used in the Job Control ASSGN card (or in the Supervisor-Assembly macro SYMUN) to assign the actual I/O device address to this file. If the ASSGN card is used to assign the device for program execution, the file of logical records can be read from or written on different units at different times. For example, a reel of tape may be mounted on any tape drive that is available at the time the job is ready to be run, by merely assigning that drive to the symbolic unit.

Whenever two devices are used for one logical file, such as an alternate tape drive (specified in DTFSR ALTTAPE), this DEVADDR entry specifies the symbolic unit for the first device.

The symbolic unit is specified for all units except the 2311 disk drive. For files on this unit, DEVADDR is omitted. The symbolic unit for a disk drive is supplied by a Job Control XTENT card.

DEVICE=

This entry must be included to state the I/O device associated with this logical file. One of the following specifications must be entered immediately after the = sign.

DISK11 - for an input or output file on disk (2311).

TAPE - for an input or output file recorded on magnetic tape (2401, 2402, 2403, 2404).

PRINTER - for reports printed on a 1403, 1404, 1443, or 1445.

READ01 - for an input card file in a 2501.

READ20 - for an input or output card file in a 2520.

READ40 - for an input or output card file in a 2540.

READ42 - for an input or output card file in a 1442.

CONSOLE - for input from and output to the printer-keyboard (1052).

PTAPERD - for an input file recorded on paper tape (2671).

READ85 - for input from a 1285 Optical Reader.

READ87T - for journal tape processing for a 1287 Optical Reader.

READ87D - for document processing for a 1287 Optical Reader.

From this specification, IOCS sets up the proper CCWs and device-dependent routines for this file. For document processing on the 1287 Optical Reader, the coding of CCWs is accomplished by the user. This allows command chaining, comparing, and branching without experiencing an interrupt for each function.

EOFADDR=Name

This entry must be included for:

- Card reader files
- Paper tape reader files
- Magnetic tape input files
- Consecutive disk input files
- Optical roll reader files.

It specifies the symbolic name of the user's end-of-file routine. IOCS will automatically branch to this routine on an end-of-file condition. In his routine, the programmer can perform any operations required for the end of the job, and he generally issues the CLOSE instruction for the file.

IOCS detects end-of-file conditions as follows:

- Card reader - by recognizing /* punched in card columns 1 and 2. If cards are allowed to run out, without a /* trailer card, an error condition is signalled to the operator (intervention required).
- Paper tape reader - by recognizing the end of tape when the end-of-file switch is set at ON.
- Magnetic tape input - by reading a tape mark and EOF in the trailer label when standard labels are specified. If standard labels are not specified, IOCS assumes an end-of-file condition when the tapemark is read. The user must determine, in his routine, that this actually is the end of the file.
- Consecutive disk input - by reading an end-of-file record or reaching the end of the last extent supplied by user.
- Optical Reader input--when reading data from documents on a 1287, end-of-file condition is recognized by depression of the end-of-file key on the console when the input hopper is empty. When processing journal tapes on a 1285 or 1287, end-of-file is detected by depression of the end-of-file key after the end of the tape has been sensed.

When IOCS detects the end of file, it branches to the user's routine specified by EOFADDR. If journal tapes are being processed it is the user's responsibility to determine if the current roll is the last roll to be processed. For 1285, it is suggested that this be accomplished by keying in header information at the beginning of each roll. This information could then be interrogated in this routine to determine whether it is the last roll. Regardless of the situation, the tape file must be closed for each roll within this routine. If the current roll is not the last, OPEN must be issued. The OPEN macro instruction allows header (identifying) information to be entered at the reader keyboard and read by the processor when using logical IOCS. The same procedure, as well as the method described under the OPEN macro, can be used for 1287 processing of multiple journal tape rolls.

ERROPT=

This entry applies to disk or tape input files, and it specifies functions to be performed for an error block.

If a parity error is detected when a block of consecutive disk records is read, the disk block is reread 10 times before it is considered an error block. If a parity

error is detected when a block of tape records is read, the tape is backspaced and reread 100 times before the tape block is considered an error block. If either FILABL=STD or CHECKPT, or both, is specified, the error block is included in the block count that is taken. After this the job is automatically terminated, unless this ERROPT entry is included to specify other procedures to be followed on an error condition. Either IGNORE, SKIP, or the symbolic name of an error routine can be specified in this card. One of these specifications is entered immediately after the = sign in this keyword operand. The functions of these 3 specifications are:

IGNORE — The error condition is completely ignored, and the records are made available to the user for processing.

SKIP — No records in the error block are made available for processing. The next block is read from disk or tape, and processing continues with the first record of that block. The error block is included in the block count, however.

Name — IOCS branches to the user's routine, where he may perform whatever functions he desires to process or make note of the error condition. Register 1 contains the address of the block in error, and Register 14 contains the return address.

In his routine, the programmer should address the error block, or records in the error block, by referring to the address supplied in Register 1. The contents of the IOREG register or the work area (if either is specified) may vary and therefore should not be used for error blocks. Also, the programmer must not issue any GET instructions for records in the error block. If he uses any other IOCS macros in his routine, he must save the contents of Registers 14 and 15 and restore them prior to returning to IOCS. At the end of his routine, he must return to IOCS by branching to the address in Register 14. When control is returned to the problem program, the first record of the next block is available for processing in the main program.

This ERROPT entry does not apply to disk or tape output files. The job is automatically terminated if a parity error still exists after IOCS attempts 10 times to write a disk output block, or 15 times to write a tape output block. For tape, this includes erasing forward 14 times.

This entry applies to wrong-length records if the DTFSR entry WLRERR is not included.

FILABL=

This entry applies to a tape input or output file. One of the following specifications is entered immediately after the = sign :

STD — for a tape input file if standard labels are to be checked by IOCS, or for a tape output file if standard labels are to be written by IOCS.

NSTD — for a tape input or output file that has nonstandard labels. These labels may be processed by the user (see Initialization: Nonstandard Tape Labels). NSTD is specified for standard input labels if they are not to be checked by IOCS.

NO — for a tape file that does not contain labels. The entry FILABL=NO may be omitted, if desired, and IOCS will assume that there are no labels.

HEADER=YES

This entry is required if header (identifying) information is to be keyed in by the operator on the 1285 or 1287 Optical Reader keyboard. The OPEN routine reads the information only when this entry is present. If the entry is not included, OPEN assumes no header information is to be read.

INAREA=Name

This entry applies only to a card file in an IBM 1442 that is to be updated (TYPEFLE=CMBND) and for which separate input and output areas are required. INAREA specifies the symbolic name of the input area to which the card record is to be transferred. OUAREA is used in conjunction with INAREA, and both IOAREA1 and IOAREA2 must be omitted.

If the same I/O area is to be used for both input and output in a combined file,

INAREA and OUAREA are omitted, and IOAREA1 specifies the name of the I/O area.

This entry does not apply to combined files in an IBM 2520 or 2540.

INBLKSZ=n

This entry is used in conjunction with INAREA for a combined file in the 1442 when separate input and output areas are required. It specifies the maximum number (n) of characters that will be transferred to the input area (INAREA) at any one time. Whenever this entry is included, the corresponding entry OUBLKSZ must also be included, and BLKSIZE must be omitted.

IOAREA1=Name

This entry is included to specify the symbolic name of the input, or output, area used by this file. The input/output routines will transfer records to or from this area. The specified name must be the same as the name used in the DS instruction that the programmer must set up to reserve this area of main storage . If RECFORM specifies VARUNB or VARBLK, the I/O area must begin on a halfword boundary.

From this specification, IOCS constructs the data address field of the CCW for this file.

For a disk output file, the user must reserve eight bytes at the beginning of his I/O area, ahead of the positions allotted for data records. These eight bytes are necessary to allow IOCS to construct the count area for the disk record.

This entry must not be included for a 1442 card file if INAREA and OUAREA are specified for the file.

The IBM 1052 Printer-Keyboard and the 1287 Optical Reader in document mode can have only one I/O area, which must be IOAREA1.

IOAREA2=Name

Two input, or output, areas can be planned for a file, to permit an overlap of data transfer and processing operations. When this is done, this IOAREA2 entry must be included. It specifies the symbolic name of the second I/O area. The name must be the same as the name used in the DS instruction that the programmer must set up for this area. The second I/O area must be the same length as the first I/O area. If RECFORM specifies VARUNB or VARBLK, the I/O area must begin on a halfword boundary.

This entry must not be included for a 1442 card file if INAREA and OUAREA are specified for the file. Also this entry does not apply to the IBM 1052, which cannot utilize a second area, and likewise it must not be used to process documents on the 1287 optical reader.

IOCS uses this specification to construct the data address field of a CCW and to determine that overlap of I/O and processing is possible.

For a disk output file, the user must reserve eight bytes at the beginning of his I/O area, ahead of the positions allotted for data records. These eight bytes are necessary to allow IOCS to construct the count area for the disk record.

IOREG=n

This entry specifies the general-purpose register (n) that the input/output routines can use to indicate which individual record is available for processing. IOCS puts the absolute base address of the current record in this register each time a GET or PUT is issued. Any register number 2-11 may be specified. The other registers (0-1 and 12-15) cannot be used (see Base Register Instructions for register usage).

The same register may be specified in the IOREG entry for two or more files in the same program, if desired. In this case the problem program must store the address supplied by IOCS for each record, so that the address is available for processing the contents of the record.

This entry must be included whenever:

- Blocked input or output records (from disk or tape) are processed directly in the I/O area.
- Variable-length unblocked tape records are read backwards and processed directly in the input area.
- Two input, or output, areas are used and the records (either blocked or unblocked) are processed in the I/O areas.
- Undefined journal tape records are processed by the 1285 and 1287 Optical Readers. The "read" by these devices is accomplished by a backward scan which places the rightmost character in the record in the rightmost position in the I/O area and subsequent characters in sequence from right to left. The register defined by IOREG is used to indicate to the user, the leftmost position of the record.

Whenever this entry is included for a file, the DTFSR entry WORKA must be

omitted, and the GET, or PUT, instructions must not specify work areas.

LABADDR=Name

The user may require one or more disk or tape labels in addition to the standard file header label or trailer label (on tape). If so, he must include his own routine to check, or build, his user-standard label(s). The symbolic name of his routine is specified in this entry. IOCS branches to this routine after it has processed the standard label. This entry is also required whenever nonstandard labels are to be checked or written by the user (DTFSR FILABL specifies NSTD).

LABADDR allows one user's label routine to be specified for all types of labels for the file: header labels, end-of-file labels, and end-of-volume labels. On an input file, the user can determine the type of label that has been read by the identification in the label itself. For an output tape file, however, IOCS indicates to the user the type of label that is to be written. For this, IOCS supplies a code in the low-order byte of Register 0, as follows:

O - Header label (letter O)
F - End-of-file label
V - End-of-volume label

In his routine the user can test this byte and then build the appropriate type of label.

Logical IOCS uses general registers 14 and 15 for linkage to and from a called routine; therefore, these registers must not be destroyed. If they are required by the user's label routine, they must first be saved and then restored prior to returning to the mainline of the program. At the end of his routine, the programmer must return to IOCS by use of the LBRET macro.

OFFLINE=YES

This entry must be included whenever the 1285 and 1287 (journal tape mode) Optical Readers are operated in the offline mode. It generates coding for the RDLNE macro.

If this entry is omitted, an additional read is performed when five attempts to read a line containing a data check(s) are unsuccessful. It forces on-line correction of any unreadable character(s) by individually projecting the unreadable character(s) on the display scope. The operator must key in a correction (or reject) character(s). The RDLNE macro cannot be used if this parameter is omitted.

OUAREA=Name

This entry is used in conjunction with INAREA for a combined file in an IBM 1442 that requires separate input and output areas. It specifies the symbolic name of the output area from which the updated card record is punched.

OUBLKSZ=n

This entry is used in conjunction with OUAREA for a combined file. Similar to INBLKSZ, it specifies the maximum number (n) of characters that will be transferred from the output area (OUAREA) at any one time.

PRINTOV=YES

This entry must be included whenever the PRTOV macro instruction is included in the problem program.

READ=

This entry may be included for a magnetic tape input file to specify the direction in which the tape is to be read. One specification or the other is entered immediately after the = sign:

FORWARD — for a tape read in the normal forward direction.

BACK — for a tape read backwards. A tape file may be read backwards if it contains unblocked records, fixed-length blocked records, or undefined records. READ=BACK cannot be specified for a file that contains variable-length blocked records.

If this entry is omitted, IOCS assumes forward reading.

RECFORM=

This entry specifies the type of records (fixed or variable length, blocked or unblocked, or undefined) in the input or output file. If the entry RECFORM is omitted, fixed-length unblocked records are assumed. One of the following specifications may be entered immediately after the = sign:

FIXUNB — for fixed-length unblocked records.

FIXBLK for fixed-length blocked records. This applies only to disk and magnetic tape input or output.

VARUNB for variable-length unblocked records. This applies only to disk input or output (2311), magnetic tape input or output (2400), card punch output (1442, 2520, or 2540), and printer output (1403, 1404, 1443, or 1445).

VARBLK for variable-length blocked records. This applies only to disk and magnetic tape input or output.

UNDEF for undefined records. This applies to any file except card input (1442, 2501, 2520, or 2540).

Thus the records in a file can be specified as follows:

Disk and magnetic tape input or output: FIXUNB, FIXBLK, VARUNB, VARBLK, or UNDEF

Card input: FIXUNB

Card output: FIXUNB, VARUNB, or UNDEF

Printer output: FIXUNB, VARUNB, or UNDEF

Printer-keyboard input or output: FIXUNB or UNDEF

Paper tape input: FIXUNB or UNDEF

Optical reader input: FIXUNB or UNDEF

RECSIZE=n

This entry must be included for journal tape, or magnetic tape records that are fixed-length blocked (RECFORM=FIXBLK) or undefined (RECFORM=UNDEF), in an input or output file. For other files of records, this entry must be included whenever records are undefined (RECFORM=UNDEF).

For fixed-length blocked disk or tape records, this entry specifies the number (n) of characters in an individual record. The input/output routines use this factor for blocking or deblocking records, and for checking record length of input records.

For undefined records, this entry specifies the number (n) of the general-purpose register that will contain the length of each individual input or output record. This may be any register 2-11. When undefined records are read, IOCS supplies the physical record size in the register. When undefined records are built, the programmer must load the length of each record (in bytes) into the register before he issues the PUT instruction for the record. This becomes the count portion of the CCW that IOCS sets up for this file. Thus it determines the length of the record to be transferred to the output device. If an undefined punch or printer output record contains a control character in the main-storage output area (DTFSR CTLCHR specified), the length loaded into the RECSIZE register must also include one byte for this character.

If record format is specified as UNDEF for a 1287 Optical Reader that is processing documents, RECSIZE contains only the length of the last field of a document read by the user-supplied channel command word chain.

Note: When processing undefined records in document mode, the user can gain complete usage of the register normally used in the RECSIZE parameter. This can be accomplished by insuring that the suppress-length-indication (SLI) flag is always ON when processing undefined records.

REWIND=

If no specifications are given by the programmer, tape files are automatically rewound, but not unloaded, on an OPEN or CLOSE instruction and on an end-of-volume condition. If other operations are desired for a tape input or output file, this entry may be included with one of the following entered immediately after the =sign:

UNLOAD to rewind the tape on OPEN, and to rewind and unload on CLOSE or an end-of-volume condition.

NORWD to prevent rewinding the tape at any time.

TPMARK=NO

If this entry is included for unlabeled tape files, a tapemark will not be written as the first record on the tape. If omitted, a tapemark is written.

For output files containing nonstandard tape labels, this specification must be included to prevent a tapemark from being written after the last nonstandard header label and before the first data record. If omitted, the tapemark is written.

TRANS=Name

This entry applies to an input file read from the IBM 2671 Paper Tape Reader, and it

specifies the symbolic name of a code-translation table.

The input file records may be punched in 5-, 6-, 7-, or 8-channel paper tape, using any one of several different recording codes. If a code other than EBCDIC is used, it must be translated to EBCDIC code for use in System/360 programming. For IOCS to perform this translation, the user provides a translation table and specifies the symbolic name of the table in this TRANS entry. Then the logical IOCS routines translate the paper tape code and make the record available to the programmer in usable form directly in the input area, or in the work area if one is specified in the GET instruction.

The translation table must conform to the specifications of the machine instruction TRANSLATE.

TRUNCS=YES

This entry applies to disk files with fixed-length blocked records (RECFORM=FIXBLK) when short blocks are to be processed. It must be included:

- For an output file if the TRUNC macro instruction is to be issued in the problem program.
- For an input file if the TRUNC macro was issued to write short blocks when the file was originally created.

TYPEFLE=

This entry must be included to specify the type of file (input, output, or combined). One of these specifications is entered immediately after the = sign:

INPUT must be specified for:
- 2311 disk input files (with or without updating)
- 2400 magnetic tape input files
- 1442, 2501, 2520, 2540 card reader files
- 1052 keyboard input (only the GET instruction may be issued)
- 2671 paper tape files
- 1285 optical reader files
- 1287 optical reader files

OUTPUT must be specified for:
- 2311 disk output files
- 2400 magnetic tape output files
- 1442, 2520, 2540 card punch files
- 1403, 1404, 1443, 1445 printer output
- 1052 printer output (only the PUT instruction may be issued)

CMBND must be specified for a 1442, 2520, or 2540 card file that is to be updated. That is, card records are to be read, processed, and then punched (PUT) in the same cards from which they were read. Thus input and output operations are combined for the same file. This operation can be performed in the IBM 1442 or 2520, in the IBM 2540 if the punch-feed-read special feature is installed and cards are fed and read in the punch feed. (See PUT Macro: Updating.)

From this specification, IOCS sets up the CCW for this file and generates the proper blocking or deblocking routines.

UCS=

This entry is used in conjunction with the CNTRL UCS macro instruction, which controls data checks resulting from unprintable characters in an IBM 1403 Printer with the Universal Character Set special feature. DTFSR UCS= should be specified if a program that includes CNTRL UCS will ever be executed in a system that utilizes a 1403 without the UCS feature. CNTROL UCS issued for a non-UCS printer causes a command reject to occur and the system to enter the wait state. This DTFSR UCS= specification can be included to override the effect of command rejects. Either YES or NO may be specified after the = sign.

YES must be included to ignore command rejects from a non-UCS printer and continue processing.

NO may be specified. If it is, a command reject is accepted and the system enters the wait state. If DTFSR UCS= is omitted, UCS=NO is assumed.

This DTFSR UCS specification is generally omitted if a program will always be executed in a system that includes a printer with the UCS feature.

UPDATE=YES

This entry must be included if a disk input file (TYPEFLE=INPUT) is to be updated. That is, disk records are to be read, processed, and then transferred back (PUT) to the same disk-record locations from which they were read.

VARBLD=n

Whenever variable-length blocked records are built directly in the output area (no work area specified), this entry must be included. It specifies the number (n) of a general-purpose register, which will always contain the length of the available space remaining in the output area. Any register 2-11 may be specified.

After the PUT instruction is issued for a variable-length record, IOCS calculates the space still available in the output area, and supplies it to the programmer in this VARBLD register. The programmer then compares the length of his next variable-length record with the available space to determine if the record will fit in the area. This check must be made before the record is built. If the record will not fit, the programmer issues a TRUNC instruction to transfer the completed block of records to the tape file. Then the present record is built at the beginning of the output area, as the first record in the next block.

VERIFY=YES

This entry is included if the user wants disk records to be checked after they are written. If this entry is omitted, any records written on disk are not verified.

WLRERR=Name

This entry applies only to disk or tape input files. It specifies the symbolic name of a user's routine to which programming will branch if a wrong-length record is read. In his routine the user can perform any operation he desires for wrong-length records. However, he must not issue any GET macro instructions for this file. Also, if he uses any other IOCS macros in his routine, he must save the contents of Registers 14 and 15 and restore them prior to returning to IOCS. The address of the wrong-length record is supplied by IOCS in Register 1. At the end of his routine, the user must return to IOCS by branching to the address in Register 14.

Whenever fixed-length blocked records or variable-length records are specified (RECFORM=FIXBLK,=VARUNB, or =VARBLK), the machine check for wrong-length records is suppressed and IOCS generates a programmed check of record length. For fixed-length blocked records, record length is considered incorrect if the physical disk or tape record (gap to gap) that is read is not a multiple of the logical-record length (specified in DTFSR RECSIZE), up to the maximum length of the block (specified in DTFSR BLKSIZE). This permits the reading of short blocks of logical records, without a wrong-length-record indication.

For variable-length records, record length is considered incorrect if the length of the disk or tape record is not the same as the block length specified in the first two bytes of the block.

When fixed-length unblocked records are specified (RECFORM=FIXUNB), IOCS checks for a wrong-length-record indication that may have been set as the result of an I/O operation.

If this WLRERR entry is omitted from the set of DTFSR entries but a wrong-length record is detected by IOCS, one of the following will result:

- If the DTFSR ERROPT entry is included for this file, the wrong-length record will be treated as an error block and handled according to the user's specifications for an error (IGNORE, SKIP, or Name of error routine).
- If the DTFSR ERROPT entry is not included, the job will be terminated.

The WLRERR entry does not apply to undefined records on tape. However, a WLRERR routine may be used to handle undefined records on disk.

WORKA=YES

Input/output records can be processed, or built, in work areas instead of the input/output areas. If this is planned, this WORKA=YES entry must be included, and the programmer must set up the work area(s) in main storage. Then the symbolic name, used in the DS instruction that reserves the work area must be specified in each GET, or PUT, instruction. On a GET or PUT, IOCS moves the record to, or from, the specified work area.

Whenever this entry is included for a file, the DTF entry IOREG must be omitted. When variable-length records are used, WORKA must be defined to start on a halfword boundary. This entry is not applicable when documents are being processed on the 1287.

DIRECT ACCESS METHOD (DTFDA)

The DTFDA detail entries that apply to a file when records are processed by the direct access method are explained in the following text and shown in DTFDA Entries (Figure 36).

AFTER=YES

This entry must be included if any record is to be added to a file following the last record previously written on a track. That is, whenever the macro instruction WRITE Filename,AFTER will be used in a program, this entry is required.

BLKSIZE=n

This entry indicates the size of the I/O area by specifying the maximum number (n) of characters that will be transferred to, or from, the area at any one time. When undefined records are read or written, the area must be large enough to accommodate the largest record.

If key length is specified by DTFDA KEYLEN and if macro instructions that transfer the key areas of records will be issued, this area must provide for both the key area and data area of a record (see IOAREA1 and Figure 15). If a file is to be created or if records are to be added to a file, the count area of the records must be included in this specification.

IOCS uses this specification to construct the count field of the CCW for reading or writing fixed-length records.

CONTROL=YES

This entry must be included if a CNTRL macro instruction will be issued for this file. A control command issues orders to the disk drive to perform the non-data operation SEEK.

DEVICE=DISK11

This entry must be included to state that the logical file is on a 2311 disk drive.

NAME	OPERATION	OPERAND#	MUST BE INCLUDED	REMARKS *
Filename†	DTFDA†		Each file	Header Card. Specify symbolic filename.
		AFTER=YES	Record reference AFTER used for an output record	
		†BLKSIZE=n	Each file	Length of I/O area. n=maximum number of characters.
		CONTROL=YES	CNTRL macro used	
		†DEVICE=DISK11	Each file	
		†ERRBYTE=Name	Each file	Symbolic name of 2-byte field for error/status codes supplied by IOCS.
		IDLOC=Name	ID of same or next record to be supplied by IOCS	Symbolic name of 5-byte field for ID.
		†IOAREA1=Name	Each file	Symbolic name of input/output area. Same as used in DS.
		KEYARG=Name	Record reference by key	Symbolic name of key field.
		KEYLEN=n	Records contain key areas	All keys must be the same length. n=length of keys.
		LABADDR=Name	Check/write additional labels	Symbolic name of user's label routine.
		READID=YES	Record reference by ID used for an input record	
		READKEY=YES	Record reference by key used for an input record	
		RECFORM=FIXUNB	Fixed-length records	If this entry is omitted, IOCS assumes FIXUNB.
		UNDEF	Records not fixed-length, or Records added to a file and EOF record written	
		RECSIZE=n	Undefined records	n=number of a register 2-11.
		†SEEKADR=Name	Each file	Symbolic name of track-reference field. Field is 8 bytes long.
		SRCHM=YES	Search multiple tracks	Applies to record reference by key.
		†TYPEFLE=INPUT	Each file	Read and check standard labels.
		OUTPUT		Write standard labels.
		VERIFY=YES	Check record written on disk	
		WRITEID=YES	Record reference by ID used for an output record	
		WRITEKY=YES	Record reference by key used for an output record	
		XTNTXIT=Name		Symbolic name of user's extent routine.

† Must be included. Other entries are included when applicable.

When two choices are shown, select only the appropriate one and enter it after the = sign.

* The header card and each detail card except the last one used in a file set must contain a continuation punch in column 72. Each detail card except the last one used must also contain a comma after the last operand.

In all entries: Solid caps must be entered as shown (For example, AFTER=YES)

Lower-case letters are to be replaced by programmer's symbolic name or a number (For example, Filename in header card, or BLKSIZE=n where n is replaced).

n is a decimal self-defining value.

Figure 36. DTFDA Entries

ERRBYTE=Name

This entry is required for IOCS to supply indications of exceptional conditions to the problem program. The symbolic name of a 2-byte field, in which IOCS can store the error-condition or status codes, is entered in this card after the = sign.

The codes are available for testing by the problem program at WAITF time, after the transfer of a record has been completed. One or more of the following codes may be set by IOCS in the bits indicated:

Byte	Bit		Error/Status Code	Remarks
	Position	Hex Value		
0	0	80	---	---
	1	40	Wrong-length record	For fixed unblocked records, wrong blocksize specified while using READID, WRITEID, READKEY, and WRITEKY.
	2	20	---	---
	3	10	---	---
	4	08	No room found	Occurs only on WRITEAFTER. No room to write record on track specified.
	5	04	---	---
	6	02	---	---
	7	01	---	---
1	0	80	Data check in count area	Data error detected. Occurs when reading or searching a count field.
	1	40	---	---
	2	20	End of cylinder	Occurs when record is not found on specified cylinder while using SRCHM
	3	10	Data check when reading key or data	Data error detected when reading key or data, or while searching keys.
	4	08	No record found *)	Occurs when searching ID or key without SRCHM and ID or key is not found in specified track.
	5	04	End of file	Count field with data length of zeros detected.
	6	02	---	---
	7	01	---	---

*) If the Read ID command is issued for an ID one greater than the highest ID on a track, record 1 of that track will be read and no NRF condition is posted in the ERRBYTE. The user may use DTFDA parameter IDLOC=name, which supplies the ID of the next record to the problem program, to handle this specific type of NRF.

Note: Any bits that were set on by one I/O operation are reset before another I/O operation is started.

IDLOC=Name

This entry is included if the programmer wants IOCS to supply the ID of a record after each READ or WRITE instruction is executed for specified records. The symbolic name of a 5-byte field, in which IOCS is to store the ID, is specified after the = sign in this parameter.

IOCS supplies the ID of the record specified in the READ/WRITE instruction, or the ID of the next record on the track. This is affected by the type of record reference specified in the READ/WRITE instruction and by the use of the SRCHM specification to search multiple tracks for a particular key (Figure 37).

MACRO INSTRUCTION	ID SUPPLIED	
	With SRCHM	Without SRCHM
READ filename,KEY	Same record	Next record
READ filename,ID	(Invalid)	Next record
WRITE filename,KEY	Same record	Next record
WRITE filename,ID	(Invalid)	Next record
WRITE filename,RZERO	(Invalid)	None
WRITE filename,AFTER	(Invalid)	None

Figure 37. ID Supplied After a READ or WRITE Instruction

IOAREA1=Name

This entry must be included to specify the symbolic name of the input/output area used by this file. The input/output routines will transfer records to or from this area. The specified name must be the same as the name used in the DS instruction that reserves this area of main storage.

The main-storage input/output area must be large enough to contain the maximum number of bytes that will be required in any READ or WRITE instruction issued for this file in the problem program. This is affected by the length of record data areas, and by the use of the count and key areas as follows:

- If undefined records are specified in the DTFDA entry RECFORM, the area must provide for the largest data record that will be processed.
- If the DTFDA entry KEYLEN is specified and if any instructions that read or write the key area of a record are to be issued in the problem program, the input/output area must provide room for the key area as well as the data area. The length needed for the key is the length specified in KEYLEN.
- If any write instructions that transfer the count area to a disk record will be issued in the problem program, eight bytes of main storage must be allotted at the beginning of the I/O area. In these eight bytes IOCS will construct the count field to be transferred to disk.

Whenever a READ or WRITE instruction is issued, IOCS assumes that the input/output area (see Figure 15) contains the information implied by the type of instruction that is to be executed (Figure 38).

KEYARG=Name

This entry must be included if records are to be identified by key. That is, if the macro instruction READ Filename,KEY or WRITE Filename,KEY will be issued in the problem program, this card is required. KEYARG specifies the symbolic name of the key field in which the user will supply the record key for the READ/WRITE routines.

When record reference is by key, IOCS uses this specification at assembly time to construct the data address field of the CCW for search commands.

KEYLEN=n

This entry must be included if record reference is by key or if keys are to be read or written. It specifies the number (n) of bytes in each key. All keys must be the same length. If this entry is omitted, IOCS assumes a key length of zero.

When record reference is by key, IOCS uses this specification to construct the count field of the CCW for this file. IOCS also uses this in conjunction with IOAREA1 to determine where the data field in the main-storage I/O area is located (see IOAREA1).

LABADDR=Name

The user may require one or more labels in addition to the standard file label. If

so, he must include his own routine to check, or write, the additional labels. The symbolic name of his routine is specified in this entry. Programming branches to this routine after IOCS has processed the standard label.

At the end of his routine, the programmer must return to IOCS by use of the LBRET macro.

READID=YES

This entry must be included if any input records are to be specified by ID (identifier) in the problem program. That is, whenever the macro instruction READ filename,ID will be used in the program. this entry is required.

READKEY=YES

This entry must be included if any input records are to be specified by key in the problem program. That is, whenever the macro instruction READ filename,KEY will be used in the program, this card is required.

MACRO INSTRUCTION		I/O AREA CONTENTS	
		With KEYLEN	Without KEYLEN
READ	filename,KEY	Data	(Invalid)
READ	filename,ID	Key and Data	Data
WRITE	filename,KEY	Data	(Invalid)
WRITE	filename,ID	Key and Data	Data
WRITE	filename,AFTER	Count, Key, and Data	Count and Data

Figure 38. I/O Area Requirements for DAM

RECFORM=

This entry specifies the type of records in the input, or output, file. Either of the following specifications may be entered immediately after the = sign:

FIXUNB for fixed-length records. All records are considered unblocked in the DAM method. If the user wants blocked records, he must provide his own blocking and deblocking.

UNDEF for undefined records. This specification is required:

- Whenever records are not fixed-length.
- If records are to be added to a file (RZERO or AFTER used in a WRITE instruction) and an end-of-file record will be written.

RECSIZE=n

This entry must be included if undefined records are specified (RECFORM=UNDEF). It specifies the number (n) of the general-purpose register that will contain the length of each individual input or output record. This may be any register 2-11.

Whenever each undefined record is read, IOCS supplies the length of the data area of that record in the register.

When an undefined record is to be loaded, added, or written in the file, the programmer must load the length of the data area of the record (in bytes) into this register, before he issues the WRITE instruction for the record. IOCS adds the length of the key area when required.

When records are to be loaded or added to a file (RZERO or AFTER specified in the WRITE instruction), IOCS uses the length in constructing the count area to be written on disk.

SEEKADR=Name

This entry must be included to specify the symbolic name of the user's track-reference field. In this field the user stores the track location of the particular record to be read or written. The READ/WRITE routines refer to this field to determine which disk pack and which track on that pack contains the desired record. Whenever records are to be located by searching for a specified ID, the track reference field must also contain the number of the record on the track.

The track-reference field is an eight-byte field (MBBCCHHR), and the symbolic name labels the first byte (see Figure 16).

The bytes are used for:

M	Symbolic unit number
BB	Reserved for IBM 2321 Data Cell Drive (BB=00 for 2311 disk address)
CC	Cylinder number (the first C=0)
HH	Head number (the first H=0)
R	Record number

IOCS uses this specification to construct the data address field of the CCW for seek commands. When record reference is by ID, IOCS also uses this to construct the CCW data address field for search commands.

SRCHM=YES

If input/output records will be identified by key, this entry may be included to cause IOCS to search multiple tracks for each specified record. The instruction READ filename,KEY or WRITE filename,KEY will cause a search of the track specified in the track-reference field and all following tracks in the cylinder, until the record is found or the end of the cylinder is reached. If the logical file ends before the end of the cylinder and the record is not found, the search continues into the next file, if any, on the cylinder. If the record is not found on the cylinder, the end-of-cylinder bit in the error/status field is set on; the no-record-found bit is not affected.

Without this entry, each search is confined to the specified track. In this case, if the record is not found, the no-record-found bit in the error/status field is set on.

TYPEFLE=

This entry must be included to indicate how standard volume and file labels are to be processed:

INPUT	Standard labels are to be read and checked.
OUTPUT	Standard labels are to be written.

Because logical files on disk must always contain labels, this entry is always required.

VERIFY=YES

This entry is included if the user wants records to be checked after they are written on disk. If this entry is omitted, any records written on disk are not verified.

WRITEID=YES

This entry must be included if the disk storage location for writing any output record is to be specified by record ID (identifier) in the problem program. That is, whenever the macro instruction WRITE filename,ID will be used in the program, this entry is required.

WRITEKY=YES

This entry must be included if the disk location for writing any output record is to be specified by record key in the problem program. That is, whenever the macro instruction WRITE filename,KEY will be used in the program, this entry is required.

XTNTXIT=Name

This entry is included if the programmer wants to process XTENT card information. It specifies the symbolic name of his extent routine. Whenever XTNTXIT is included, IOCS branches to the user's routine during the initial OPEN for the file. It branches after each specified extent has been completely checked and conflicts, if any, have been resolved.

Upon entry to the user's routine, IOCS stores in Register 1 the address of a 12-byte area from which the user can retrieve extent card information (in binary form). This area contains:

Bytes	Contents
0-1	Symbolic unit (hexadecimal representation of SYSnnn) SYSRES = 0000 SYSRDR = 0004 SYSLST = 0008 SYSIPT = 000C SYSOPT = 0010 SYSLOG = 0014 SYS000 = 0018 SYS001 = 001C etc.
2	Extent type code (as specified in the XTENT card)

3	Extent sequence number (as specified in the XTENT card)
4-7	Lower limit of the extent (track address - CCHH)
8-11	Upper limit of the extent (track address - CCHH)

Also upon entry to the user's extent routine, IOCS stores a return address in Register 14. Therefore, at the end of his routine, the user must branch to this address to continue processing.

INDEXED SEQUENTIAL SYSTEM (DTFIS)

The DTFIS detail entries that apply to a file when records are processed by the Indexed Sequential File Management System are explained in the following text and shown in DTFIS Entries (Figure 39). Operand entries vary with the application, but name and operand must be included. Filenames should be unique to each other to be cataloged properly into the core image library. The length of the filename is limited to five characters.

ADAREX=Name

For each new record inserted within an organized file, a record is entered in an overflow area. If the specified overflow area(s) becomes filled and more records are yet to be stored, ISFMS branches to the user's routine specified by this entry. The symbolic name of the user's routine is entered after the = sign in this entry.

The user should specify an independent overflow area if one has not already been specified. Or, he should extend the size of the independent overflow area, if that is filled. For this, he supplies a new XTENT card for the independent overflow area and restarts the job at the point at which the overflow area was exceeded.

Note: If the CYLOFL option is taken, other additions may occur on cylinders where there is room for overflow records. The user, therefore, may desire to continue.

CYLOFL=n

This entry must be included if cylinder overflow areas are to be reserved for a logical file. A cylinder overflow area is located on each cylinder within the prime area of the data file. It contains records that overflow from tracks in that cylinder.

To reserve the areas for cylinder overflow this card is required when a file is to be loaded onto disk and when records are to be added to an organized file. It specifies the number (n) of tracks to be reserved on each cylinder.

If an independent overflow area is specified (by an XTENT card) along with the CYLOFL entry, overflow records are written in the independent overflow area after a cylinder overflow area becomes filled.

CYNDEX=Name

When a file is to be loaded onto disk, the user specifies the disk area to be used for the cylinder index (by including a Job Control XTENT card). If the index exceeds this area as the file is being loaded, ISFMS branches to the user's routine specified by this entry. The symbolic name of the user's routine is entered after the = sign in this entry.

The user must supply a new XTENT card for the cylinder index area and restart the job if the file is being loaded or extended.

DERREX=Name

This entry must be included to specify a user's routine for uncorrectable disk errors. If any uncorrectable error is detected when records are transferred either to or from disk storage, ISFMS branches to this routine. The symbolic name of the user's routine is entered after the = sign in this entry. The user has the choice of continuing, but he should save the I/O and work areas. During a LOAD function, the job is terminated automatically for all the count, key, and data WRITE operations. This routine will also be entered if a No-Record-Found condition occurs and no RTRVEX is specified.

NAME	OPERATION	OPERAND#	MUST BE INCLUDED	REMARKS*
Filename†	DTFIS†		Each file	Header card. Specify symbolic file name.
		ADAREX=Name	IOROUT specifies ADD or ADDRTR	Symbolic name of user's routine for full overflow areas.
		CYLOFL=n	Cylinder overflow areas	May be specified alone or with an independent overflow area. n = number of tracks for each area.
		CYNDEX=Name	IOROUT specifies LOAD	Symbolic name of user's routine for full cylinder index area.
		†DERREX=Name	Each file	Symbolic name of user's routine for any uncorrectable disk error.
		†DSKXTNT=n	Each file	Maximum number of extents specified for the file.
		DTAREX=Name	IOROUT specifies LOAD	Symbolic name of user's routine for full prime data area.
		DUPREX=Name	IOROUT specifies LOAD, ADD, or ADDRTR	Symbolic name of user's duplicate-record routine.
		EOFADDR=Name	Sequential processing	Symbolic name of user's end-of-file routine.
		ILIDEX=Name	TYPEFILE specifies SEQNTL or RANSEQ and SETL macro indicates ID	Symbolic name of user's routine for an ID outside the file limits.
		IOAREAL=Name	IOROUT specifies LOAD, ADD, ADDRTR	Symbolic name of input/output area. Same as used in DS. At least one I/O area must be specified for a file.
		IOAREAR=Name	TYPEFLE specifies RANDOM or RANSEQ	
		IOAREAS=Name	TYPEFLE specifies SEQNTL or RANSEQ	
		IOREG=n	Process records in I/O area	n = number of register 2-11
		†IOROUT=LOAD	Each file	Build or extend a file on disk.
		ADD		Insert new records in an organized file.
		RETRVE		Retrieve records for processing/updating.
		ADDRTR		Insert and retrieve records.
		KEYARG=Name	Retrieve records randomly, or sequentially starting by key	Symbolic name of key field in main storage.
		†KEYLEN=n	Each file	All keys must be the same length. n = length of key. (Maximum is 95)
		KEYLOC=n	Blocked records	n = high-order position of key field within each record
		MANDEX=Name	IOROUT=LOAD and MSTIND=YES	Symbolic name of user's routine for full master index area.
		MSTIND=YES	Master Index	
		†NRECDS=n	Each file	n = number of records in a block. For unblocked records, n = 1.
		†RECFORM=FIXUNB	Unblocked records	Applies to records in prime data area only.
		FIXBLK	Blocked records	
		†RECSIZE=n	Each file	n = number of characters in each logical record.
		RTRVEX=Name	If KEYARG is specified	Symbolic name of user's routine for records not found.
		SQCHEX=Name	IOROUT specifies LOAD	Symbolic name of user's sequence-error routine.
		TYPEFLE=RANDOM	IOROUT specifies RETRVE or ADDRTR	Random processing.
		SEQNTL		Sequential processing.
		RANSEQ		Random and sequential processing.

Figure 39. DTFIS Entries (Part 1 of 2)

NAME	OPERATION	OPERAND#	MUST BE INCLUDED	REMARKS*
		UPDATE=RANDOM SEQNTL RANSEQ	Update records	Random processing with updating. Sequential processing with updating. Random and sequential processing with updating.
		VERIFY=YES	Check records written on disk	
		†WLRERR=Name	Each file	Symbolic name of user's wrong-length-record routine.
		WORKL=Name	IOROUT specifies LOAD, ADD, or ADDRTR	
		WORKR=Name	TYPEFLE specifies RANDOM or RANSEQ and records are processed in a work area	
		WORKS=YES	TYPEFLE specifies SEQNTL or RANSEQ and records are processed in work areas	

† Must be included. Other entries are included when applicable.

When two choices are shown, select only the appropriate one and enter it after the = sign.

* The header card and each detail card except the last one used in a file set must contain a continuation punch in column 72. Each detail card except the last one used must also contain a comma after the last operand.

In all entries: Solid caps must be entered as shown (For example, IOROUT=LOAD).

Lower-case letters are to be replaced by programmer's symbolic name or a number (For example, Filename in header card, or CYLOFL=n where n is replaced).

n is a decimal self-defining value.

Figure 39. DTFIS Entries (Part 2 of 2)

DSKXTNT=n

This entry must be included to specify the maximum number (n) of extents for this file. The number must include all the prime data area extents (if more than one disk area is used for the data records), the master and cylinder index areas (treated as one extent), and the independent overflow area, all of which are specified by XTENT cards. Thus the minimum number specified by this entry is 2: one extent for one prime data area, and one for a cylinder index.

DTAREX=Name

When a file is to be loaded onto disk, the user specifies the disk area to be used for data records (by including a Job Control XTENT card). If the records require more than the allotted space as they are being loaded, ISFMS branches to the user's routine specified by this entry. The symbolic name of the user's routine is entered after the = sign in this entry. In the DTAREX routine, the user can issue an ENDFL instruction to prepare the organized file for closing. This permits the remaining records to be treated as extensions.

To continue loading the file, the user must supply a new XTENT card for the data area and restart the job at the point at which the data area was exceeded.

DUPREX=Name

When records are loaded or added to a file, ISFMS checks for duplicate record keys. If a duplication is found, ISFMS branches to the user's routine specified by this entry. The symbolic name of the user's routine is entered after the = sign in this entry. The user need not terminate the job. He may continue if he desires.

EOFADDR=Name

This entry must be included when a sequential retrieval operation is to be performed. It specifies the symbolic name of the user's end-of-file routine. ISFMS will branch to this routine when the end-of-file disk record is read. In his routine the user may perform any operations required for the end of the job, and he generally issues the CLOSE instruction for the file.

ILIDEX=Name

This entry is included when records are to be retrieved in sequential order and retrieval is to start a specified ID. ILIDEX gives the symbolic name of the user's routine to which ISFMS will branch if an ID that is outside of the file limits (specified by XTENT cards) is supplied in the SETL idname field. Another SETL may then be issued with a new ID. An ESETL is not required in this routine.

IOAREAL=Name

This entry must be included when a file is created (loaded) or when records are added to an organized file. It specifies the symbolic name of the output area used for loading or adding records to the file. The specified name must be the same as the name used in the DS instruction that reserves this area of main storage. The ISFMS routines construct the contents of this area and transfer records from this area to disk storage.

This main-storage output area must be large enough to contain the count area, key area, and data area of records. Furthermore, the data-area portion must provide enough space for the sequence-link field of overflow records whenever records are added to a file (Figure 40).

IOAREAR=Name

This entry must be included whenever records are processed in random order. It specifies the symbolic name of the input/output area used for random retrieval (and updating). The specified name must be the same as the name used in the DS instruction that reserves this area of main storage.

This main-storage I/O (Figure 41) must be large enough to contain:

- the data area of an unblocked record read from the prime data area,
- the data area of a block of records read from the prime data area, or
- the key area and the data area, including the sequence-link field, of a record read from an overflow area. This applies regardless of whether the logical file contains blocked or unblocked records, because all overflow records are unblocked.

IOAREAS=Name

This entry must be included whenever records are processed in sequential order by key. It specifies the symbolic name of the input/output area used for sequential retrieval (and updating). The specified name must be the same as the name used in the DS instruction that reserves this area of main storage.

This main-storage I/O area (Figure 41) must be large enough to contain:

- the key and data areas of an unblocked record read from the prime data area,
- the data area of a block of records read from the prime data area, or
- the key area and the data area, including the sequence-link field, of a record read from an overflow area. This applies regardless of whether the logical field contains blocked or unblocked records, because all overflow records are unblocked.

IOREG=n

This entry must be included whenever records (blocked or unblocked) are to be retrieved and processed directly in the I/O area. It specifies the number (n) of the register that ISFMS can use to indicate the address of the data portion of the record that is available for processing. ISFMS puts this address in this register each time a READ, WRITE, GET, or PUT is executed. Any register 2-11 may be specified.

Note: For sequential unblocked records, the key is at the beginning of the I/O area.

FUNCTION	OUTPUT AREA REQUIREMENTS (IN BYTES)			
	Count	Key	Sequence Link	Data
Load Unblocked Records	8	Key Length	—	Record Length
Load Blocked Records	8	Key Length	—	Record Length x Blocking Factor
Add Unblocked Records	8	Key Length	10	Record Length
Add Blocked Records	8	Key Length	— OR*	Record Length x Blocking Factor
	8	Key Length	10	Record Length
* Whichever Is Larger				

Figure 40. Output Area Requirements for Loading or Adding Records to a File by ISFMS

FUNCTION	I/O AREA REQUIREMENTS (IN BYTES)			
	Count	Key	Sequence Link	Data
Retrieve Unblocked Records	—	Key Length	10	Record Length
Retrieve Blocked Records	—	—	— OR*	Record Length x Blocking Factor
	—	Key Length	10	Record Length
* Whichever Is Larger				

Figure 41. I/O Area Requirements for Random or Sequential Retrieval by ISFMS

Whenever IOREG is specified for a file, the program should not include both DTFIS entries WORKR and WORKS for that file.

IOROUT=

This entry must be included to specify the type of function to be performed. One of the following specifications is entered after the = sign.

LOAD — to build a logical file on disk or to extend a file beyond the highest record presently in an organized file.

ADD — to insert new records into an organized file.

RETRVE — to retrieve records from a file for either random or sequential processing and/or updating.

ADDRTR — to both insert new records into a file (ADD) and retrieve records for processing and/or updating (RTR).

In using ADD and ADDRTR, it is necessary to guard against losing records, should the problem program not reach normal end of job after the instructions are issued. The CLOSE macro updates the Format-2 label with the address of the last record written in the current overflow area. If the program aborts, CLOSE is not issued and updating will not occur. Future ADDS will destroy the sequential chain of the file and records may be lost.

KEYARG=Name

This entry must be included for random retrieval (READ) or sequential retrieval (GET) beginning with key. It specifies the symbolic name of the main-storage key field in which the user must supply the record key to ISFMS. Whenever this entry is included, the DTFIS entry RTRVEX must also be included.

KEYLEN=n

This entry must be included to specify the number (n) of bytes in the record key. All keys must be the same length (maximum length is 95 bytes).

KEYLOC=n

This entry must be included if blocked records are specified in DTFIS RECFORM. It supplies ISFMS with the high-order position of the key field within the data record. That is, if the key is recorded in positions 21-25 of each record in the file, this card specifies 21.

ISFMS uses this specification to locate (by key) a specified record within a block. The key area of a disk record contains the key of the highest record in the block. To search for any other record, ISFMS locates the proper block and then examines the key field within each record in the block.

MANDEX=Name

When a logical file is to be loaded onto disk, the user may request that the ISFMS build a master index (by specifying MSTIND=YES). If so, the area reserved for the master index is also specified by a Job Control XTENT card. If the master index exceeds the allotted area as the file is being loaded, ISFMS branches to the user's routine specified by this entry. The symbolic name of the user's routine is entered after the = sign in this entry.

The user must supply a new XTENT card for the master index area and restart the job if the file is being loaded. This may also necessitate a new XTENT card for the cylinder index area. If the file is being extended, the user must reorganize the file.

MSTIND=YES

This entry is included whenever a master index is used for a file. In this case, it is required when a file is loaded (to instruct ISFMS to build the index) and when records are added to or retrieved from a file with a master index.

ISFMS always builds a track index and a cylinder index, but the master index is optional. The master index, if used, is the highest level index, and it includes an index record for each track of the cylinder index. Thus, it points to the cylinder index on a search for a particular record (see Indices: Master Index). The location of the master index is specified by a Job Control XTENT card.

NRECDS=n

This entry is always required. It specifies the number (n) of logical records in a block (called the blocking factor). The maximum number of records (n) that may be specified is 255. If unblocked records are specified (RECFORM=FIXUNB), this card must specify "1".

RECFORM=

This entry specifies the type of records in the logical file. All logical records in the file must be fixed length. However, they may be either blocked or unblocked. One or the other of these specifications must be entered after the = sign:

FIXUNB	for unblocked records.
FIXBLK	for blocked records. With this specification the key of the highest record in the block becomes the key for the block and must be recorded in the key area.

The specification that is included when the logical file is loaded into disk storage must also be included whenever the file is processed.

Records in the overflow area(s) are always unblocked (see Addition of Records, and Overflow Areas), but that does not affect this entry. RECFORM refers to records in the prime data area only.

RECSIZE=n

This entry must be included to specify the number(n) of characters in a logical record. This is the length of the data area of each individual record. All logical records must be the same size.

RTRVEX=Name

If a specified record cannot be found in a retrieve function, ISFMS branches to the user's routine specified in this entry. The symbolic name of the user's routine is entered after the = sign in this entry. The job need not be terminated. If this occurred during a SETL, an ESETL must be issued prior to the next SETL. This entry must be included whenever DTFIS KEYARG is specified.

SQCHEX=Name

When records are loaded onto disk, ISFMS checks the records for sequential order by key. If a sequence error is detected, ISFMS branches to the user's routine specified by this entry. The symbolic name of the user's routine is entered after the = sign in this entry. The user may do whatever he wishes with the record containing the error without terminating the loading process.

TYPEFLE=

This entry must be included when a retrieval function is to be performed. It specifies the type(s) of processing that is to be performed by the problem program for this file. One of the following specifications is entered after the = sign:

RANDOM for random processing. Records are retrieved from the file in random order specified by key. Only READ instructions may be issued to transfer records to main storage.

SEQNTL for sequential processing. The problem program specifies the first record to be retrieved, and thereafter ISFMS retrieves records in sequential order by key. The first record is specified by key, ID, or the beginning of the logical file (see SETL Macro). Only GET instructions may be issued to transfer records to main storage.

RANSEQ for both random and sequential processing. READ and/or GET instructions may be issued to transfer records.

TYPEFLE is not required for loading or adding functions.

UPDATE=

This entry must be included if disk records are to be updated. That is, records are to be retrieved, processed, and then transferred back (WRITE or PUT) to the same disk records from which they were read. One of the following specifications is entered after the = sign:

RANDOM for random processing with updating. Records (retrieved by READ instructions) are written back into the file by WRITE instructions. RANDOM may be specified only if TYPEFLE specifies RANDOM or RANSEQ.

SEQNTL for sequential processing with updating. Records (retrieved by GET instructions) are written back into the file by PUT instructions (GET instructions for blocked records). See Macro Instructions for Sequential Retrieval by ISFMS. SEQNTL may be specified only if TYPEFLE specifies SEQNTL or RANSEQ.

RANSEQ for random or sequential retrieval with updating. Records (retrieved by READ or GET instructions) may be written back into the file by WRITE or PUT instructions. RANSEQ may be specified only if TYPEFLE specifies RANSEQ.

VERIFY=YES

This entry is included if the user wants records to be checked after they are written on disk. If this entry is omitted, any records written on disk are not verified.

WLRERR=Name

This entry must be included to specify the user's routine for wrong-length records. If a wrong-length record is detected when data is transferred to or from disk storage, ISFMS branches to this routine. The symbolic name of the user's routine is entered after the = sign in this entry.

Note: An erroneous specification to ISFMS in RECSIZE, KEYLEN, etc., may cause a wrong-length record.

WORKL=Name

This entry must be included whenever a file is to be created (loaded) or records are to be added to an organized file. It specifies the symbolic name of the work area in which the user must supply the data records to ISFMS for loading or adding to the file. The specified name must be the same as the name used in the DS instruction that reserves this area of main storage.

This work area must provide space for both the logical data record (data area) and the record key (for the key area of the disk record) when a file of unblocked records is created or when records are added to any file. To create a file of blocked records, however, the work area must provide space for data only.

Due to record shifting in an ADD function, the original contents of WORKL will be changed.

WORKR=Name

When records are processed in random order, this entry must be included if the individual records are to be processed in a work area rather than the I/O area. It specifies the symbolic name of the work area. This name must be the same as the name used in the DS instruction that reserves this area of main storage. This area must provide space for one logical record (data area).

When this entry is included and a READ or WRITE instruction is executed, ISFMS moves the individual record to, or from, this area. Whenever this entry is included for a file, IOREG should not be specified.

WORKS=YES

When records are processed in sequential order, this entry must be included if the individual records are to be processed in work areas rather than the I/O area. Each GET or PUT instruction must specify the symbolic name of the work area to, or from, which ISFMS is to move the record. The area must be large enough for one logical record (data area) and the record key (key area) for unblocked records. For blocked records, it must be large enough for data only. Whenever this entry is included for a file, IOREG should not be specified.

PROCESSING WITH STR DEVICES (DTFSN, DTFRF)

When STR macro instructions (READ, WRITE, etc.) are used in a program, the definition macros DTFSN and DTFRF must be used. Where other DTF macros are not used, the DTFRF and DTFSN macros for STR support should appear first in the problem program. When the problem program uses other DTF macros, the DTFRF should follow the DTFEN card.

DTFSN Macro

Name	Op	Operand
dtfname	DTFSN	DEVADDR=SYSnnn
		AREA=abuck
		LENGTH=lbuck

The DTFSN macro generates the STR channel command block (CCB), channel command word (CCW) lists, and linkages to the READ, WRITE, and CNTRL routines for each line.

The file definition macro DTFSN is coded as a keyword macro with three operands:

DEVADDR=SYSnnn is the symbolic unit address.

AREA=abuck, where abuck is a symbolic name, unique to this DTFSN, for a full word in the generated STR CCB (Figure 42). Before every READ or WRITE to this line, abuck must be loaded with the starting address of the data area.

LENGTH=lbuck, where lbuck is a symbolic name, unique to this DTFSN, for a half word in the generated STR CCB. Before every READ or WRITE to this line, lbuck must be loaded with the length of the data.

During processing with STR devices, bits are set in STR CCB byte 12 to indicate the current status of the file. Figure 43 shows the conditions indicated by byte 12.

In addition, each CCB generated by the DTFSN macro is preceded by a doubleword "user area", which the problem program may use for any purpose. For example, this area would be useful in providing information for multiline control during STR processing.

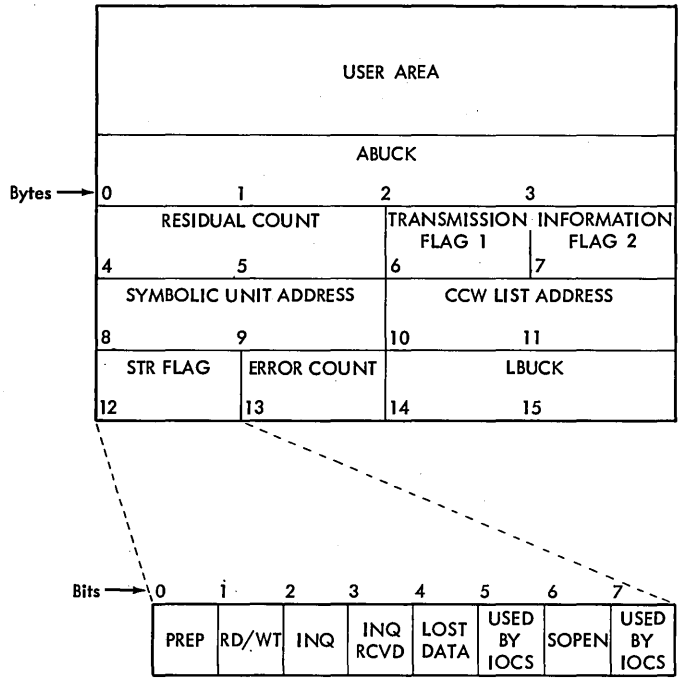


Figure 42. Expanded STR Channel Command Block (CCB)

BYTE	BIT	CONDITION INDICATED	
		1 (ON)	0 (OFF)
12	0 - PREP	CNTRL PREP issued	PREP not issued
	1 - READ/WRITE	READ or WRITE issued	READ or WRITE not issued
	2 - INQUIRY	CNTRL INQ issued	INQ not issued
	* 3 - INQ RCVD	INQUIRY received	---
	* 4 - LOST DATA	Abnormal end of message (operator reply 4) was specified. EOF bit in transmission flag also will be on. Depending on the instructions being executed, indicates: (1) SOPEN could not establish synchronism, or (2) lost or duplicate record during a READ.	---
	5 - Used By IOCS	---	---
	6 - SOPEN	SOPEN issued	SOPEN not issued
	7 - Used By IOCS	---	---

* To be tested by problem program

Figure 43. Conditions Indicated by Expanded STR CCB, Byte 12

DTFRF Macro

Name	Op	Operand
endop	DTFRF	startaddr, reg

The DTFRF macro instruction is used to establish a base register which the problem program must use to reference the expanded STR channel command block (CCB).

The name field (endop) must contain the symbolic operand that is specified in the problem program's END card.

In the operand field, "startaddr" must be the symbolic address of the first executable instruction of the problem program. "reg" is a general register (2-11) to be used as a base register. This register must not be used for any other purpose while it is being used for the DTFRF.

This macro must precede the first DTFSN macro definition statement.

These definition type macros (DTFSN and DTFRF) should precede the problem program and reside in the first 16K of main storage.

BINARY SYNCHRONOUS COMMUNICATION (DTFBS, DTFRF)

When BSC support macros (READ, WRITE, etc.) are used in a program, the file-definition macros DTFBS and DTFRF must be used. The DTFRF and DTFBS macros should appear before the first executable instruction of the problem program. Where other DTF macros are used, the DTFRF and DTFBS should follow the DTFEN card.

DTFBS Macro

Name	Op	Operand
	DTFBS	DEVADDR=SYSnnn,
		AREA=name,
		LENGTH=name,
		BSCFLAG=name,
		RCOUNT=n

The DTFBS macro generates the expanded BSC command control block (CCB), linkages to the READ, WRITE, and CNTRL routines for the line and the channel programs (channel command word lists) used by these routines. It also generates a table of EBCDIC special characters containing the BSC control characters.

The BSC file definition macro, DTFBS, is coded as a keyword macro with five operands:

DEVADDR=SYSnnn is the symbolic unit address of the IBM 2701, SDA II.

AREA=name is the symbolic name to be assigned to the full word area address field in the expanded BSC CCB (Figure 44).

LENGTH=name is the symbolic name to be assigned to the half-word length field in the BSC CCB (Figure 44).

BSCFLAG=name is the symbolic name to be associated with the first BSC flag byte. The problem program may use this optional name when referencing the flag bytes. (See Figure 44).

RCOUNT=n is a count of the number of times operations ending in I/O errors are to be retried before returning to the problem program or issuing an operator's message. The maximum retry count that can be specified is 15. The suggested retry count is either three or seven with a higher retry count being recommended for higher speed lines. RCOUNT=3 is assumed if the RCOUNT parameter is omitted.

The DTFBS macro generates an EBCDIC special character table containing the BSC control characters. This table may be referenced by the name IOBSCT. The table contains only the hexadecimal control characters (no other attributes). The characters are in the same order as in Appendix M, Part 1.

During processing using BSC support macros, bits are set in the BSC flag bytes (bytes 16-19 of the CCB) and in the transmission flags (byte 7 of the CCB) to indicate the current status of the file. Figure 45 shows the conditions indicated by these bytes. I/O operations ending in errors are retried by BOS/BSC support up to the maximum count (RCOUNT=n) in an attempt to get the correct response or to read or write a message.

If the retry count is exceeded, the following actions are taken by BSC Support error routines:

- A three-part PIOCS message is issued with reply requested for the command reject, equipment check, data check, and overrun error conditions.
- An operator awareness message is issued for the lost data, bus-out check, intervention required, and time out error conditions. These are also posted to the CCB (transmission flags).

  Operator awareness messages are also issued and the job terminated for errors occuring on I/O operations initiated by the BSC error routines and for errors which are meaningless, or not defined for the I/O operation.
- Other errors are posted to the expanded CCB (BSC flag bytes).

For detailed information on the messages issued by BSC Support, refer to IBM System/360 Basic Operating System, Operator Messages (Form C24-5024).

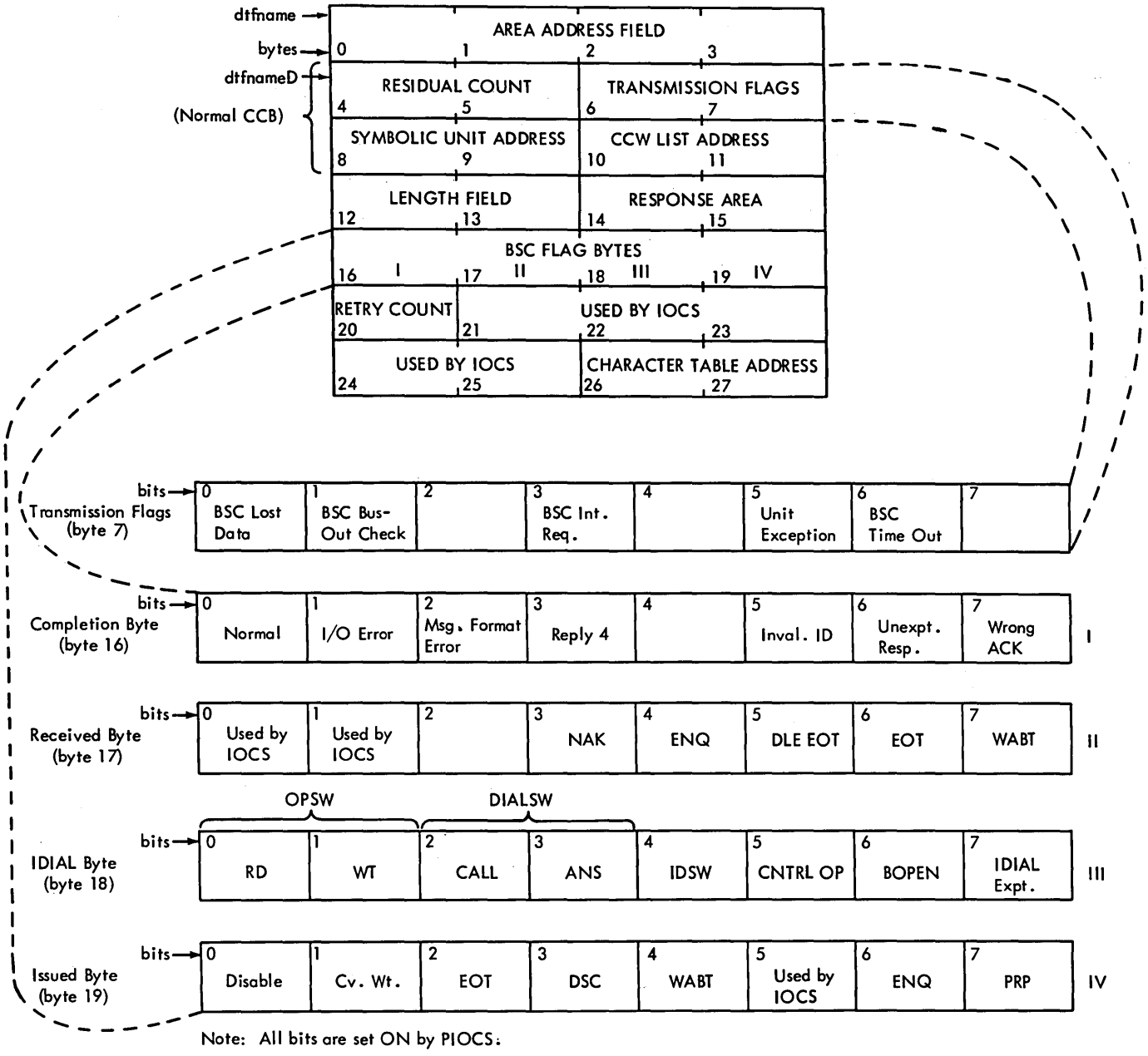


Figure 44. Expanded BSC Command Control Block (CCB)

BYTE	BIT	CONDITION INDICATED	
		1 (ON)	0 (OFF)
16	*0	Normal I/O operation completion	I/O operation did not reach normal completion
	*1	I/O error detected (see Appendix M, Part 2)	
	*2	Message format error detected (see Appendix M, Part 2)	No message format errors detected
	3	Operator reply 4 specified	
	4	Not used	
	*5	Invalid ID-sequence received	
	*6	Unexpected response received	
	*7	Wrong alternating acknowledgment (ACK) received	
17	0	Used by IOCS	
	1	Used by IOCS	
	2	Not used	
	*3	NAK control character received in response to ENQ	
	*4	ENQ control character received instead of a message or ENQ control character received in response to ENQ (contention for the line)	
	*5	Disconnect sequence (DLE EOT) received	
	*6	EOT (end of transmission signal received)	
	*7	WABT sequence received	
18	0,1	OPSW--set by IDIAL 11--WTX (transparent WRITE) 10--RD (READ) 01--WT (WRITE)	
	2,3	DIALSW--set by IDIAL 00--NO (leased line) 01--ANS (answer) 10--CALL (calling) 11-MAN (manual)	
	4	ID-verification to be included	No ID-verification to be included
	5	CNTRL operation being initiated	
	6	BOPEN issued	BOPEN not issued
	7	IDIAL macro should be coded following BOPEN (DIAL=YES)	IDIAL has been completed if required, or IDIAL is not necessary
19	0	Disable command issued	
	1	Conversational WRITE issued	
	2	CNTRL EOT issued	
	3	CNTRL Disconnect (DSC) issued	
	4	CNTRL WABT issued	
	5	Used by IOCS	
	6	CNTRL ENQ issued **	ENQ not issued
	7	CNTRL Prepare (PRP) issued	PRP not issued
* Should be checked by the user (see Appendix M, Part 2). ** Set with PRP issued for conversational mode.			

Figure 45. Conditions Indicated by BSC Flags, Bytes 16-19 of CCB

Before any READ, WRITE or IDIAL macro (except READ ENQ-type TQ), the area address field and length field must be properly set up. Normally, the two fields contain the starting address and the length of the data area, respectively.

For a conversational WRITE (type TC or TV), the length field should contain X'FFFF'. The area address field should point to a parameter list that contains the starting addresses and lengths of the WRITE data area and the READ data area.

For a READ with leading graphics (type TG or TL), the length field should contain the length of the READ data area. The area address field should point to a parameter list that contains the starting address of the READ data area and up to seven graphic characters (the first of which is preceded by a binary count).

DTFRF Macro

Name	Op	Operand
endop	DTFRF	startaddr,reg

The DTFRF macro instruction is used to establish a base register for referencing information stored in a BSC CCB (command control block). If this macro is used, the base register specified by "reg" must be maintained or must be reloaded with the proper address before referencing a BSC CCB or executing any BSC macro instruction. It should be coded after the DTFEN card and before the DTFBS macro.

The name field (endop) must contain the symbolic operand specified on the problem program END card.

In the operand field, "startaddr" must be the symbolic address of the first executable instruction of the problem program. "Reg" is the general register (2-11) to be used as a base register.

These file definition macros (DTFBS and DTFRF) should precede the problem program and reside in the first 16K of main storage.

NAME	OPERATION	OPERAND	APPLIES TO								MUST BE INCLUDED	REMARKS
			2311 DISK DRIVE	2400 MAGNETIC TAPE UNIT	1442/2501/2520 2540 READER	1442/2520/2540 PUNCH	1403/1404/1443/ 1445 PRINTER	1052 PRINTER-KEYBOARD	2671 PAPER TAPE READER	1285 OPTICAL READER		
Filename	DTFPH		X	X							Labelled Disk/Tape File	Symbolic File Name
		DEVADDR=SYSnnn		X							Labelled Tape File	Symbolic Unit for the Device Used for the File.
		LABADDR=Name	X	X							Check/Build Additional User-Standard Labels	Symbolic Name of User's Label Routine. For Tape Input and Disk Files, Applies to Header Labels Only.
		MOUNTD=ALL SINGLE	X								Each Disk File	All Extents Are to Be Available at the Initial OPEN. Only the First Extent Is to Be Available at the Initial OPEN.
		TYPEFLE=INPUT OUTPUT	X	X							Labelled Disk/Tape Input File Labelled Disk/Tape Output File	
		XTNTXIT=Name	X									Symbolic Name of User's Extent Routine.

Figure 46. DTFPH Entries

PHYSICAL IOCS (DTFPH)

When physical IOCS macro instructions (EXCP, WAIT, etc) are used in a program, only disk or tape files with standard labels need to be defined by DTFPH entries (DTF for a file handled by physical IOCS). No other files require definition.

If a disk or tape file with standard volume and file labels is processed, a DTFPH header card and five detail cards may be used (Figure 46). This 6-card set indicates to IOCS that labels are to be read and checked (on input) or written (on output). The header card is punched with DTFPH in the operation field, the symbolic name of the file in the name field, blank in the operand field, and a continuation punch in column 72. The symbolic name may be seven characters long.

The keyword parameters may be specified in one or more detail cards (see File Definition Macros: Logical IOCS), or in the header card if desired. Specifying the parameters in the header card eliminates the need for the detail card(s) and the continuation punch in the header card.

DEVADDR=SYSnnn

This entry specifies the symbolic unit (SYSnnn) to be associated with this logical file. The symbolic unit represents an actual I/O device address. The symbolic unit may be:

SYSIPT for main system input device

SYSOPT for main system output device

SYS000-SYS254 for other devices in the system

The symbolic unit (SYSnnn) is used in the Job Control ASSGN card (or in the Supervisor-Assembly macro SYMUN) to assign the actual I/O device address to this file. If the ASSGN card is used to assign the device for program execution, the file of records can be read from or written on different units at different times. For

example, a reel of tape may be mounted on any tape drive that is available at the time the job is ready to be run, by merely assigning that drive to the symbolic unit.

LABADDR=Name

The user may require one or more disk, or tape, labels in addition to the standard file labels. If so, he must include his own routine to check (on input) or build (on output) his label(s). He specifies the symbolic name of his routine in this entry, and IOCS branches to his routine after the standard label has been processed.

LABADDR may be included to specify a user routine for header or trailer labels as follows:

- Disk input or output file: header labels only
- Tape input file: header labels only
- Tape output file: header and trailer labels

Thus, if LABADDR is specified for the file, user-standard header labels can be processed for an input/output disk or tape file, and user-standard trailer labels can be built for a tape output file. Similar to the functions performed by logical IOCS, physical IOCS reads input labels and makes them available to the user for checking, and writes output labels after they are built.

LABADDR allows one user's label routine to be specified for all types of labels for the file: header labels, end-of-file labels, and end-of-volume labels. On an input file, the user can determine the type of label that has been read by the identification in the label itself. For an output tape file, however, IOCS indicates to the user the type of label that is to be written. For this, IOCS supplies a code in the low-order byte of Register 0, as follows:

O - Header label (letter O)
F - End-of-file label
V - End-of-volume label

In his routine the user can test this byte and then build the appropriate type of label.

Logical IOCS uses general registers 14 and 15 for linkage to and from a called routine; therefore, these registers must not be destroyed. If they are required by the user's label routine, they must first be saved and then restored prior to returning to the mainline of the program.

At the end of his label routine, the programmer must return to IOCS by use of the LBRET macro.

MOUNTD=

This entry must be included for a disk file to specify how many extents (disk areas) for the file are to be made available for processing when the file is initially opened. One of the following specifications is entered after the = sign:

ALL — if all extents on all packs are to be available for processing. When the file is opened, IOCS checks all labels on all packs and makes available all extents specified by the user's control cards. Only one OPEN is required for the file. ALL should be specified whenever the user plans to process records in a manner similar to that performed by the direct access method or the indexed sequential system.

SINGLE — if only the first extent on the first pack is to be available for processing. IOCS checks the labels on the first pack and makes the first extent specified by the user's control cards available for processing. The user must keep track of the extents and issue a subsequent OPEN whenever another extent is required for processing. On each OPEN after the first, IOCS makes available the next extent specified by the control cards. SINGLE should be specified when the user plans to process records in consecutive order.

When the user issues a CLOSE for an output file, the pack on which he is currently writing records will be indicated, in the file label, as the last volume for this file.

This entry must not be included for a tape file.

TYPEFLE=

This entry must be included to specify the type of file (input or output). One specification or the other is entered immediately after the = sign:

INPUT for a disk or tape input file

OUTPUT for a disk or tape output file

XTNTXIT=Name

This entry is included if the programmer wants to process XTENT card information. It specifies the symbolic name of the user's extent routine.

Whenever XTNTXIT is included, IOCS branches to the user's routine during an OPEN for the file. If the DTFPH entry MOUNTD=ALL is also specified for the file, IOCS branches during the initial OPEN. It branches after each specified extent has been completely checking and after conflicts, if any, have been resolved. If MOUNTD=SINGLE is specified for the file, IOCS branches to the user's routine whenever an individual extent is opened. This includes both the initial OPEN for the first extent and each succeeding OPEN for additional extents. It branches after checking the extent.

Upon entry to the user's routine, IOCS stores in Register 1 the address of an 12-byte area from which the user can retrieve extent card information (in binary form). This area contains:

Bytes	Contents
0 - 1	Symbolic unit (hexadecimal representation of SYSnnn) SYSRES = 0000 SYSRDR = 0004 SYSLST = 0008 SYSIPT = 000C SYSOPT = 0010 SYSLOG = 0014 SYS000 = 0018 SYS001 = 001C etc.
2	Extent type code (as specified in the XTENT card)
3	Extent sequence number (as specified in the XTENT card)
4 - 7	Lower limit of the extent (track address - CCHH)
8 - 11	Upper limit of the extent (track address - CCHH)

Also upon entry to the user's extent routine, IOCS stores a return address in Register 14. Therefore at the end of his routine the user must branch to this address to continue processing.

SUPERVISOR-COMMUNICATION MACROS

During problem program execution a Supervisor is located in lower main storage , and provides over-all control of operations. The functions performed by the Supervisor are interruption handling, I/O requests, and program retrieval. The interruptions handled by the Supervisor result from five basic types of conditions:

Input/Output	Conditions in the channel and/or I/O devices, such as the completion of an I/O operation.
Program Check	Improper specification or use of instructions and data.
Machine Check	Machine malfunction.
External Signal	Signal from interruption key (operator) or timer.
Supervisor Call	Execution of a Supervisor Call instruction to make use of a Supervisor function or to convey a message from the calling program to the Supervisor.

The Supervisor also contains a communication region that can be used for storing information useful to several problem programs or between problem programs.

The region provides for:

- A user area for inter- or intra-program communication.
- Address of user-supplied routines for program check, timer interruption, and operator communication to the problem program.
- End-of-Supervisor address.
- User program switches.
- Machine configuration.
- Date.
- Job name.

Several macro instructions are available to the programmer to enable him to communicate with the Supervisor. Thus he can utilize the functions performed by the Supervisor or have access to the communication region in the Supervisor. To make use of the Supervisor functions requires switching from problem state to Supervisor state. Therefore the macros

used for this purpose generate a Supervisor Call (SVC) instruction and a specific code to indicate the desired routine. These macros are:

EXCP	Execute channel program.
FETCH	Load a program or a program phase.
EXIT	Return from the timer or operator-communication routine to the problem program main line.
MSG	Transmit a message from the problem program to the operator via the Supervisor.

The macros that provide access to the communication region are:

COMRG	To obtain the address of the communication region.
MVCOM	to move information to the communication region for modifying any portion of it.
STXIT	To place the address of a user's interruption routine in the communication region, or to change an interruption option.

Two other macro instructions are provided:

DUMP	Informs the Supervisor that the problem program has reached an end-of-job condition and that a storage print-out is desired.
EOJ	(End of job). Informs the Supervisor that the problem program has been completed.

EXCP Macro

Name	Op	Operand
	EXCP	blockname

This macro instruction (execute channel program) is issued to request that an I/O operation be performed. It causes physical IOCS (Channel Scheduler) to perform a start I/O operation.

EXCP is used for a file of data that is controlled by physical IOCS macro instructions and it is described in the section Processing Records with Physical IOCS.

FETCH Macro

Name	Op	Operand
	FETCH	name

The FETCH macro instruction is issued whenever the program requires that another phase of the program, or a different program, be loaded for execution. The name of the phase or program is specified in the operand field, and it may be 1-6 characters long.

Execution: Disk-Resident System

When the FETCH macro instruction is executed in a disk-resident system, control is turned over to the System Loader (program-fetch routine), which is one of the Supervisor routines. The System Loader then retrieves the specified program phase, program routine, or separate program, from the core image library and loads it into main storage. The System Loader determines the proper phase (or program) to be retrieved by matching the name specified in the operand of the FETCH macro with the names in the directories of the core image library. After the phase is loaded, control is transferred to it at the address originally specified when the phase was stored in the core image library. If the phase requested cannot be found, a program check occurs (see Program Check).

The program is loaded from the core image library, without relocation. Therefore it is the user's responsibility to ensure that the phase or program to be loaded does not contain assembled addresses that will load text into the main-storage area occupied by the Supervisor or by any portion of his program that will be required later. Also, the user must make provision for returning to his original program if that is required.

Registers 12 and 13 are used by the Supervisor, and their contents are not saved or restored, unless SUPVR SAVEREG=YES was specified when the supervisor was assembled. (See Macro Instruction to Assemble a Supervisor.)

Additional information about libraries and the System Loader is given in the BOS Programmer's Guide, as listed on the front cover of this publication.

Execution: Not a Disk-Resident System

If the disk-resident system is not used for execution of the object program, control is turned over to the Program Loader when the FETCH is executed. The Program loader must

be in main storage at this time, and it loads the next phase or program that is presently available in the system input device (designated by SYSIPT). The system input device may be a card reader or a tape unit. Loading continues until the loader recognizes an XFR card or an Assembler END card with an operand, at which time control is transferred to the problem program. The XFR or END card contains the address at which execution of the program is to start.

The program is loaded from TXT or REP cards, without relocation. Therefore, it is the user's responsibility to ensure that the phase or program to be loaded does not contain assembled addresses that will load text intc the main-storage area occupied by the Supervisor, Program Loader, or any portion of his program that will be required later. Also the user must make provision for returning to his original program, if that is required, because the loader does not save and restore registers. Registers 1, 14, and 15 are destroyed by the loader. Registers 12 and 13 are used by the Supervisor, and may be used by the programmer if SUPVR SAVEREG=YES was specified when the supervisor was assembled. (See Macro Instructicn to Assemble a Supervisor.)

Text and transfer cards (TXT and XFR or END) are output from the Assembler and input to the Program Loader. Replacement cards (REP) can be added to the object deck and used as input to the Program Loader.

Additional information about the Prcgram Loader and TXT, REP, XFR, and END cards is given in the BPS Programmer's Guide.

Fetch Macro Source Statements

Issuing the FETCH macro instruction in the source program causes several statements to be assembled, including the following:

Statement		Meaning
FETCH	PHASE2	Macro instruction
.		
.		
.		
SVC		Supervisor Call
DC	CL6'PHASE2'	6-byte phase or program name

During execution, the Supervisor Call interruption routine analyzes the code and gives control to the appropriate loader (system or program), depending on whether the program is executed in a disk system.

EXIT Macro

Name	Op	Operand
	EXIT	TR
	EXIT	CR

The EXIT macro instruction is issued at the end of either the user's time-interruption routine or the user's operator communication routine, to return to the main line of the problem program at the point where the interruption occurred. This instruction requires one of the following operands:

TR To return from the user's timer routine.

CR To return from the user's operator-communication routine.

MSG Macro

Name	Op	Operand
Name	MSG	code,REPLY

The MSG (message) macro instruction allows a user's problem program or an IBM-supplied program to give a message to the operator. It is used to indicate error conditions and/or to request an operator decision before proceeding further with processing of the problem program.

This instruction can be labeled with a symbolic name and it can have one or two operands. The first operand (code) is the message to be communicated to the operator. It may contain 1-4 characters, which are generally a code or some meaningful characters such as ISEQ for input-sequence-error.

The second operand (REPLY) is optional, and it is included only if further processing is dependent upon a response from the operator. For example, if ISEQ,REPLY is specified in the instruction, a reply from the operator is required. This request for a reply is indicated by the letter A following the message. The message printed or displayed in this case is ISEQA.

The operator's response to a message must be a one-character code as follows:

Code	Meaning
0	Terminate the job.

1	Dump and terminate the job.
2	Turn on program switch 7 in the UPSI byte of the communication region (see Figure 47, Byte 23) and return to the problem program.
3	Turn OFF program switch 7 (UPSI byte) and return to the problem program.
4	Ignore the indicated I/O error and continue processing if the message was initiated by physical IOCS. If not, return to the problem program. If STR routines are included in the assembled supervisor, the message is discontinued, but processing continues. The lost data and EOF bits in the Expanded STR CCB are set on. If BSC routines are included in the assembled supervisor, the transmission is discontinued, but processing continues. The reply 4 bit in the expanded BSC CCB is set on.
5	Retry the indicated I/O operation if the message was initiated by physical IOCS. If not, return to the problem program.
6	Disable STR lines, dump the program, and terminate the job. If STR routines are not included in the assembled supervisor, the reply of 6 is an invalid reply and will be ignored.
8	Terminate the BSC job by disabling the line, displaying the error statistics and transmission counts, clearing the BSC CCB table, and dumping the program before end of job. If BSC support routines are not included in the assembled supervisor, the reply of 8 is an invalid reply and is ignored.
Other	Return to the problem program.
All Codes	The reply code is returned to the program that initiated the message, and (except for 0, 1, 6, or 8) can be tested by the program by addressing "Name+7".

The MSG instruction can be used in a system that includes an IBM 1052 Printer Keyboard, or in one that does not. However the operations differ. If a 1052 is available for messages from the MSG macro, it must be assigned to SYSLOG. Otherwise, the MSG macro cannot determine that the 1052 is available and reacts as though it were not. When a 1052 is available, the message is automatically typed and the operator types the one-character response, if REPLY is specified and indicated by the letter A. Then the response, if any, is analyzed by the supervisor and processing continues. If an error is detected during printer-keyboard input/output, operation is executed as though there were no printer-keyboard available. If no reply is required, the message will be printed and processing will continue uninterrupted.

When a 1052 is not available, programming waits (enters the wait state) and the operator can display the message on the console. The message and the response request (A), if any, are stored in fixed main-storage positions 0-4. If a reply is required, the operator must press the stop key, store the one-character code in fixed main-storage position 5, and press the start key. To resume processing, the operator presses the interruption key. If a reply is not required, the operator need only press the interruption key to continue processing.

Issuing the MSG macro instruction in the source program causes three statements to be assembled. For example:

Statement			Meaning
NAME	MSG	ISEQ,REPLY	Macro Instruction
NAME	SVC	2	Supervisor Call
	DC	CL4'ISEQ'	4-byte message
	DC	CL2'Ab'	1-byte position (A) indicates reply requested, and 1-byte position (b) provides for reply. If a reply is not required, this statement is DC CL2'bb'

During execution, the Supervisor interruption routine analyzes the code, stores the return address, moves the message to main-storage positions 0-4, and sets up the printer-keyboard (if available) for input/output operations. After the operator enters the response (either by the printer-keyboard, or by the console in main-storage position 5), the Supervisor analyzes the response code, and performs the indicated function:

- Terminates the job. (Code 0 or 1)

- Dumps and terminates the job. (Code 1)
- Sets program switch 7. (Code 2 = ON; Code 3 = OFF)
- The STR message is discontinued, but processing (on other lines or devices) continues. EOF is simulated with the lost-data and EOF bits on in the expanded STR CCB (Code 4).
- The BSC transmission is discontinued, but processing continues (on other devices). The reply 4 bit in the expanded BSC CCB is set on.
- Retries the designated operation (Code 5).
- Terminates an STR job. STR lines are disabled, and the problem program is dumped before the job is terminated, as with code 1 (Code 6).
- Terminates a BSC job. The BSC line is disabled, the error statistics and transmission count are displayed, the BSC CCB table is cleared, and the problem program is dumped before the job is terminated as with code 1 (Code 8).
- Stores the code in the last byte of the assembled statements (byte "b" shown in DC CL2'Ab')
- Returns control to the problem program, at the instruction following the MSG macro unless the code is 0, 1, or 6. If the STR routines are not included in the assembled supervisor, a reply of 6 is an invalid reply and will be ignored. The same applies to reply of 8 if BSC routines are not included in the assembled supervisor.

A MSG macro can be reused without resetting the reply byte (b).

Several programs supplied by IBM also utilize the MSG macro to give messages to the operator. These messages are numerical codes that start with a specified digit:

Digit (1052)	Displayed (On Console Byte 0)	Message from
0	11110000	IPL or Supervisor
1	11110001	Job Control
2	11110010	Linkage Editor or Program Loader
3	11110011	Other IBM programs
4	11110100	Logical IOCS

Therefore, when the user establishes his messages, he should avoid using 0-4 in the first position of his message.

COMRG Macro

Name	Op	Operand
	COMRG	

The COMRG (communication region) macro instruction allows the program to have access to information stored in the communication region. The program can either use this information, or alter the information as stored.

When this macro instruction is executed, the address of the first byte of the communication region is loaded into register 1. Using this address as a base, the program can then refer to any field in the communication region. After the macro has been executed and reference to the communication region is completed, register 1 may be used for other purposes.

Communication Region: Both the Supervisor and the problem program can use the communication region. A problem program can use the first 30 bytes of this storage area, which is arranged as shown in Figure 47 and described here:

Bytes — Field

0-8 — Calendar date (unpacked decimal), available in two forms.
For example: 08176 4230
A B

A = 8/17/64
B = year'64, 230th day

The first six bytes supply the date in the form of month/day/year (A). The last five bytes supply the date in the form of year, and day of the year (B). Form B is used by the label routine.

9 — System configuration, represented in binary form:

Bits — Configuration

0-3 — Number of bytes in main storage:
- 0000 = 8K
- 0010 = 16K
- 0011 = 24K
- 0100 = 32K
- 0110 = 64K
- 1000 = 128K
- 1010 = 256K

4 — Model (for diagnostic scan-out area):
- 0 = 30
- 1 = other

5	Floating-point feature	1 = Present
6	Decimal feature	0 = Not Present
7	Printer-Keyboard	

10-11 Address of the end of the Supervisor.

12-19 User area for interprogram or intraprogram communication. These eight bytes are not reset by Job Control.

20-22 User area for intraprogram communication. These three bytes are reset by Job Control.

23 UPSI byte. User's program switches.

24-29 Program name. The first six bytes of the name on the JOB card.

30-39 Area used by the Supervisor to retain information applicable to the problem program being executed, e.g., addresses of user-supplied routines specified by the STXIT macro.

40-45 Name of the last phase requested by a JOB card or a FETCH macro instruction.

MVCOM Macro

Name	Op	Operand
	MVCOM	byte,n,location

The MVCOM (move to communication region) macro allows the user to modify bytes 12-23 in the communication region of the Supervisor.

Before issuing the MVCOM macro, the user must set up the new information he wishes to insert in the communication region. Then when he issues the macro, he must specify three operands. He specifies the first byte that is to be modified in the communication region, the number of bytes to be modified, and the location of the new information.

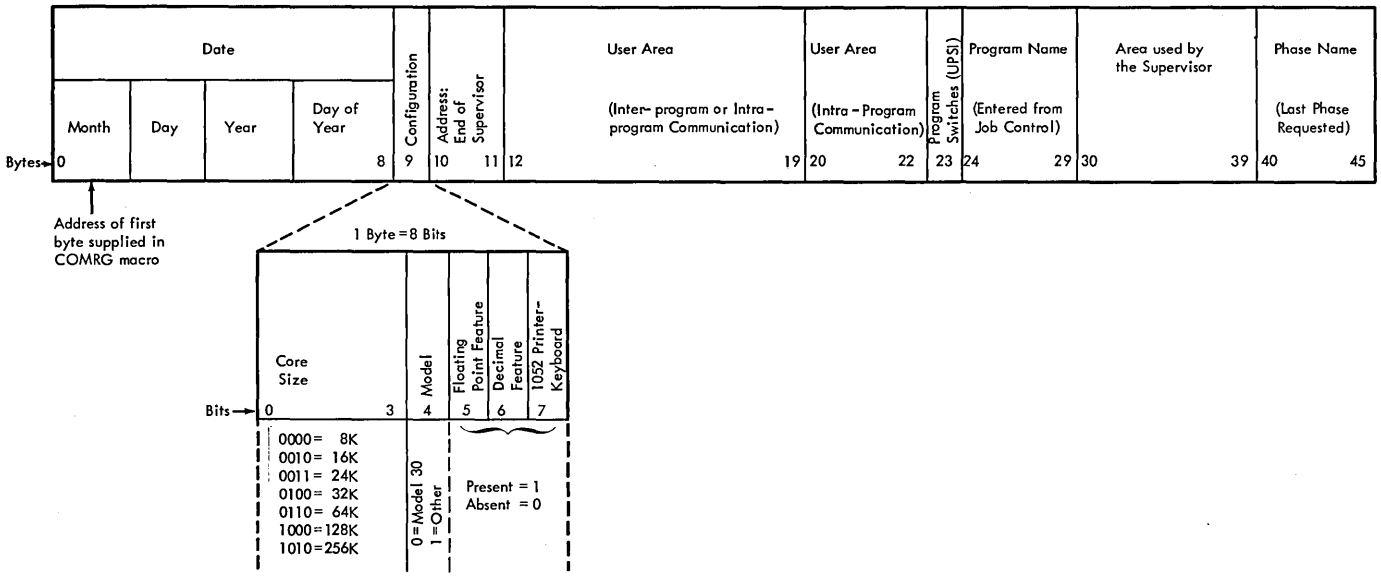


Figure 47. Communication Supervisor (in Supervisor)

The first operand (byte) specifies the first byte that is to be modified relative to the beginning of the communication region. For example, to modify the UPSI byte, 23 is punched.

The second operand specifies the number (n) of bytes to be modified. If only the UPSI byte in the previous example is to be altered, n equals 1.

The third operand (location) specifies either the symbolic name of the main-storage location that contains the information, or the register that contains the address of the information. When a register is to be specified, it must be enclosed in parentheses, and it may be either a number or a symbolic name not exceeding six characters.

Using this macro, a programmer might set up his program switches (UPSI) with the following coding:

```
Name    Op        Operand
          .
          .
          .
        MVCOM     23,1,NEWUPSI
```

Or, he might prefer to set his switches this way:

```
Name    Op        Operand
          .
          .
          .
        LA        5,NEWUPSI
          .
          .
          .
          .
        B         HERE
          .
          .
          .
          .
          .
HERE    MVCOM     23,1,(REG5)
```

In both examples, NEWUPSI could be defined this way:

```
NEWUPSI  DC       X'57'
```

STXIT Macro

Op	Operand
STXIT	n,pc-name,it-name,oc-name

The STXIT (set exit) macro instruction can be used to activate the user's interruption routines. The user may provide four different routines to which programming automatically branches on certain interruption conditions: program-check interruption, interval-timer interruption, and operator communication I/O interruption (request key on the 1052). The addresses of one or more of these four routines may be specified by one STXIT instruction.

When the problem program is executed in a disk-resident system, the interval timer (it) and operator communication (oc) routines must not be located, or begin, in the first 2500 main-storage positions above the end of the Supervisor.

Whenever this STXIT instruction is issued, the first operand must be included to specify the number (n) of a general-purpose register. The macro uses this register to refer to the address of the communication region. Any register 1-11 or 14-15 may be specified. After the macro has been executed, the register may be used for other purposes. If this STXIT instruction is issued again later in the program, the same or a different register may be specified.

Each of the other four operands is optional. One or more operands are specified, depending on the routine(s) desired. Each operand specifies either the symbolic name of the user's routine or an action to be taken by the Supervisor. The STXIT macro stores this information in the Supervisor's area of the communication region. These names or actions must be entered in the operand field in the order shown:

pc-name — Symbolic name of user's program-check interruption routine, ABORT, or DUMP.

it-name — Symbolic name of user's interval-timer interruption routine, or CLOSE.

oc-name — Symbolic name of user's operator-communication I/O interruption routine, or CLOSE.

If any name(s) is omitted but following names are to be included, a comma must be entered to indicate the omission. For

example, if this instruction is to specify Register 5, the name of a program-check routine (PROCHK), and the name of an operator-communication routine (OPCOM), it is written:

```
STXIT  5,PROCHK,,OPCOM
```

Effect of CLOSE: The addresses for the it and oc interruption routines can be cleared from the Supervisor area by specifying CLOSE as the operand in the STXIT macro. If an interruption occurs while a routine is closed (no address available), the interruption is ignored. The IPL procedure also closes these three routines.

Program Check

When an interruption occurs due to a program check, the Supervisor turns control over to the user's routine, if that routine is presently active (specified by pc-name). At the end of his routine, the programmer can return to the main line of his problem program by loading the address stored in the program-check old PSW into a register and branching to that address.

A program check may occur if the problem program specifies erroneous information to the Supervisor, such as:

- The program phase specified by a JOB card or the FETCH macrc instruction cannot be found in the core image library. Byte 43 in main storage contains a hexadecimal value of 20. Bytes 40-45 of the communication region contain the phase name, which can be examined by the user's program check routine. However, if the DUMP option is used, Bytes 40-45 do not contain the phase name, and, instead, the phase name is listed on the first line of the printout.
- The main storage size specified in the configuration byte of the communication region exceeds the storage actually available in the system.
- The floating-point feature is specified as present in the configuration byte of the communication region, but it is not actually available in the system.
- An invalid SVC (Supervisor call) code is used. The only valid SVC codes are those generated by the macros that communicate with the Supervisor (FETCH, EXCP, EXIT, MSG, DUMP, EOJ, the STR processing macros, STR device error routines, BSC support macros and BSC error routines). Byte 43 in main storage contains a hexadecimal value of 10.

If the user does not specify a routine, the actions that can be taken when a program check occurs vary depending on whether the object program is executed in a disk-resident system.

No User Routine - Disk-Resident System: Instead of specifying a routine, the user may specify either ABORT or DUMP in the STXIT macro instruction when the object program is to be executed in a disk-resident system. ABORT terminates the job; DUMP terminates the job and causes the contents of registers and main storage to be printed. The DUMP specification is always activated when Job Control is executed before the execution of the problem program.

When a program check occurs, the Supervisor issues a MSG macro instruction with a message to notify the operator. If a 1052 printer-keyboard is available, the message is typed, the previously specified action (ABORT or DUMP) is taken, and the Job Control program is loaded into main storage to continue operation with the next job. If a 1052 is not available, however, the message is stored in main-storage positions 0-3 and the system enters the wait state. The operator can display the message and then press the interruption key to continue operation. At this time the specified action (ABORT or DUMP) is taken and Job Control is fetched to continue with the next job.

No User Routine - Not a Disk-Resident System: Instead of specifying a routine, the user may specify ABORT in the STXIT macro instruction when a disk-resident system is not to be used for program execution. The ABORT specification is initially set when the Supervisor is loaded into main storage. Job Control, if used, also sets program check to ABORT. (The user may specify DUMP but, if so, it is treated the same as ABORT. That is, no printing occurs.)

When a program check occurs, the Supervisor issues a MSG macro instruction, terminates the job, and causes the system to enter the wait state. If a 1052 printer-keyboard is available, the message is typed. If not, the message is stored in main-storage positions 0-3 and the operator can display it. The execution of a different program can then be initialized.

Interval Timer

When an interruption occurs due to the timer, the Supervisor turns control over to the user's routine if that routine is presently active (specified by it-name) and if TIMER was specified in the SUPVR macro instruction when the Supervisor was

assembled. The Supervisor saves the program status of the main line of the problem program so that programming can return to it at the end of the timer-interruption routine. The Supervisor also saves the contents of general registers 10 and 11, so that these registers may be used in the timer routine. The last instruction in the timer routine should be the macro instruction EXIT TR. When this macro is executed, registers 10 and 11 are restored to their original value and the program status is restored.

Once a routine has been specified for an interval-timer interruption , that routine remains active until it is either cleared or replaced. It is cleared by the IPL procedure, or by issuing a STXIT n,,CLOSE instruction. It is replaced by issuing a STXIT instruction with a different it-name.

If TR=YES was not specified in the SUPVR macro instruction at Supervisor-assembly time, or if a timer-interruption routine is not active, any interval-timer interruption that occurs is ignored.

Operator Communication

If a 1052 printer-keyboard is available on-line, an operator can initiate a communication to cause any one of three actions: terminate a job, set a program switch, or branch to a user-supplied routine. However, if a 1052 is not available, only the first two actions can be initiated.

1052 Available: When an IBM 1052 Printer-Keyboard is available, the operator can initiate a communication to the system by pressing the request key and then keying a one-character code. The code may be one of the following:

Code	Meaning
Blank	Continue processing. Request key pressed in error.
0	Terminate the job.
1	Print out the contents of registers and main storage , and terminate the job.
2	Set program switch 7 in the UPSI byte of the communication region (see Figure 47, Byte 23) and continue with the problem program
3	(Code 2=ON; Code 3=OFF).
6	Disable STR lines, dump the problem program, and terminate the job. If the STR routines are not included in the assembled supervisor, code 6 is an invalid reply and will be ignored.
8	Terminate the BSC job by disabling the line and displaying the error statistics and transmission counts before end of job. If BSC support routines are not included in the assembled supervisor, the reply of 8 is an invalid reply and is ignored.
Other	Turn control over to the user's operator-communication routine.

When some code other than 0-3, 6, or 8 is entered, the Supervisor turns control over to the user's routine if that routine is presently active (specified by oc-name) and if INQUIRY was specified in the SUPVR macro instruction when the Supervisor was assembled. The Supervisor saves the program status of the main line of the problem program so that programming can return to it at the end of the operator-communication routine. The Supervisor also saves the contents of general registers 10 and 11 so that these registers may be used in the operator-communication routine. In this routine, a GET instruction is issued for the printer-keyboard. The operator enters his request by keying a message that the routine has been programmed to analyze. The last instruction in the operator communication routine should be the macro instruction EXIT CR. When this macro is executed, registers 10 and 11 are restored to their original value and the program status is restored.

Once a routine has been specified for an operator-communication interruption , that routine remains active until it is either cleared or replaced. It is cleared by the IPL procedure, or by issuing a STXIT n,,, CLOSE instruction. It is replaced by issuing a STXIT instruction with a different oc-name.

If CR=YES was not specified in the SUPVR macro instruction at Supervisor-assembly time, or if an operator communication routine is not active, any code other than 0-3 that is entered as an interruption request is ignored.

1052 Not Available: When a 1052 is not available, the operator can initiate communication to the Supervisor only, not to a user routine. For supervisor communication, the operator stops the system operation, enters a code 0-3, 6, or 8 in fixed main-storage position 5, and presses the start and interruption keys. As described previously, code 0 terminates the job, code 1 causes a print-out and terminates the job, code 2 or 3 sets

program switch 7 (ON or OFF, respectively) and permits processing to continue, code 6 terminates an STR job, and code 8 terminates a BSC job. (Code 6 must be used when the problem program is using STR devices and code 8 must be used when the problem program is using BSC support, to ensure that the lines are properly disabled.) If the operator enters any other code, it is ignored.

DUMP Macro

Name	Op	Operand
	DUMP	

The DUMP macro instruction applies only to object programs that will be executed in a disk system. It should be issued when an unusual condition requires that the job be terminated and that a print-out of registers and main storage be supplied to the user. It may also be issued on a normal end-of-job condition to cause a print-out. DUMP does not require an operand. In a BSC environment, the DUMP macro should be preceded by the ERRPT macro (see Binary Synchronous Communication).

When DUMP is executed, the Supervisor causes a print-out to be taken, and then fetches and turns control over to the Job Control program.

EOJ Macro

Name	Op	Operand
	EOJ	

The EOJ (end-of-job) macro instruction should be issued at the end of a problem program to inform the system (and the operator) that the job is finished. EOJ does not require an operand. In a BSC environment, the EOJ macro should be preceded by the ERRPT macro (see Binary Synchronous Communication).

When EOJ is executed in a disk system, the system loader (program-fetch routine) loads and transfers control to the Job Control program. If the disk system is not used for execution of the object program, however, control is turned over to the Supervisor and the system enters the wait state when the EOJ macro is executed. Then, if the program loader has not been overlaid, the operator presses the interruption and start keys to resume operation. This gives control to the program loader to load the next program for execution.

SUPERVISOR-ASSEMBLY MACROS

Whenever any program is executed, a Supervisor must be located in lower main storage to provide over-all control of operations. The Supervisor consists of routines to control the functions of machine interruptions, external interruptions, operator communication, and physical IOCS requests and interruptions.

Program Execution: In a Disk-Resident System

In a disk-resident system, one Supervisor contains the routines required to control the interruptions and all I/O devices in the installation. It is permanently stored on the system-residence disk pack, from which it is loaded into main storage for program executions. It can be loaded once at the beginning of the day to control the execution of all jobs run during the day. The jobs that may be executed include both the user's object programs and IBM-supplied programs, such as Assembler, Report Program Generator (RPG), etc.

This Supervisor also contains routines to load programs for execution, and to turn control over to those programs and/or receive control back from the programs that are executed. Thus the Supervisor in a disk system enables the user to stack jobs, so that one job after the other can be run without intervention from an operator.

In addition to interruption routines, physical I/O routines, and loading routines, the Supervisor contains a transient area. This area starts on a double word boundary and provides a section of main storage to which certain standard but seldom-used routines may be transferred for execution. These transient routines are permanently stored on the system-residence disk pack, and they are loaded (by the Supervisor) into the transient area only when they are needed. This means that only one section of main storage must be reserved to accommodate a number of fairly large routines. Some of the routines using this area are:

OPEN	DUMP
CLOSE	Checkpoint
End of Volume	

A Supervisor that will be used in a disk system is generally assembled once, at system-generation time when the installation is first established. It is stored on the system-residence pack at that time. Such a Supervisor is assembled to contain only the routines that are needed to control those I/O devices included in the user's configuration of the System/360. And, as previously stated, it contains

routines to control all the I/O devices in the installation. Therefore the Supervisor must be reassembled only if the system configuration changes by the addition or deletion of I/O devices or special features. Several macro instructions (SUPVR, SYMUN, IOCFG, and SEND) are used to assemble a Supervisor. These are discussed in the following pages under Macro Instructions to Assemble a Supervisor.

Additional information about the functions of a Supervisor is given in the BOS Programmer's Guide, as listed on the front cover of this publication.

Program Execution: Not in a Disk-Resident System

When a disk-resident system (with a system Supervisor) is not used for the execution of a user's object program, a Supervisor that contains all the routines required to handle the individual object program must be loaded into main storage ahead of the object program. This Supervisor must remain in main storage during the execution of the object program. If it contains the appropriate I/O routines for succeeding object programs, it can remain in main storage for their execution also.

Supervisors for object programs that are not executed in a disk-resident system must be assembled by the user either with his source program or separately. One or more Supervisors may be assembled and stored in card decks for use in an installation. The number of Supervisors that are assembled depends upon the input/output devices that are available in the installation and used in the various problem programs, and upon the available main-storage positions. The user can follow these procedures for Supervisor assembly:

- A Supervisor that contains all the I/O routines required to control all the I/O devices in the installation can be assembled once for an installation. It is then used for the execution of any problem program, regardless of what units are utilized by the program. This Supervisor does not operate in a disk-resident system and will not contain any 2311 disk input/output routines.
- Several Supervisors can be assembled such that each one controls particular groups of I/O devices. For example, one Supervisor may contain routines to handle magnetic-tape units, 2540 card reader, 2540 card punch, and 1403 printer. Another Supervisor could be assembled to handle those devices plus the 1442 card reader and the 2671 paper tape reader. Then the appropriate Supervisor would be used with each problem program to control the I/O devices used by that program.
- A Supervisor can be assembled to contain only those I/O routines that are required for the I/O devices used by a particular problem program. Such a tailored Supervisor can be assembled with the problem program, or it can be assembled separately. Separate assembly simplifies debugging and reassembly if errors are found in the problem program. In this case, only the problem program would need to be reassembled.

Assembling a Supervisor separately from the problem program is possible because linkage between the problem program and the Supervisor is accomplished by Supervisor Calls or macros. There is no symbolic linkage between the two.

Whenever a Supervisor and a problem program are assembled together, the problem program must not contain any symbolic names that start with I or SYS.

A Supervisor that controls programs that are not executed in a disk-resident system does not contain a transient area. Instead routines, such as OPEN and CLOSE, are assembled with the user's object program.

Additional information about the functions of a Supervisor is given in the Tape Programmer's Guide, as listed on the front cover of this publication.

The same macro instructions that are used to assemble the Supervisor for a disk-resident system are used to assemble Supervisors for object programs that are not to be executed in a disk-resident system. However, only those macro parameters that apply to devices utilized by the problem program should be included.

MACRO INSTRUCTIONS TO ASSEMBLE A SUPERVISOR

Whenever a Supervisor is assembled (at system-generation time, with the problem program, or separately), the assembly requires four basic macros: SUPVR (supervisor), SYMUN (symbolic units), IOCFG (I/O configuration), and SEND (supervisor end). The IOCFG macro assembles error routines for the various devices in the system.

The basic macro instructions must be issued in the order as listed and described in the following sections.

```
┌──────┬──────────────────────────────────────────────────────────┐
│Op    │ Operand                                                  │
├──────┼──────────────────────────────────────────────────────────┤
│SUPVR │ DISK=YES,CONFG=nnnnnnnn,TR=YES,CR=YES,SAVEREG=YES        │
│SUPVR │ CONFG=nnnnnnnn,TR=YES,CR=YES,CHKPT=YES                   │
└──────┴──────────────────────────────────────────────────────────┘
```

The SUPVR (Supervisor) macro instruction indicates that a Supervisor is to be assembled for operation in a particular model of the System/360. The system residence (if applicable), machine size, and available special features are specified.

Keyword parameters are specified in this macro. Therefore, the specifications may be entered in any order; those that do not apply may be omitted. Each keyword parameter except the last must be followed by a comma.

DISK=YES: This operand must be included when a Supervisor is to be assembled to operate in a disk-resident system. If the Supervisor will be used to control operations for the execution of object programs when a disk-resident system is not used, this operand must be omitted. If desired, however, DISK=NO may be used to achieve the same purpose.

CONFG=nnnnnnnn: This operand is the same as the configuration byte (byte 9) of the communication region. It must consist of eight "0" or "1" digits. The eight digits indicate the machine configuration for which the Supervisor is being assembled, as follows (numbering from the left):

Digit Positions	Configuration	
1-4	Number of bytes of main storage: 8K=0000 16K=0010 24K=0011 32K=0100 64K=0110 128K=1000 256K=1010	
5	Model * 30=0 other=1	
6	Floating-point option	Present=1
7	Decimal feature	
8	Printer=Keyboard	Not present=0

* When Model 30 is specified, 12 bytes are reserved for a diagnostic scan-out area. For any other model, 256 bytes are reserved.

If this keyword field is omitted, it is assumed to be CONFG=00000000. This is interpreted as an 8K Model 30 without the floating point option, the decimal feature, and the printer-keyboard.

If a Supervisor will ever be used in a system with main storage greater than 32K, CONFG must specify 64K (0110) or greater. This is necessary because additional coding is required in the Supervisor for configurations greater than 32K. If the assembled Supervisor is subsequently used for a configuration of 32K or less, a Job Control CONFG card must specify the machine size for program execution.

TR=YES: This operand (timer routine) must be included if the supervisor being assembled is to be used for any problem program that contains a routine(s) for the interval timer. If this operand is omitted, the interval timer (INTTMR) switch must be set OFF for any program(s) that uses this Supervisor.

CR=YES: This operand (communication routine) must be included if the Supervisor being assembled is to be used for any problem program that contains a routine(s) for operator-initiated communication from the 1052.

CHKPT=YES: This operand (checkpoint) must be included if the Supervisor being assembled is to be used for any problem program that includes the checkpoint macro or that requires restarting from checkpointed records. This operand applies to the assembly of a Supervisor only if a disk-resident system will not be used for the execution of programs that utilize this Supervisor. From the parameters in this instruction, the initial values for the Supervisor communication region are assembled.

SAVEREG=YES: This operand is optional, but must be included if the user plans to use registers 12 and 13 for programs that are executed in a disk-resident system. It directs the supervisor and physical IOCS to save and restore these registers. If the operand is omitted, the programmer must not use registers 12 and 13 because these

registers are used by the Supervisor Interruption Routine, and interruptions are unpredictable. If the supervisor is to be used to control operations for the execution of object programs when a disk-resident system is not used, this operand must be omitted.

Note: If Autotest is used, the programmer must not use registers 12 and 13 because these registers are used by the Autotest Master Control routine.

SYMUN Macro

Op	Operand
SYMUN	n,X'ccuu',X'ddss',X'ccuu',X'ddss'

Each file of input or output data is associated with an input or output device that is represented in a problem program by a symbolic unit. The symbolic unit is used instead of a device address (channel and unit). Then when the program is executed, the file of records (disk pack, tape reel, card deck, etc) is placed in an actual device and that device is assigned to the symbolic unit. Thus, the actual device (which disk or tape drive, for example) need not be determined until the job is run. All devices are handled in this manner, and a specified set of symbolic units can be used:

SYSRDR - system control-card reader
SYSLST - system printer
SYSIPT - main system input device
SYSOPT - main system output device
SYSLOG - control card logging device
SYS000-SYS254 - other devices in the system

In the last group (SYS000-SYS254) the user assigns numbers, in order, for all the devices required in his problem program.

To relate the actual device (channel and unit address) to the symbolic unit, a device table is maintained in the Supervisor. The number of entries needed in the table must be determined by the programmer when the Supervisor is assembled. The first five (SYSRDR, SYSLST, SYSIPT, SYSOPT, and SYSLOG) are assumed to be present, and he must specify only the additional number. That is, he specifies how many numbered units, starting with SYS000, are used. This must be the first operand (n) in the SYMUN macro instruction. If no units above the basic five are required, n is specified as "0".

When a supervisor is assembled with a large number of options (in SUPVR and IOCFG macros), the number of devices that can be specified (SYMUN macro) may be limited. The absolute maximum value for n is 255. However, the maximum value of n for a supervisor generated specifying a large number of options depends upon the options specified, and may be less than 255.

The assignment of an actual device to a symbolic unit can be made by the Job Control ASSGN card, or it can be made at the time the Supervisor is assembled by this SYMUN macro. Any, all (up to 24), or none of the symbolic units may be assigned actual devices at assembly time. When the assignment is made by this macro, the device remains assigned to the symbolic unit unless the assignment is overridden by a Job Control ASSGN card. Therefore, if no change in assignment is required for the execution of a program, Job Control assignment can be bypassed when the object program is loaded.

Any of the first 24 devices (SYSRDR-SYS018) to be used during program execution may be assigned by the SYMUN macro instruction. Any others must be assigned by Job Control.

When devices are assigned by the SYMUN macro, the operand should be in the positional format. A separate pair of parameters (X'ccuu',X'ddss') is entered for each device in this exact order:

```
SYSRDR, SYSLST, SYSIPT, SYSOPT, SYSLOG,
      SYS000, SYS001,---,SYS018
```

That is, the first pair of parameters (after n) assigns a card reader to be used as the system reader for control cards, the second pair assigns a printer for system listings, the seventh pair assigns a device to unit SYS001, etc. When any symbolic unit is not to be assigned a device at this time, two commas must be entered to indicate the omission if other assignments follow. If no more devices are to be assigned, however, the commas are also omitted.

The parameters (X'ccuu',X'ddss') for each assigned device are two 4-position hexadecimal numbers indicating the channel (cc) and unit (uu) address, the device type (dd), and specifications (ss), as follows:

cc - channel number
  00 - multiplexor channel
  01 - 1st selector channel
  02 - 2nd selector channel

uu - 00-FF- unit address (0-255)

dd - device type
  00 - 2401, 2402, 2403, 2404 magnetic tape unit
  02 - 1052 printer-keyboard

04 - 1442 card read-punch
06 - 1403, 1404 printer
08 - 2540 card read
0A - 2540 card punch or punch feed read
0C - 2311 disk storage drive
10 - 2671 paper tape reader
12 - 1443, 1445 printer
14 - 2501 card reader
16 - 2520 card read-punch (reading, or reading and punching for combined files)
18 - 1285 optical reader or 1287 optical reader operated in journal tape mode
1A - 2520 card punch or 2520 card read-punch (punching only)
1C - 2701 data adapter unit with SDA I (STR) data adapter unit
1E - 1287 optical reader operated in document mode
20 - 2701 data adapter unit with SDA II (BSC)

ss - specifications
These two hexadecimal characters are 00 unless 7- or 9-track tape is specified. With 7-track tape they provide 4 different options: Density, Parity, Translate, and Convert. A code is specified to represent a valid combination of options (Figure 48). For 9-track tape, codes are: C8 to indicate 800 bytes/inch; C0 to indicate 1600 bytes/inch. If this operand is omitted, a density of 800 BPI is assumed. This device specifications byte is also used internally as a queue pointer for the channel scheduler. Users should not utilize this byte for any purpose other than magnetic tape specifications.

From the parameters in this instruction, the macro sets up the physical-device table that is used by the I/O request and interruption routines.

IOCFG Macro

Op	Operand
IOCFG	keyword=YES,keyword=YES,---

The IOCFG (I/O configuration) macro instruction defines the input /output units and channels that the Supervisor must service. When a Supervisor is assembled for a disk-resident system, this macro must include all the types of devices and features (as indicated in the following list of parameters) that are available in the user's installation. However, when a Supervisor is assembled for an object program(s) that is not to be executed in a disk-resident system, only those devices and features that will be needed by the object program(s) should be specified. Thus, if the object program uses tape units, a 2540 reader, and a 1403 printer, the parameters that relate to these units should be included, and all others should be omitted. The available channel(s) and certain tape-unit specifications are also included, as indicated in the following list of parameters.

CODE	DENSITY			PARITY		CONVERT		TRANSLATE	
	200	556	800	Odd	Even	On	Off	On	Off
10	X			X		X			X
20	X				X		X		X
28	X				X		X	X	
30	X			X			X		X
38	X			X			X	X	
50		X		X		X			X
60		X			X		X		X
68		X			X		X	X	
70		X		X			X		X
78		X		X			X	X	
90			X	X		X			X
A0			X		X		X		X
A8			X		X		X	X	
B0			X	X			X		X
B8			X	X			X	X	

Figure 48. Options for 7-Track Tape

Keyword parameters are specified in this instruction. Therefore the specifications may be entered in any order, and those that do not apply may be omitted. Each keyword parameter except the last must be followed by a comma.

The parameters that are used to specify required devices are:

Parameter	Object Program(s) Requires
MPX=n	Multiplexor channel. n = 1-255 to specify the number of queues

desired for the multiplexor channel. (If omitted, n=1 is assumed.)

SEL=n — One selector channel (n is "1") or both selector channels (n is "2")

TAU=YES — Two-channel read-while-write Tape Control Unit

DVE=n — In general DVE should be specified if problem programs may contain command control blocks with the Wait for Device End bit on. Such CCB's are generated by IOCS:

1. for control commands,
2. if CRDERR=RETRY is specified for 2540 or 2520 punches,
3. if PRINTOV=YES is specified for printers.

n = maximum number of devices to be tested for Device End.

A carriage tape should not have the channel 9 punch under these two conditions:

1. Supervisor used for the problem program does not include the DVE specification.
2. Supervisor does include the DVE specification, but the PRTOV or CCB macro instruction in the problem program does not request posting of device end in the command control block (CCB).

If a 9 punch is used under either condition, a physical IOCS error message will result.

T=YES — 2401, 2402, 2403, or 2404 Magnetic Tape Unit

R1=YES — 2540 reader

R2=YES — 2540 punch feed read

R3=YES — 1442 read, or read and punch for combined files

R4=YES — 2501 reader

R5=YES — 2520 read, or read and punch for combined files

P1=YES — 2540 punch only

P2=YES — 1442 punch only

P3=YES — 2520 punch only

L1=YES — 1403 or 1404 Printer

L2=YES — 1443 or 1445 Printer

C1=YES — 1052 Printer-Keyboard

R0=YES — 2671 Paper Tape Reader

RR=YES — 1285 or 1287 (in either mode) Optical Reader

BACKWRD=YES — Backwards reading of magnetic tape

TRK7=YES — 7-track magnetic tape

TRK9=YES — 9-track magnetic tape. (If omitted and T=YES is specified, TRK9=YES is assumed.)

ST=YES — 2701 Data Adapter Unit with SDA I (STR)

ANSWR=n — Used to specify the maximum number (n=1-6) of STR devices to be monitored for ringing when using the STR SOPEN (DIAL=IN) macro. If more lines are specified to be monitored in the problem program than were specified in n of ANSWR, an error message is issued and the job is terminated.

BSC=YES — 2701 Data Adapter Unit with SDA II (BSC)

BTAB=n — Specifies the maximum number (n=1-12) of BSC CCB's used in any single BSC program. If BSC=YES is coded and BTAB is omitted, BTAB=1 is assumed.

From the parameters in this IOCFG instruction, the macro assembles the channel scheduler and I/O interruption routines for the Supervisor, and selects the appropriate device error routines.

SEND Macro

Name	Op	Operand
	SEND	n,REP,nnnn(X'nnnn')

The SEND (supervisor end) macro instruction must be included as the last instruction in the set required to assemble the Supervisor. It contains one, two or three operands.

The first operand is optional, and it specifies the number (n) of bytes to be reserved as a patch area at the end of the

Supervisor in main storage . This area can be used for changes or corrections in the Supervisor itself. With this area the Supervisor can be enlarged, if necessary, without requiring the reassembly of programs that utilize main storage immediately above the Supervisor. The minimum size of this area should be 150 bytes for IBM-supplied changes to the Supervisor. The user may also reserve additional room for expansion of the Supervisor due to changes in his system configuration or for the inclusion of additional supervisor functions. If this operand is not included but the second operand is required, a comma must be entered to indicate the omission.

Whenever a Supervisor that is not to be used in a disk-resident system is assembled, a Program Loader also is assembled. It follows the Supervisor and is considered part of the Supervisor in main storage. In this case, the patch area is located in main storage between the Supervisor and Program Loader.

The second operand (REP) also is optional and applies only to the assembly of a Supervisor and Program Loader that are not to be used in a disk-resident system. It causes the Program Loader that is assembled with the Supervisor to include a replace-text function (see Tape Programmer's Guide as listed on the front cover of this publication). The ability to process REP cards in the Program Loader can only be achieved by specifying this operand.

The third operand is also optional and applies only to the assembly of a supervisor to be used in a disk-resident system. This operand specifies the user's desired supervisor end address, and may be either in decimal or in hexadecimal. It must specify a double word boundary; if this is not done, alignment will be to the next highest double word. Enough room must be allowed for the supervisor being assembled or the supervisor could be destroyed by the transient routines. (See the supervisor core size chart or the Programmer's Guide publication.)

Whenever the third operand is used an MNOTE will be issued at assembly time to warn the user. Then the user has to check that enough room has been reserved.

Once a SEND address is specified for a given set of parameters, it will never have to be changed, thus eliminating the necessity of relinkage editing the entire system for maintenance purposes when the supervisor changes.

The specified SEND address overrides the optional patch area, as specified by the first operand of the SEND macro. It also overrides the built-in supervisor patch area which is reserved along with several options in the SUPVR and IOCFG macros.

ORGANIZATION TO ASSEMBLE A SUPERVISOR

The set of instructions in Figure 49 illustrates the requirements for assembling a Supervisor for the control of problem program execution and a Program Loader for loading problem programs. The example assumes the setup described in the following eight items:

1. Job control cards are used to prepare for the assembly. These job control cards are input to the Job Control program on the system tape. The assembled object deck is to be punched in a 2540 with the address of channel 0, unit 04. A DATE control card also is required if this assembly is the first job after performing an IPL.

2. An IPL (Initial Program Loading) loader is to be assembled with a Supervisor that is assembled for the execution of object programs when a disk-resident system is not used. The loader will be used at program execution time to load the Supervisor and Program Loader from a card reader. (If a Supervisor were to be assembled for a disk-resident system, the AOPTN IPL control card would be omitted.)

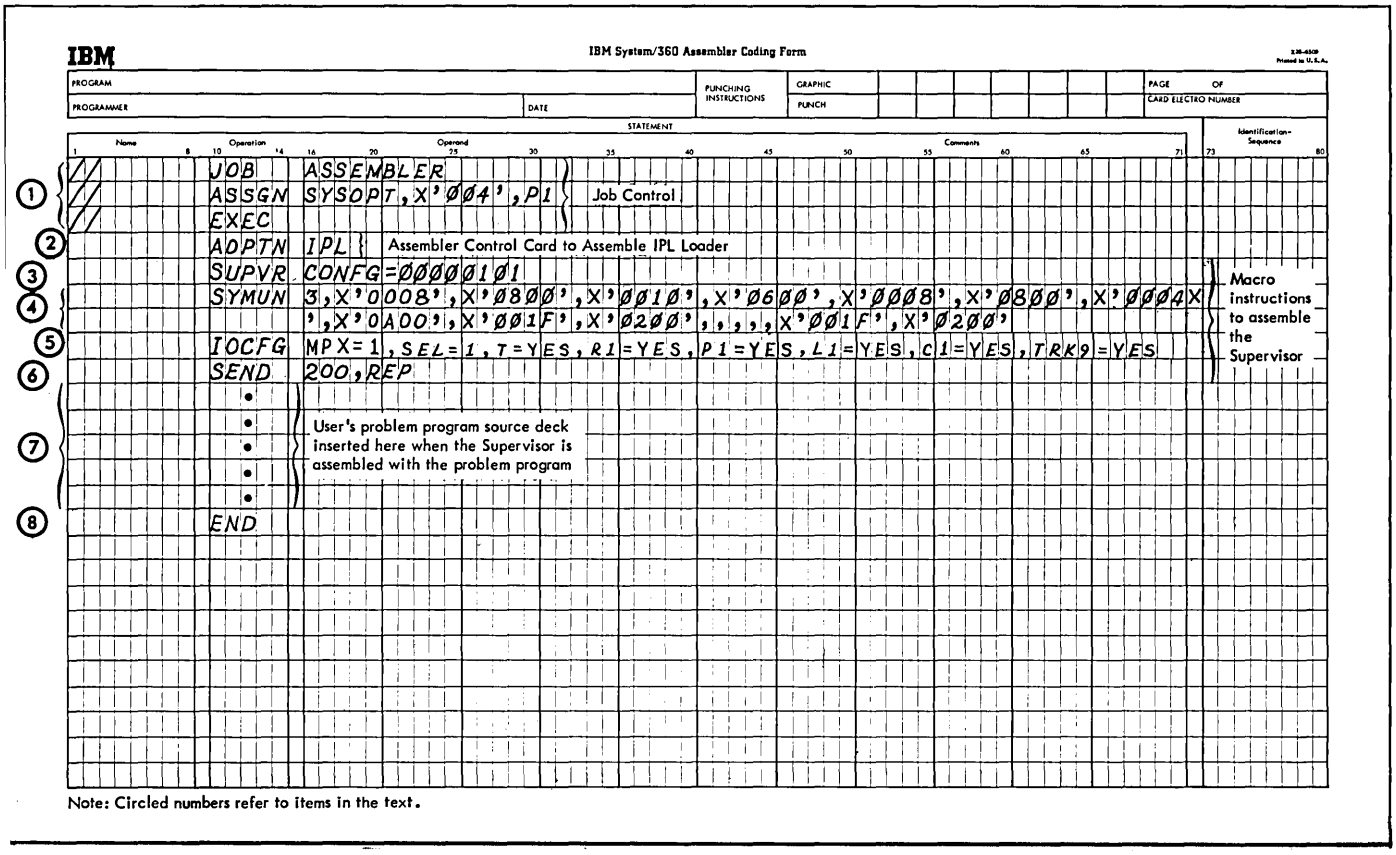


Figure 49. Assembling a Supervisor and Program Loader

3. A Supervisor is to be assembled. (If a Supervisor were to be assembled for a disk-resident system, DISK=YES would be specified.)

4. The problem program will require three I/O units over and above the basic five (SYSRDR, SYSLST, SYSIPT, SYSOPT, and SYSLOG).

   Control cards for the execution of the problem program will be read by the system reader which is a 2540 reader addressed as channel 0, unit 08.

   A 1403 printer addressed as channel 0, unit 10, is assigned as the system printer.

   The main system input device is the same device as the system reader, which is addressed as channel 0, unit 08.

   The main system output device is a 2540 card punch addressed as channel 0, unit 04.

   Control cards will be logged on a 1052 printer-keyboard addressed as channel 0, unit 1F.

   Two tape drives will be required by the object program. They will be assigned to symbolic units SYS000 and SYS001 when the program is executed. Job Control ASSGN cards, rather than the SYMUN macro, will make these assignments.

   The program also uses the 1052 printer-keyboard addressed as channel 0, unit 1F, and referred to as symbolic unit SYS002.

5 The features and types of I/O devices that will be required by the object program are:

   The multiplexor channel and one selector channel

   Magnetic tape units, with nine-track tape, on the selector channel

2540 card reader

2540 card punch

1403 printer

1052 printer-keyboard

6. This is the end of the Supervisor-Assembly macro instructions. An area of 200 bytes is to be reserved at the end of the Supervisor.

   This Supervisor is to be assembled for use in conjunction with the execution of object programs when a disk-resident system is not used. The Program Loader is assembled immediately after the Supervisor and the replace-text function is assembled as part of the Program Loader so that REP cards can be used when problem programs are loaded for execution. (If a Supervisor were to be assembled for a disk-resident system, as indicated by the SUPVR macro instruction, Program Loader would not be assembled and REP is omitted.)

7. The user's problem program is assembled next. This applies only to the combined assembly of a Supervisor and a problem program, when a disk-resident system is not to be used for the execution of the object program.

8. If the Supervisor and Program Loader are not assembled with the problem program, or if a Supervisor is assembled for a disk-resident system, the END card comes immediately after the SEND card.

JOB-CONTROL-ASSEMBLY MACROS (NOT FOR A DISK-RESIDENT SYSTEM)

When a user's problem program is to be executed, it may require the setup functions provided by a program called the Job-Control program. A Job-Control program is available on the system pack for problem programs that are to be executed in a disk-resident system. However, if a problem program is to be executed when a disk-resident system is not used, a Job-Control program must be made available. In this case, the Job Control program must be assembled by the user from IBM-supplied macro routines. Once the Job-Control program has been assembled, it can be executed any time its functions are required for a problem program. (This Job-Control program is separate from and has no relation to the Job-Control program that resides on the system pack.) For additional information about Job-Control programs, see the Programmer's Guides as listed on the front cover of this publication.

A Job-Control program has two main functions: preparing jobs to be run, and restarting the execution of a job at some point other than the beginning. The program consists of two phases, which are obtained by two assemblies, the first using the JBCTL macro and the second, the RSTRT macro.

The phase assembled by the JBCTL macro prepares jobs to be executed. It contains routines to:

1. Indicate the name of the job to be executed next.
2. Assign actual input/output devices (addresses) to the symbolic units (SYSRDR, SYSLST, etc.) used by the problem program.
3. Place today's date in the communication region of the Supervisor. This can be used by problem programs.
4. Set (ON or OFF) the user-program-switch indicators (UPSI byte) in the communication region of the Supervisor.
5. Reset the program check indicator to ABORT.
6. Edit and store label information for later use by the tape label routines.
7. Store machine configuration in communication region of the Supervisor.
8. Allow restarting of previously checkpointed records.
9. Print (log) job control cards.
10. Assign I/O request queues to multiplex mode devices on the multiplexor channel.
11. Initiate execution of of the next job.
12. Allow the wait state to be entered before the job is executed, so that the operator can perform any preparatory operations.

The user can communicate with this program using job-control cards such as JOB, ASSGN, and EXEC.

The phase obtained from the RSTRT macro contains routines for continuing the execution of an interrupted job at a point other than the beginning. The checkpoint

(CHKPT) macro must be used in the problem program to provide input information about the status of the job so that the restart phase of the Job-Control program can continue execution, if necessary.

To restart a job, both phases of Job Control must be used. The job-control cards required for restarting a checkpointed job are described in the Tape Programmer's Guide, as listed on the front cover of this manual.

These two phases of the Job-Control program must be assembled separately. The operand field of each macro instruction must be blank.

JBCTL Macro

Name	Op	Operand
	JBCTL	

RSTRT Macro

Name	Op	Operand
	RSTRT	

ASSEMBLING THE JOB CONTROL PROGRAM

To obtain both phases of the Job-Control program, two assemblies are required. The job-control cards for the system tape Job-Control program (see section 1 of Figure 20) must be used at the beginning of each assembly. Three other cards must follow the job-control EXEC card. To assemble the first phase of the Job-Control program, these three cards are:

Op	Operand
START	4096 - Assembler Start Card
JBCTL	- Macro Instruction
END	START - Assembler End Card

To assemble the restart phase of the Job-Control program, these three cards are:

Op	Operand
START	4096 - Assembler Start Card
RSTRT	- Macro Instruction
END	SYSRST - Assembler End Card

The START cards in these examples give an absolute machine address (4096), which is the address at which the user wishes to begin loading the Job-Control program. In general, the user can either specify an absolute loading address, or he can omit the address operand and use the Tape Linkage Editor to provide a job-control (or restart) card deck with the proper loading addresses.

The Assembler program operates under control of the Supervisor program, which resides on the systems pack. Certain job-control cards are therefore needed by the Job Control program to identify and initialize each Assembler run, or assembly. Four Assembler files (or five, if two workfiles are utilized) are required for an Assembler run. These files and their permissible device assignments are as follows:

1. Workfile 1 (disk)
2. If used, Workfile 2 (disk)
3. Source input (card reader or tape)
4. Text output (card punch, tape, or relocatable library)
5. Listing output (printer or tape)

Device assignments for these files are normally given at systems generation time. If, however, these assignments were not given at that time or the user wishes to alter any of the assignments, certain job-control cards may be used at assembly time to make the desired assignments. The types of job-control cards and their functions are described in Appendix I and the Programmer's Guide for Basic Operating System.

As shown in the preceding list, three of the Assembler files (source input, text output, and listing output) may be assigned to tape units. They each may be assigned to a separate unit or one of the following two combinations may be assigned to the same unit:

1. source input and text output
2. source input and listing output.

Either 7- or 9-track tape may be used for these assignments, (see Figure 55 for 7-track tape requirements). Special messages produced by an ALOG card are not considered to be an Assembler file. They are listed on the printing device that is assigned to SYSLOG.

Certain control cards may be supplied for use by the Assembler program. These cards indicate which of the Assembler processing options the Assembler program is to perform or provide. The Assembler control cards are: AOPTN (Assembler option) card, ALOG (Assembler log) card, AWORK (Assembler workfile) card, and AFILE (system file) card. These cards have the same format as Assembler-language instructions. The AOPTN, ALOG, AWORK, and AFILE appear as if they were mnemonic operation codes, and the various options are specified as operand(s).

AOPTN (Assembler Option) Card

AOPTN card(s) are required only if the user wishes to alter the normal output from the Assembler.

The normal Assembler output consists of two major files: the object program and the program listing. The object program consists of three types of information: the external symbol dictionary (ESD), text, and the relocation dictionary (RLD). The program listing consists of five lists of information: ESD listing, source and object program listing, RLD listing, error listing, symbol table, or cross reference listing.

The option(s) that may be supplied in the AOPTN cards are shown in Figure 50; each option is identified by a symbol which is used in the AOPTN card. Each option of the AOPTN card may be specified in a different AOPTN card, or the options may appear as multiple operands (separated by commas) in a single card.

If the user wishes to include the options BGNBATCH (begin batch processing), BATCH, and ENDBATCH, the AOPTN BGNBATCH card should be included only in the first assembly, and AOPTN BATCH in each succeeding assembly except the last. In the last assembly, the AOPTN ENDBATCH card should be included to discontinue the batch processing environment. Job control cards should precede only the first assembly (Figure 51).

ALOG (Assembler Log) Card

The contents of the ALOG card, all assembler-option (AOPTN) cards, all assembler workfile (AWORK) cards, and all assembler system file (AFILE) cards following the ALOG card are printed out on SYSLOG. A statement of the total number of errors also is printed out on SYSLOG.

OPTION	MEANING
NODECK	The object deck will not be produced in cards, tape, or disk. (This does not affect the appearance of the program listing). Also, no cards will be produced as a result of REPRO or PUNCH instructions.
NOESD	No ESD data will appear in the object program or the program listing. (Program will not be acceptable to a Basic Operating System or Operating System).
NORLD	No RLD data will appear in the object program or the program listing. (Program will not be relocatable).
NOLIST	The program listing will not appear.
NOERR	The error listing will not appear in the program listing.
NOSYM	The symbol table will not appear in the program listing.
PCHSYM	The symbol table will be punched out (depending on the device) on the same device as specified for text output. This table is in the format required for the Autctest program.
CROSSREF	A cross-reference listing will appear instead of the symbol table listing. The cross-reference listing contains up to a total of 20,625 statement number references to symbols used. If more than 20,625 references are used, symbols and references above that number will not be shown.
IPL	An IPL routine (for loading an independent Supervisor from a card reader) will precede the object program.
ENTRY	An ENTRY card will be produced at the end of the output text.
BGNBATCH	If SYSLST and SYSOPT are assigned to a tape unit, the tapes will be opened, initiating a batched environment. The listing and output tapes will not be closed at the end of assembly.
BATCH	A batched environment will be initiated or continued. Listing and output tapes will not be closed.
ENDBATCH	If SYSLST and SYSOPT are assigned to tape, the tapes will be closed. The batched environment will be discontinued and end-of-job will be called.

Figure 50. AOPTN Card Option Indicators

AWORK (Assembler Workarea) Card

An AWORK card indicates the number of workareas the Assembler is to use in processing source-program statements. The Assembler can use either one or two workareas. Normally two workareas are provided only if two disk drives are available. In this case, one workarea can be assigned to each disk drive. Using two workareas on separate disk drives shortens the processing time required by the Assembler.

The AWORK card requires one operand. If the digit 1 appears as the operand, the Assembler assumes one workarea. If the digit 2 is used, the Assembler assumes two workareas. If no AWORK card is provided, the Assembler assumes one workarea.

A workarea must always be assigned to a continuous disk area between specified limits (XTENT) on one disk pack. This is accomplished by using the proper job-control cards.

AFILE (Assembler File) Card

An AFILE card directs the Assembler to place the entire output deck (see Figure 57) in the relocatable library as a module. The format of the AFILE Assembler control card is:

Name	Op	Operand
Name	AFILE	LIBRARY, RETAIN or LIBRARY

A name must be given in the name field. The name given may include one to six characters. Any additional characters are ignored. (The name ALL is not allowed.) The name is placed in the relocatable library directory to identify the module.

The first operand is LIBRARY and must always appear in the AFILE card. If RETAIN appears as the second operand in the AFILE card, the module is entered permanently and remains until it is deleted or replaced.

Note:
If any of the other assembler options are used, the cards can be placed anywhere in this deck between the EXEC card and the appropriate source deck.

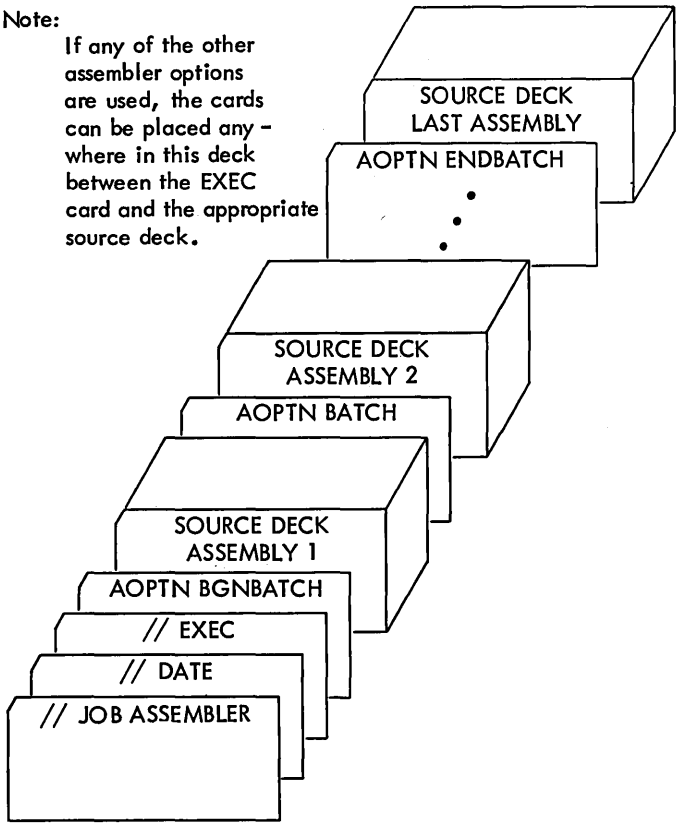


Figure 51. Batch Processing If the second operand is omitted, the module is entered temporarily and will be overlaid by the next module entered into the relocatable library.

Based upon the intended usage of the module, the user must provide a PHASE and/or an ENTRY card in the module. The initial PHASE card can be obtained with a PUNCH or REPRO statement placed before the (actual or implied) START statement. An ENTRY card can be obtained with an AOPTN ENTRY card. A description of the alternatives available for processing modules in the relocatable library may be found in the section Linkage Editor in the BOS Programmer's Guide as listed on the front cover of this publication.

It is possible to load and execute an object program immediately after the assembly is completed. This may be accomplished by entering the name of the first phase to be executed as the second operand of the JOB card used for the Assembler run. The module created for this purpose must be terminated by an ENTRY card and must contain one or more PHASE cards. This procedure for processing a module immediately after it is assembled and placed in the relocatable library has no effect on the future availability of the module. It remains in the relocatable library and may be accessed normally until deleted or (in the case of a temporary module) the next module is entered.

Note: Multiple phases are created in a single assembly by using an XFR card to end a phase. A PHASE card, created by a REPRO or PUNCH statement, follows this XFR card and precedes the next phase in line. To ensure proper execution, each phase created in this manner should be a separate control section.

There are three separate System/360 assembler languages. One is for Basic Programming Support (Card) and is called the basic assembler language. The second is for Basic Programming Support and Basic Operating System and is called the tape and disk assembler language. The third is the Operating System assembler language. The basic assembler language is a subset of the tape and disk assembler language, which in turn is a subset of the Operating System assembler language. Any language features that exist on one level exist on all higher levels, with the exception of the XFR assembler instruction. (The XFR assembler instruction is unique to the tape and disk assembler language.)

The differences between the Operating System assembler language and the Basic Operating System disk assembler language are shown in Figure 52.

LANGUAGE FEATURE	BASIC OPERATING SYSTEM SPECIFICATION	OPERATING SYSTEM SPECIFICATION
CODING CONVENTIONS		
Continuation Lines	Maximum of one continuation line allowed	Maximum of two continuation lines allowed
CONSTANTS		
Address Constants	Only one address constant may be specified in a DC statement or a literal	One or more address constants may be specified in a DC statement or a literal
Bit-Length Specification	Feature excluded	Feature is provided in Operating System
DC Operands	Only one operand allowed per DC statement	One or more operands allowed per DC statement
Duplication Factor	Duplication factor expressable only by a decimal self-defining term	Duplication factor expressable by any absolute expression
Exponent Modifier	Exponent modifier expressable only by a decimal self-defining term	Exponent modifier expressable by any absolute expression
Length Modifier	Length modifier expressable only by a decimal self-defining term	Length modifier expressable by any absolute expression
Literals	In case of duplicate literals, more than one may be stored	In case of duplicate literals, only one is stored
Scale Modifier	Scale modifier expressable only by a decimal self-defining term	Scale modifier expressable by any absolute expression
EXPRESSIONS		
Parentheses	Only one set of parentheses ( ) allowed in an expression	No limit on the number of sets of parentheses ( ) allowed in an expression
Terms	Maximum of three terms allowed per expression	No limit on the number of terms allowed per expression
STORAGE DEFINITION (DS STATEMENT)		
DS Operand	Only one operand allowed in a DS statement	One or more operands allowed in a DS statement
Length Modifier	Maximum length designation of a storage field is 256 bytes	Maximum length designation of a storage field is 65,536 bytes
ASSEMBLER-INSTRUCTION STATEMENTS		
CNOP	Each operand expressable only by a decimal self-defining term	Each operand expressable by any absolute expression
COM	Statement excluded	Statement provided in Operating System
COPY	Statement excluded	Statement provided in Operating System
CSECT and DSECT	In addition to the combined number of CSECTs, DSECTs, EXTRNs, and V-type address constants not being allowed to exceed 255, the combined number of CSECT and DSECT statements must not exceed 32.	The combined number of CSECT statements, DSECT statements, EXTRN statements, and V-type address constants must not exceed 255
EXTRN and ENTRY	Only one relocatable symbol is allowed in each EXTRN and ENTRY statement	One or more relocatable symbols are allowed in each EXTRN and ENTRY statement
TITLE	The first TITLE statement provides the heading only for pages of the listing that follow it, until the next TITLE statement (if any) is encountered	The first TITLE statement, in addition to providing the heading for all pages of the listing that lie between it and the next TITLE statement (if any), also provides the heading for any page(s) of the listing that precede it.
USING and DROP	Maximum of 5 base register designations allowed in each USING or DROP statement	Maximum of 15 base register designations allowed in each USING or DROP statement
XFR	Statement provided in Basic Operating System	Statement excluded

Figure 52. Differences Between Basic Operating System and Operating System Assembler Languages

This appendix lists all System/360 card codes to which a printer graphic is assigned. See Part 2 for a complete list of character codes.

Card Code	Printer Graphics	Internal Representation	Card Code	Printer Graphics	Internal Representation
	blank	01000000	11,1	J	11010001
12,8,3	. (period)	01001011	11,2	K	11010010
12,8,4	<	01001100	11,3	L	11010011
12,8,5	(	01001101	11,4	M	11010100
12,8,6	+	01001110	11,5	N	11010101
12	&	01010000	11,6	O	11010110
11,8,3	$	01011011	11,7	P	11010111
11,8,4	*	01011100	11,8	Q	11011000
11,8,5	)	01011101	11,9	R	11011001
11	-	01100000	0,2	S	11100010
0,1	/	01100001	0,3	T	11100011
0,8,3	,	01101011	0,4	U	11100100
0,8,4	%	01101100	0,5	V	11100101
8,3	#	01111011	0,6	W	11100110
8,4	@	01111100	0,7	X	11100111
8,5	' (prime)	01111101	0,8	Y	11101000
8,6	=	01111110	0,9	Z	11101001
12,1	A	11000001	0	0	11110000
12,2	B	11000010	1	1	11110001
12,3	C	11000011	2	2	11110010
12,4	D	11000100	3	3	11110011
12,5	E	11000101	4	4	11110100
12,6	F	11000110	5	5	11110101
12,7	G	11000111	6	6	11110110
12,8	H	11001000	7	7	11110111
12,9	I	11001001	8	8	11111000
			9	9	11111001

APPENDIX A: CHARACTER CODES--PART 2

8-BIT BCD CODE	CHARACTER SET PUNCH COMBINATION	PRINTER GRAPHICS	DECIMAL	HEXADECIMAL
00000000	12,0,9,8,1		0	00
00000001	12,9,1		1	01
00000010	12,9,2		2	02
00000011	12,9,3		3	03
00000100	12,9,4		4	04
00000101	12,9,5		5	05
00000110	12,9,6		6	06
00000111	12,9,7		7	07
00001000	12,9,8		8	08
00001001	12,9,8,1		9	09
00001010	12,9,8,2		10	0A
00001011	12,9,8,3		11	0B
00001100	12,9,8,4		12	0C
00001101	12,9,8,5		13	0D
00001110	12,9,8,6		14	0E
00001111	12,9,8,7		15	0F
00010000	12,11,9,8,1		16	10
00010001	11,9,1		17	11
00010010	11,9,2		18	12
00010011	11,9,3		19	13
00010100	11,9,4		20	14
00010101	11,9,5		21	15
00010110	11,9,6		22	16
00010111	11,9,7		23	17
00011000	11,9,8		24	18
00011001	11,9,8,1		25	19
00011010	11,9,8,2		26	1A
00011011	11,9,8,3		27	1B
00011100	11,9,8,4		28	1C
00011101	11,9,8,5		29	1D
00011110	11,9,8,6		30	1E
00011111	11,9,8,7		31	1F
00100000	11,0,9,8,1		32	20
00100001	0,9,1		33	21
00100010	0,9,2		34	22
00100011	0,9,3		35	23
00100100	0,9,4		36	24
00100101	0,9,5		37	25
00100110	0,9,6		38	26
00100111	0,9,7		39	27
00101000	0,9,8		40	28
00101001	0,9,8,1		41	29
00101010	0,9,8,2		42	2A
00101011	0,9,8,3		43	2B
00101100	0,9,8,4		44	2C
00101101	0,9,8,5		45	2D
00101110	0,9,8,6		46	2E
00101111	0,9,8,7		47	2F
00110000	12,11,0,9,8,1		48	30
00110001	9,1		49	31
00110010	9,2		50	32
00110011	9,3		51	33
00110100	9,4		52	34
00110101	9,5		53	35
00110110	9,6		54	36
00110111	9,7		55	37
00111000	9,8		56	38
00111001	9,8,1		57	39

00111010	9,8,2		58	3A
00111011	9,8,3		59	3B
00111100	9,8,4		60	3C
00111101	9,8,5		61	3D
00111110	9,8,6		62	3E
00111111	9,8,7		63	3F
01000000		blank	64	40
01000001	12,0,9,1		65	41
01000010	12,0,9,2		66	42
01000011	12,0,9,3		67	43
01000100	12,0,9,4		68	44
01000101	12,0,9,5		69	45
01000110	12,0,9,6		70	46
01000111	12,0,9,7		71	47
01001000	12,0,9,8		72	48
01001001	12,8,1		73	49
01001010	12,8,2		74	4A
01001011	12,8,3	. (period)	75	4B
01001100	12,8,4	<-	76	4C
01001101	12,8,5	(	77	4D
01001110	12,8,6	+	78	4E
01001111	12,8,7		79	4F
01010000	12	&	80	50
01010001	12,11,9,1		81	51
01010010	12,11,9,2		82	52
01010011	12,11,9,3		83	53
01010100	12,11,9,4		84	54
01010101	12,11,9,5		85	55
01010110	12,11,9,6		86	56
01010111	12,11,9,7		87	57
01011000	12,11,9,8		88	58
01011001	11,8,1		89	59
01011010	11,8,2		90	5A
01011011	11,8,3	$	91	5B
01011100	11,8,4	*	92	5C
01011101	11,8,5	)	93	5D
01011110	11,8,6		94	5E
01011111	11,8,7		95	5F
01100000	11	-	96	60
01100001	0,1	/	97	61
01100010	11,0,9,2		98	62
01100011	11,0,9,3		99	63
01100100	11,0,9,4		100	64
01100101	11,0,9,5		101	65
01100110	11,0,9,6		102	66
01100111	11,0,9,7		103	67
01101000	11,0,9,8		104	68
01101001	0,8,1		105	69
01101010	12,11		106	6A
01101011	0,8,3	,	107	6B
01101100	0,8,4	%	108	6C
01101101	0,8,5		109	6D
01101110	0,8,6		110	6E
01101111	0,8,7		111	6F
01110000	12,11,0		112	70
01110001	12,11,0,9,1		113	71
01110010	12,11,0,9,2		114	72
01110011	12,11,0,9,3		115	73
01110100	12,11,0,9,4		116	74
01110101	12,11,0,9,5		117	75
01110110	12,11,0,9,6		118	76
01110111	12,11,0,9,7		119	77
01111000	12,11,0,9,8		120	78
01111001	8,1		121	79
01111010	8,2		122	7A
01111011	8,3	#	123	7B
01111100	8,4	@	124	7C
01111101	8,5	'	125	7D

01111110	8,6	=	126	7E
01111111	8,7		127	7F
10000000	12,0,8,1		128	80
10000001	12,0,1		129	81
10000010	12,0,2		130	82
10000011	12,0,3		131	83
10000100	12,0,4		132	84
10000101	12,0,5		133	85
10000110	12,0,6		134	86
10000111	12,0,7		135	87
10001000	12,0,8		136	88
10001001	12,0,9		137	89
10001010	12,0,8,2		138	8A
10001011	12,0,8,3		139	8B
10001100	12,0,8,4		140	8C
10001101	12,0,8,5		141	8D
10001110	12,0,8,6		142	8E
10001111	12,0,8,7		143	8F
10010000	12,11,8,1		144	90
10010001	12,11,1		145	91
10010010	12,11,2		146	92
10010011	12,11,3		147	93
10010100	12,11,4		148	94
10010101	12,11,5		149	95
10010110	12,11,6		150	96
10010111	12,11,7		151	97
10011000	12,11,8		152	98
10011001	12,11,9		153	99
10011010	12,11,8,2		154	9A
10011011	12,11,8,3		155	9B
10011100	12,11,8,4		156	9C
10011101	12,11,8,5		157	9D
10011110	12,11,8,6		158	9E
10011111	12,11,8,7		159	9F
10100000	11,0,8,1		160	A0
10100001	11,0,1		161	A1
10100010	11,0,2		162	A2
10100011	11,0,3		163	A3
10100100	11,0,4		164	A4
10100101	11,0,5		165	A5
10100110	11,0,6		166	A6
10100111	11,0,7		167	A7
10101000	11,0,8		168	A8
10101001	11,0,9		169	A9
10101010	11,0,8,2		170	AA
10101011	11,0,8,3		171	AB
10101100	11,0,8,4		172	AC
10101101	11,0,8,5		173	AD
10101110	11,0,8,6		174	AE
10101111	11,0,8,7		175	AF
10110000	12,11,0,8,1		176	B0
10110001	12,11,0,1		177	B1
10110010	12,11,0,2		178	B2
10110011	12,11,0,3		179	B3
10110100	12,11,0,4		180	B4
10110101	12,11,0,5		181	B5
10110110	12,11,0,6		182	B6
10110111	12,11,0,7		183	B7
10111000	12,11,0,8		184	B8
10111001	12,11,0,9		185	B9
10111010	12,11,0,8,2		186	BA
10111011	12,11,0,8,3		187	BB
10111100	12,11,0,8,4		188	BC
10111101	12,11,0,8,5		189	BD
10111110	12,11,0,8,6		190	BE
10111111	12,11,0,8,7		191	BF
11000000	12,0		192	C0
11000001	12,1	A	193	C1

11000010	12,2	B	194	C2
11000011	12,3	C	195	C3
11000100	12,4	D	196	C4
11000101	12,5	E	197	C5
11000110	12,6	F	198	C6
11000111	12,7	G	199	C7
11001000	12,8	H	200	C8
11001001	12,9	I	201	C9
11001010	12,0,9,8,2		202	CA
11001011	12,0,9,8,3		203	CB
11001100	12,0,9,8,4		204	CC
11001101	12,0,9,8,5		205	CD
11001110	12,0,9,8,6		206	CE
11001111	12,0,9,8,7		207	CF
11010000	11,0		208	D0
11010001	11,1	J	209	D1
11010010	11,2	K	210	D2
11010011	11,3	L	211	D3
11010100	11,4	M	212	D4
11010101	11,5	N	213	D5
11010110	11,6	O	214	D6
11010111	11,7	P	215	D7
11011000	11,8	Q	216	D8
11011001	11,9	R	217	D9
11011010	12,11,9,8,2		218	DA
11011011	12,11,9,8,3		219	DB
11011100	12,11,9,8,4		220	DC
11011101	12,11,9,8,5		221	DD
11011110	12,11,9,8,6		222	DE
11011111	12,11,9,8,7		223	DF
11100000	0,8,2		224	E0
11100001	11,0,9,1		225	E1
11100010	0,2	S	226	E2
11100011	0,3	T	227	E3
11100100	0,4	U	228	E4
11100101	0,5	V	229	E5
11100110	0,6	W	230	E6
11100111	0,7	X	231	E7
11101000	0,8	Y	232	E8
11101001	0,9	Z	233	E9
11101010	11,0,9,8,2		234	EA
11101011	11,0,9,8,3		235	EB
11101100	11,0,9,8,4		236	EC
11101101	11,0,9,8,5		237	ED
11101110	11,0,9,8,6		238	EE
11101111	11,0,9,8,7		239	EF
11110000	0	0	240	F0
11110001	1	1	241	F1
11110010	2	2	242	F2
11110011	3	3	243	F3
11110100	4	4	244	F4
11110101	5	5	245	F5
11110110	6	6	246	F6
11110111	7	7	247	F7
11111000	8	8	248	F8
11111001	9	9	249	F9
11111010	12,11,0,9,8,2		250	FA
11111011	12,11,0,9,8,3		251	FB
11111100	12,11,0,9,8,4		252	FC
11111101	12,11,0,9,8,5		253	FD
11111110	12,11,0,9,8,6		254	FE
11111111	12,11,0,9,8,7		255	FF

This appendix contains a table of the mnemonic operation codes for all machine instructions that can be represented in assembler language, including extended mnemonic operation codes. It is in alphabetic order by instruction. Indicated for each instruction are both the mnemonic and machine operation codes, explicit and implicit operand formats, program interruptions possible, and condition code set.

The column headings in this appendix and the information each column provides follow.

Instruction: This column contains the name of the instruction associated with the mnemonic operation code.

Mnemonic Operation Code: This column gives the mnemonic operation code for the machine instruction. This is written in the operation field when coding the instruction.

Machine Operation Code: This column contains the hexadecimal equivalent of the actual machine operation code. The operation code will appear in this form in most storage dumps and when displayed on the system control panel. For extended mnemonics, this column also contains the mnemonic code of the instruction from which the extended mnemonic is derived.

Operand Format: This column shows the symbolic format of the operand field in both explicit and implicit form. For both forms, R1, R2, and R3 indicate general registers in operands one, two, and three, respectively. X2 indicates a general register used as an index register in the second operand. Instructions which require an index register (X2) but are not to be indexed are shown with a 0 replacing X2. L, L1, and L2 indicate lengths for either operand, operand one, and operand two, respectively.

For the explicit format, D1 and D2 indicate a displacement and B1 and B2 indicate a base register for operands one and two.

For the implicit format, D1,B1 and D2,B2 are replaced by S1 and S2 which indicate a storage address in operands one and two.

Type of Instruction: This column gives the basic machine format of the instruction (RR, RX, SI, or SS). If an instruction is included in a special feature or is an extended mnemonic, this is also indicated.

Program Interruptions Possible: This column indicates the possible program interruptions for this instruction. The abbreviations used are: A - Addressing, S - Specification, Ov - Overflow, P - Protection, Op - Operation (if feature is not installed) and Other - other interruptions which are listed. The type of overflow is indicated by: D - Decimal, E - Exponent, or F - Floating Point.

Condition Code Set: The condition codes set as a result of this instruction are indicated in this column. (See legend following the table).

Instruction	Mnemonic Operation Code	Machine Operation Code	Operand Format	
			Explicit	Implicit
Add	A	5A	R1,D2(X2,B2) or R1, D2(,B2)	R1,S2(X2) or R1,S2
Add	AR	1A	R1,R2	
Add Decimal	AP	FA	D1(L1,B1),D2(L2,B2)	S1(L1),S2(L2)or S1,S2
Add Halfword	AH	4A	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2)or R1,S2
Add Logical	AL	5E	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2)or R1,S2
Add Logical	ALR	1E	R1,R2	
Add Normalized,Long	AD	6A	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2)or R1,S2
Add Normalized,Long	ADR	2A	R1,R2	
Add Normalized,Short	AE	7A	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2)or R1,S2
Add Normalized, Short	AER	3A	R1,R2	
Add Unnormalized,Long	AW	6E	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2)or R1,S2
Add Unnormalized,Long	AWR	2E	R1,R2	
Add Unnormalized,Short	AU	7E	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2)or R1,S2
Add Unnormalized,Short	AUR	3E	R1,R2	
And Logical	N	54	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2)or R1,S2
And Logical	NC	D4	D1(L,B1),D2(B2)	S1(L),S2 or S1,S2
And Logical	NR	14	R1,R2	
And Logical Immediate	NI	94	D1(B1),I2	S1,I2
Branch and Link	BAL	45	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2)or R1,S2
Branch and Link	BALR	05	R1,R2	
Branch on Condition	BC	47	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2)or R1,S2
Branch on Condition	BCR	07	R1,R2	
Branch on Count	BCT	46	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2)or R1,S2
Branch on Count	BCTR	06	R1,R2	
Branch on Equal	BE	47(BC 8)	D2(X2,B2)or D2(,B2)	S2(X2) or S2
Branch on High	BH	47(BC 2)	D2(X2,B2)or D2(,B2)	S2(X2) or S2
Branch on Index High	BXH	86	R1,R3,D2(B2)	R1,R3,S2
Branch on Index Low or Equal	BXLE	87	R1,R3,D2(B2)	R1,R3,S2
Branch on Low	BL	47(BC 4)	D2(X2,B2)or D2(,B2)	S2(X2) or S2
Branch if Mixed	BM	47(BC 4)	D2(X2,B2)or D2(,B2)	S2(X2) or S2
Branch on Minus	BM	47(BC 4)	D2(X2,B2)or D2(,B2)	S2(X2) or S2
Branch on Not Equal	BNE	47(BC 7)	D2(X2,B2)or D2(,B2)	S2(X2) or S2
Branch on Not High	BNH	47(BC 13)	D2(X2,B2)or D2(,B2)	S2(X2) or S2
Branch on Not Low	BNL	47(BC 11)	D2(X2,B2)or D2(,B2)	S2(X2) or S2
Branch if Ones	BO	47(BC 1)	D2(X2,B2)or D2(,B2)	S2(X2) or S2
Branch on Overflow	BO	47(BC 1)	D2(X2,B2)or D2(,B2)	S2(X2) or S2
Branch on Plus	BP	47(BC 2)	D2(X2,B2)or D2(,B2)	S2(X2) or S2
Branch if Zeros	BZ	47(BC 8)	D2(X2,B2)or D2(,B2)	S2(X2) or S2
Branch on Zero	BZ	47(BC 8)	D2(X2,B2)or D2(,B2)	S2(X2) or S2
Branch Unconditional	B	47(BC 15)	D2(X2,B2)or D2(,B2)	S2(X2) or S2
Branch Unconditional	BR	07(BCR 15)	R2	
Compare Algebraic	C	59	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2 or R1,S2
Compare Algebraic	CR	19	R1,R2	
Compare Decimal	CP	F9	D1(L1,B1),D2(L2,B2)	S1(L1),S2(L2)or S1,S2
Compare Halfword	CH	49	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2)or R1,S2
Compare Logical	CL	55	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2)or R1,S2
Compare Logical	CLC	D5	D1(L,B1),D2(B2)	S1(L),S2 or S1,S2
Compare Logical	CLR	15	R1,R2	
Compare Logical Immediate	CLI	95	D1(B1),I2	S1,I2
Compare,Long	CD	69	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2)or R1,S2
Compare,Long	CDR	29	R1,R2	
Compare,Short	CE	79	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2)or R1,S2
Compare,Short	CER	39	R1,R2	
Convert to Binary	CVB	4F	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2)or R1,S2
Convert to Decimal	CVD	4E	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2)or R1,S2

Operand Format (Add)

Instruction	Type of Instruction	Program Interruption Possible						Condition Code Set			
		A	S	Ov	P	Op	Other	00	01	10	11
Add	RX	x	x	F				Sum=0	Sum<0	Sum>0	Overflow
Add	RR			F				Sum=0	Sum<0	Sum>0	Overflow
Add Decimal	SS,Decimal	x		D	x	x	Data	Sum=0	Sum<0	Sum>0	Overflow
Add Halfword	RX	x	x	F				Sum=0	Sum<0	Sum>0	Overflow
Add Logical	RX	x	x					Sum=0 (H)	Sum 0 (H)	Sum= 0 (I)	Sum 0 (I)
Add Logical	RR							Sum=0 (H)	Sum= 0 (H)	Sum= 0 (I)	Sum 0 (I)
Add Normalized, Long	RX,Floating Pt.	x	x	E		x	B,C	R	L	M	P
Add Normalized, Long	RR,Floating Pt.		x	E		x	B,C	R	L	M	P
Add Normalized, Short	RX,Floating Pt.	x	x	E		x	B,C	R	L	M	P
Add Normalized, Short	RR,Floating Pt.		x	E		x	B,C	R	L	M	P
Add Unnormalized, Long	RX,Floating Pt.	x	x	E		x	C	R	L	M	P
Add Unnormalized, Long	RR,Floating Pt.		x	E		x	C	R	L	M	P
Add Unnormalized, Short	RX,Floating Pt.	x	x	E		x	C	R	L	M	P
Add Unnormalized, Short	RR,Floating Pt.		x	E		x	C	R	L	M	P
Add Logical	RX	x	x					J	K		
And Logical	SS	x			x			J	K		
And Logical	RR							J	K		
And Logical Immediate	SI	x			x			J	K		
Branch and Link	RX							N	N	N	N
Branch and Link	RR							N	N	N	N
Branch on Condition	RX							N	N	N	N
Branch on Condition	RR							N	N	N	N
Branch on Count	RX							N	N	N	N
Branch on Count	RR							N	N	N	N
Branch on Equal	RX,Ext.Mnemonic							N	N	N	N
Branch on High	RX,Ext.Mnemonic							N	N	N	N
Branch on Index High	RX,Ext.Mnemonic							N	N	N	N
Branch on Index Low or Equal	RX,Ext.Mnemonic							N	N	N	N
Branch on Low	RX,Ext.Mnemonic							N	N	N	N
Branch if Mixed	RX,Ext.Mnemonic							N	N	N	N
Branch on Minus	RX,Ext.Mnemonic							N	N	N	N
Branch on Not Equal	RX,Ext.Mnemonic							N	N	N	N
Branch on Not High	RX,Ext.Mnemonic							N	N	N	N
Branch on Not Low	RX,Ext.Mnemonic							N	N	N	N
Branch if Ones	RX,Ext.Mnemonic							N	N	N	N
Branch on Overflow	RX,Ext.Mnemonic							N	N	N	N
Branch on Plus	RX,Ext.Mnemonic							N	N	N	N
Branch if Zeros	RX,Ext.Mnemonic							N	N	N	N
Branch on Zero	RX,Ext.Mnemonic							N	N	N	N
Branch Unconditional	RX,Ext.Mnemonic							N	N	N	N
Branch Unconditional	RR,Ext.Mnemonic							N	N	N	N
Compare Algebraic	RX	x	x					Z	AA	BB	
Compare Algebraic	RR							Z	AA	BB	
Compare Decimal	SS,Decimal	x				x	Data	Z	AA	BB	
Compare Halfword	RX	x	x					Z	AA	BB	
Compare Logical	RX	x	x					Z	AA	BB	
Compare Logical	RX	x	x					Z	AA	BB	
Compare Logical	SS	x						Z	AA	BB	
Compare Logical Immediate	SI	x						Z	AA	BB	
Compare, Long	RX,Floating Pt.	x	x			x		Z	AA	BB	
Compare, Long	RR,Floating Pt.	x	x			x		Z	AA	BB	
Compare, Short	RX,Floating Pt.	x	x			x		Z	AA	BB	
Compare, Short	RR,Floating Pt.		x			x		Z	AA	BB	
Convert to Binary	RX	x	x				Data,F	N	N	N	N
Convert to Decimal	RX	x	x		x			N	N	N	N

Condition Code Set (Add)

Instruction	Mnemonic Operation Code	Machine Operation Code	Operand Format	
			Explicit	Implicit
Divide	D	5D	R1,D2(X2,B2) or R1,D2(,B2)	R1, S2(X2) or R1,S2
Divide	DR	1D	R1,R2	
Divide Decimal	DP	FD	D1,(L1,B1),D2(L2,B2)	S1(L1),S2(L2)or S1,S2
Divide, Long	DD	6D	R1,D2(X2,B2),or R1,D2(,B2)	R1,S2(X2) or R1,S2
Divide, Long	DDR	2D	R1,R2	
Divide, Short	DE	7D	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2) or R1,S2
Divide, Short	DER	3D	R1,R2	
Edit	ED	DE	D1(L,B1),D2(B2)	S1(L),S2 or S1,S2
Edit and Mark	EDMK	DF	D1(L,B1),D2(B2)	S1(L),S2 or S1,S2
Exclusive Or	X	57	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Exclusive Or	XC	D7	D1(L,B1),D2(B2)	S1(L),S2 or S1,S2
Exclusive Or	XR	17	R1,R2	
Exclusive Or Immediate	XI	97	D1(B1),I2	S1,I2
Execute	EX	44	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) R1,S2
Halve, Long	HDR	24	R1,R2	
Halve,Short	HER	34	R1,R2	
Halt I/O	HIO	9E	D1(B1)	
Insert Character	IC	43	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Insert Storage Key	ISK	09	R1,R2	
Load	L	58	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Load	LR	18	R1,R2	
Load Address	LA	41	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Load and Test	LTR	12	R1,R2	
Load and Test, Long	LTDR	22	R1,R2	
Load and Test, Short	LTER	32	R1,R2	
Load Complement	LCR	13	R1,R2	
Load Complement, Long	LCDR	23	R1,R2	
Load Complement, Short	LCER	33	R1,R2	
Load Halfword	LH	48	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Load, Long	LD	68	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Load, Long	LDR	28	R1,R2	
Load Multiple	LM	98	R1,R3,D2(B2)	R1,R3,S2
Load Negative	LNR	11	R1,R2	
Load Negative, Long	LNDR	21	R1,R2	
Load Negative, Short	LNER	31	R1,R2	
Load Positive	LPR	10	R1,R2	
Load Positive, Long	LPDR	20	R1,R2	
Load Positive, Short	LPER	30	R1,R2	
Load PSW	LPSW	82	D1(B1)	
Load, Short	LE	78	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Load, Short	LER	38	R1,R2	
Move Characters	MVC	D2	D1(L,B1),D2(B2)	S1(L),S2 or S1,S2
Move Immediate	MVI	92	D1(B1),I2	S1,I2
Move Numerics	MVN	D1	D1(L,B1),D2(B2)	S1(L),S2 or S1,S2
Move with Offset	MVO	F1	D1(L1,B1),D2(L2,B2)	S1(L1),S2(L2)or S1,S2
Move Zones	MVZ	D3	D1(L,B1),D2(B2)	S1(L),S2 or S1,S2
Multiply	M	5C	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2) or R1,S2
Multiply	MR	1C	R1,R2	
Multiply Decimal	MP	FC	D1(L1,B1),D2(L2,B2)	S1(L1),S2(L2) or S1,S2
Mulitply Halfword	MH	4C	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Multiply, Long	MD	6C	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Multiply, Long	MDR	2C	R1,R2	
Multiply, Short	ME	7C	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Multiply, Short	MER	3C	R1,R2	
No Operation	NOP	47(BC 0)	D2(X2,B2) or D2(,B2)	S2(X2) or S2

Operand Format (Divide)

Instruction	Type of Instruction	Program Interruptions Possible						Condition Code Set			
		A	S	Ov	P	Op	Other	00	01	10	11
Divide	RX	x	x				F	N	N	N	N
Divide	RR		x				F	N	N	N	N
Divide Decimal	SS, Decimal	x	x		x	x	D, Data	N	N	N	N
Divide, Long	RX, Floating Pt.	x	x	E		x	B, E	N	N	N	N
Divide, Long	RR, Floating Pt.		x	E		x	B, E	N	N	N	N
Divide, Short	RX, Floating Pt.	x	x	E		x	B, E	N	N	N	N
Divide, Short	RR, Floating Pt.		x	E		x	B, E	N	N	N	N
Edit	SS, Decimal	x			x	x	Data	S	T	U	
Edit and Mark	SS, Decimal	x			x	x	Data	S	T	U	
Exclusive Or	RX	x	x					J	K		
Exclusive Or	SS	x			x			J	K		
Exclusive Or	RR							J	K		
Exclusive Or Immediate	SI	x			x			J	K		
Execute	RX	x	x				G	(May be set by this instruction)			
Halve, Long	RR, Floating Pt.		x			x		N	N	N	N
Halve, Short	RR, Floating Pt.		x			x		N	N	N	N
Halt I/O	SI						A	DD	CC	GG	KK
Insert Character	RX	x						N	N	N	N
Insert Storage Key	RR	x	x			x	A	N	N	N	N
Load	RX	x	x					N	N	N	N
Load	RR							N	N	N	N
Load Address	RX							N	N	N	N
Load and Test	RR							J	L	M	
Load and Test, Long	RR, Floating Pt.		x			x		R	L	M	
Load and Test, Short	RR, Floating Pt.		x			x		R	L	M	
Load Complement	RR			F				P	L	M	O
Load Complement, Long	RR, Floating Pt.		x			x		R	L	M	
Load Complement, Short	RR, Floating Pt.		x			x		R	L	M	
Load Halfword	RX	x	x					N	N	N	N
Load, Long	RX, Floating Pt.	x	x			x		N	N	N	N
Load, Long	RR, Floating Pt.		x			x		N	N	N	N
Load Multiple	RS	x	x					N	N	N	N
Load Negative	RR							J	L		
Load Negative, Long	RR, Floating Pt.		x			x		R	L		
Load Negative, Short	RR, Floating Pt.		x			x		R	L		
Load Positive	RR			F				J		M	O
Load Positive, Long	RR, Floating Pt.		x			x		R	L	M	
Load Positive, Short	RR, Floating Pt.		x			x		R	L	M	
Load PSW	SI	x	x				A	QQ	QQ	QQ	QQ
Load, Short	RX, Floating Pt.	x	x			x		N	N	N	N
Load, Short	RR, Floating Pt.		x			x		N	N	N	N
Move Characters	SS	x			x			N	N	N	N
Move Immediate	SI	x			x			N	N	N	N
Move Numerics	SS	x			x			N	N	N	N
Move with Offset	SS	x			x			N	N	N	N
Move Zones	SS	x			x			N	N	N	N
Multiply	RX	x	x					N	N	N	N
Multiply	RR		x					N	N	N	N
Multiply Decimal	SS, Decimal	x	x		x	x	Data	N	N	N	N
Multiply Halfword	RX	x	x					N	N	N	N
Multiply, Long	RX, Floating Pt.	x	x	E		x	B	N	N	N	N
Multiply, Long	RR, Floating Pt.		x	E		x	B	N	N	N	N
Multiply, Short	RX, Floating Pt.	x	x	E		x	B	N	N	N	N
Multiply, Short	RR, Floating Pt.		x	E		x	B	N	N	N	N
No Operation	RX, Ext.Mnemonic							N	N	N	N

Condition Code Set (Divide)

Instruction	Mnemonic Operation Code	Machine Operation Code	Operand Format	
			Explicit	Implicit
No Operation	NOPR	07(BCR 0)	R2	
Or Logical	O	56	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Or Logical	OC	D6	D1(L,B1),D2(B2)	S1(L),S2 or S1,S2
Or Logical	OR	16	R1,R2	
Or Logical Immediate	OI	96	D1(B1),I2	S1,I2
Pack	PACK	F2	D1(L1,B1),D2(L2,B2)	S1(L1),S2(L2) or S1,S2
Read Direct	RDD	85	D1(B1),I2	S1,I2
Set Program Mask	SPM	04	R1	
Set System Key	SSK	08	R1,R2	
Set System Mask	SSM	80	D1(B1)	S1
Shift Left Double Algebraic	SLDA	8F	R1,D2(B2)	R1,S2
Shift Left Double Logical	SLDL	8D	R1,D2(B2)	R1,S2
Shift Left Single Algebraic	SLA	8B	R1,D2(B2)	R1,S2
Shift Left Single Logical	SLL	89	R1,D2(B2)	R1,S2
Shift Right Double Algebraic	SRDA	8E	R1,D2(B2)	R1,S2
Shift Right Double Logical	SRDL	8C	R1,D2(B2)	R1,S2
Shift Right Single Algebraic	SRA	8A	R1,D2(B2)	R1,S2
Shift Right Single Logical	SRL	88	R1,D2(B2)	R1,S2
Start I/O	SIO	9C	D1(B1)	S1
Store	ST	50	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Store Character	STC	42	R1,D2(X2,B2) or R1,D2(,B2	R1,D2(X2) or R1,S2
Store Halfword	STH	40	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Store Long	STD	60	R1,D2(X2,B2)	R1,S2(X2) or R1,S2
Store Multiple	STM	90	R1,R2,D2(B2)	R1,R2,S2
Store Short	STE	70	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Subtract	S	5B	R1,D2(X2	R1,S2(X2) or R1,S2
Subtract	SR	1B	R1,R2	
Subtract Decimal	SP	FB	D1(L1,B1),D2(L2,B2)	S1(L1),S2(L2) or S1,S2
Subtract Halfword	SH	4B	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Subtract Logical	SL	5F	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Subtract Logical	SLR	1F	R1,R2	
Subtract Normalized, Long	SD	6B	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Subtract Normalized, Long	SDR	2B	R1,R2	
Subtract Normalized, Short	SE	7B	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Subtract Normalized,	SER	3B	R1,R2	
Subtract Unnormalized, Long	SW	6F	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Subtract Unnormalized, Long	SWR	2F	R1,R2	
Subtract Unnormalized, Short	SU	7F	R1,D2(X2,B2) or R1, D2(,B2)	R1,S2(X2) or R1,S2
Subtract Unnormalized, Short	SUR	3F	R1,R2	
Supervisor Call	SVC	0A	I	
Test and Set	TS	93	D1(B1)	S1
Test Channel	TCH	9F	D1(B1)	S1
Test I/O	TIO	9D	D1(B1)	S1
Test Under Mask	TM	91	D1(B1),I2	S1,I2
Translate	TR	DC	D1(L,B1),D2(B2)	S1(L),S2 orS1,S2
Translate and Test	TRT	DD	D1(L,B1),D2(B2)	S1(L),S2 orS1,S2
Unpack	UNPK	F3	D1(L1,B1),D2(L2,B2)	S1(L1),S2(L2)or S1,S2
Write Direct	WRD	84	D1(B1),I2	S1,I2
Zero and Add Decimal	ZAP	F8	D1(L1,B1),D2(L2,B2)	S1(L1),S2(L2)or S1,S2

Operand Format (No Operation)

Instruction	Type of Instruction	Program Interruptions Possible						Condition Code Set			
		A	S	Ov	P	Op	Other	00	01	10	11
No Operation	RR, Ext. Mnemonic							N	N	N	N
Or Logical	RX	x	x					J	K		
Or Logical	SS	x			x			J	K		
Or Logical	RR							J	K		
Or Logical Immediate	SI	x			x			J	K		
Pack	SS	x			x			N	N	N	N
Read Direct	SI	x			x	x	A	N	N	N	N
Set Program Mask	RR							RR	RR	RR	RR
Set Storage Key	RR	x	x			x	A	N	N	N	N
Set System Mask	SI	x					A	N	N	N	N
Shift Left Double Algebraic	RS		x	F				J	L	M	O
Shift Left Double Logical	RS		x					N	N	N	N
Shift Left Single Algebraic	RS			F				J	L	M	O
Shift Left Single Logical	RS							N	N	N	N
Shift Right Double Algebraic	RS		x					J	L	M	
Shift Right Double Logical	RS		x					N	N	N	N
Shift Right Single Algebraic	RS							J	L	M	
Shift Right Single Logical	RS							N	N	N	N
Start I/O	SI						A	MM	CC	EE	AA
Store	RX	x	x		x			N	N	N	N
Store Character	RX	x			x			N	N	N	N
Store Halfword	RX	x	x		x			N	N	N	N
Store Long	RX, Floating Pt.	x	x		x	x		N	N	N	N
Store Multiple	RS	x	x		x			N	N	N	N
Store Short	RX, Floating Pt.	x	x		x	x		N	N	N	N
Subtract	RX	x	x	F				V	X	Y	O
Subtract	RR			F				V	X	Y	O
Subtract Decimal	SS, Decimal	x		D	x	x	Data	V	X	Y	O
Subtract Halfword	RX	x	x	F				V	X	Y	O
Subtract Logical	RX	x	x						W,H	V,I	W,I
Subtract Logical	RR								W,H	V,I	W,I
Subtract Normalized, Long	RX, Floating Pt.	x	x	E		x	B,C	R	L	M	Q
Subtract Normalized, Long	RR, Floating Pt.		x	E		x	B,C	R	L	M	Q
Subtract Normalized, Short	RX, Floating Pt.	x	x	E		x	B,C	R	L	M	Q
Subtract Normalized, Short	RR, Floating Pt.		x	E		x	B,C	R	L	M	Q
Subtract Unnormalized, Long	RX, Floating Pt.	x	x	E		x	C	R	L	M	Q
Subtract Unnormalized, Long	RR, Floating Pt.		x	E		x	C	R	L	M	Q
Subtract Unnormalized, Short	RX, Floating Pt.	x	x	E		x	C	R	L	M	Q
Subtract Unnormalized, Short	RR, Floating Pt.		x	E		x	C	R	L	M	Q
Supervisor Call	RR							N	N	N	N
Test and Set	SI	x			x			SS	TT		
Test Channel	SI						A	JJ	II	FF	HH
Test I/O	SI						A	LL	CC	EE	KK
Test Under Mask	SI	x						UU	VV		WW
Translate	SS	x			x			N	N	N	N
Translate and Test	SS	x						PP	NN	OO	
Unpack	SS	x			x			N	N	N	N
Write Direct	SI	x				x	A	N	N	N	N
Zero and Add Decimal	SS, Decimal	x		D	x	x	Data	J	L	M	O

Condition Code Set (No Operation)

Program Interruptions Possible

Under Ov:
- D = Decimal
- E = Exponent
- F = Fixed Point

Under Other:

A	Privileged Operation
B	Exponent Underflow
C	Significance
D	Decimal Divide
E	Floating Point Divide
F	Fixed Point Divide
G	Execute

Condition Code Set

H	No Carry
I	Carry
J	Result = 0
K	Result is Not Equal to Zero
L	Result is Less Than Zero
M	Result is Greater Than Zero
N	Not Changed
O	Overflow
P	Result Exponent Underflows
Q	Result Exponent Overflows
R	Result Fraction = 0
S	Result Field Equals Zero
T	Result Field is Less Than Zero
U	Result Field is Greater Than Zero
V	Difference = 0
W	Difference is Not Equal to Zero
X	Difference is Less Than Zero
Y	Difference is Greater Than Zero
Z	First Operand Equals Second Operand
AA	First Operand is Less Than Second Operand
BB	First Operand is Greater Than Second Operand
CC	CSW Stored
DD	Channel and Subchannel not Working
EE	Channel or Subchannel Busy
FF	Channel Operating in Burst Mode
GG	Burst Operation Terminated
HH	Channel Not Operational
II	Interruption Pending in Channel
JJ	Channel Available
KK	Not Operational
LL	Available
MM	I/O Operation Initiated and Channel Proceeding With its Execution
NN	Nonzero Function Byte Found Before the First Operand Field is Exhausted
OO	Last Function Byte is Nonzero
PP	All Function Bytes Are Zero
QQ	Set According to Bits 34 and 35 of the New PSW Loaded
RR	Set According to Bits 2 and 3 of the Register Specified by R1
SS	Leftmost Bit of Byte Specified = 0
TT	Leftmost Bit of Byte Specified = 1
UU	Selected Bits Are All Zeros; Mask is All Zeros
VV	Selected Bits Are Mixed (zeros and ones)
WW	Selected Bits Are All Ones

Program Interruptions Possible

Mnemonic	Name Field	Operand Field
CCW	An optional symbol	Four operands, separated by commas
CNOP	blank	Two decimal terms, separated by a comma
CSECT	An optional symbol	Not used; treated as a comment
DC	An optional symbol	One operand.
DROP	blank	One to five absolute expressions, separated by commas
DS	An optional symbol	One operand
DSECT	A required symbol	Not used; treated as a comment
EJECT	blank	Not used; treated as a comment
END	blank	A relocatable expression or blank
ENTRY	blank	One relocatable symbol
EQU	A required symbol	An absolute or relocatable expression
EXTRN	blank	One relocatable symbol
ICTL	blank	One to three decimal values, separated by commas
ISEQ	blank	Two decimal values, separated by a comma or a blank
LTORG	an optional symbol	Not used; treated as a comment
ORG	blank	A relocatable expression or blank
PRINT	blank	One to three operands
PUNCH	blank	1 to 80 characters enclosed in single quotation marks
REPRO	blank	Not used; treated as a comment
SPACE	blank	A decimal term or blank
START	An optional symbol	A self-defining term or blank
TITLE	0-4 characters	A sequence of characters, enclosed in single quotation marks
USING	blank	An absolute or relocatable expression followed by 1 to 5 absolute expressions, separated by commas
XFR	blank	A relocatable symbol

APPENDIX D: MACHINE-INSTRUCTION FORMAT

Format Code	Instruction Format Showing Bits	Assembler Operand-Field Format	Applicable Instructions	See Notes
RR	Op Code R1 R2 8 4 4	R1,R2	All RR instructions except SPM and SVC	1,6, 8,9
	Op Code R1 R2 8 4 (4)	R1	SPM	
	Op Code I 8 8	I	SVC	
RX	Op Code R1 X2 B2 D2 8 4 4 4 12	R1,D2(X2,B2) R1,S2(X2)	ALL RX instructions	1-4, 7, 9
RS	Op Code R1 R3 B2 D2 8 4 4 4 12	R1,R3,D2(B2) R1,R3,S2	BXH,BXLE,LM, STM	1-3, 7,8
	Op Code R1 R3 B2 D2 8 4 (4) 4 12	R1,D2(B2) R1,S2	All shift instructions	
SI	Op Code I2 B1 D1 8 8 4 12	D1(B1),I2 S1,I2	All SI instructions except LPSW,SSM,HIO, SIO,TIO,TCH,TS	2,3, 6-8
	Op Code I2 B1 D1 8 (8) 4 12	D1(B1) S1	LPSW,SSM,HIO, SIO,TIO,TCH,TS	
SS	Op Code L1 L2 B1 D1 B2 D2 8 4 4 4 12 4 12	D1(L1,B1),D2(L2,B2) S1(L1),S2(L2)	PACK,UNPK,MVO, AP,CP,DP,MP,SP, ZAP	2,3, 5,7
	Op Code L B1 D1 B2 D2 8 8 4 12 4 12	D1(L,B1),D2(B2) S1(L),S2	NC,OC,XC,CLC, MVC,MVN,MVZ,TR, TRT,ED,EDMK	

Notes for Appendix D:

1. R1, R2, and R3 are absolute expressions that specify general or floating-point registers. The general register numbers are 0 through 15; floating-point register numbers are 0, 2, 4, and 6.
2. D1 and D2 are absolute expressions that specify displacements. A value of 0 - 4095 may be specified.
3. B1 and B2 are absolute expressions that specify base registers. Register numbers are 0 - 15.
4. X2 is an absolute expression that specifies an index register. Register numbers are 0 - 15. If B2 is specified, X2 must not be omitted; and when indexing is not desired, X2 must be specified as 0.
5. L, L1, and L2 are absolute expressions that specify field lengths. An L expression can specify a value of 1 - 256. L1 and L2 expressions can specify a value of 1 - 16. In all cases, the assembled value will be one less than the specified value.
6. I and I2 are absolute expressions that provide immediate data. The value of the expression may be 0 - 255.
7. S1 and S2 are absolute or relocatable expressions that specify an address.
8. RR, RS, and SI instruction fields whose bits are enclosed in parentheses are not examined during instruction execution. The fields are not written in the symbolic operand, but are assembled as binary zeros.
9. R1 specifies a 4-bit mask in the BC and BCR machine instructions.

APPENDIX E: HEXADECIMAL-DECIMAL NUMBER CONVERSION TABLE

The table in this appendix provides for direct conversion of decimal and hexadecimal numbers in these ranges:

Hexadecimal	Decimal
000 to FFF	0000 to 4095

Decimal numbers (0000-4095) are given within the 8-part table. The first two characters (high-order) of hexadecimal numbers (000-FFF) are given in the left-hand column of the table, and the third character is arranged across the top of each part of the table. Thus to find the decimal equivalent of the hex number 0C9, look for 0C in the left column, and across under the column labeled 9. The decimal number is 0201.

To convert from decimal to hexadecimal, look up the decimal number within the table and read the hexadecimal number by a combination of the hex characters to the left and above the decimal number. For example, decimal number 123 has the hex equivalent of 07B, and decimal 1478 has the hex equivalent of 5C6.

For numbers outside the range of the table, add the following values to the table figures:

Hexadecimal	Decimal
1000	4096
2000	8192
3000	12288
4000	16384
5000	20480
6000	24576
7000	28672
8000	32768
9000	36864
A000	40960
B000	45056
C000	49152
D000	53248
E000	57344
F000	61440

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
01	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
02	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
03	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
04	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
05	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
06	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
07	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
08	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
09	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255
10	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
11	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
12	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
13	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
14	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
15	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
16	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
17	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
18	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
19	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
20	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
21	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
22	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
23	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
24	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
25	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
26	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
27	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
28	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
29	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767
30	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
31	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
32	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
33	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
34	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
35	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
36	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
37	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
38	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
39	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
40	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
41	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
42	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
43	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
44	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
45	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
46	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
47	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
48	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
49	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
50	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
51	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
52	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
53	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
54	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
55	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
56	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
57	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
58	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
59	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
60	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
61	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
62	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
63	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
64	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
65	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
66	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
67	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663
68	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
69	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791
70	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
71	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
72	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
73	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
74	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
75	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
76	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
77	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
78	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
79	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
80	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
81	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
82	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
83	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
84	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
85	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
86	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
87	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
88	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
89	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303
90	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
91	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
92	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
93	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
94	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
95	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
96	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
97	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
98	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
99	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A0	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A1	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A2	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A3	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A4	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A5	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A6	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A7	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A8	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A9	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
AD	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
B0	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B1	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B2	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B3	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B4	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B5	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B6	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B7	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B8	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B9	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C0	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C1	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C2	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C3	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C4	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C5	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C6	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C7	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C8	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C9	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327
D0	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D1	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D2	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D3	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D4	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D5	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D6	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D7	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D8	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D9	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
E0	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E1	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E2	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E3	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E4	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E5	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E6	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E7	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E8	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E9	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839
F0	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F1	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F2	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F3	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F4	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F5	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F6	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F7	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F8	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F9	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

TYPE	IMPLIED LENGTH (BYTES)	ALIGN-MENT	LENGTH MODI-FIER RANGE	SPECIFIED BY	NUMBER OF CON-STANTS PER OPERAND	RANGE FOR EX-PONENTS	RANGE FOR SCALE	TRUN-CATION/ PADDING SIDE
C	as needed	byte	1 to 256	characters	one			right
X	as needed	byte	1 to 256	hexadecimal digits	one			left
B	as needed	byte	1 to 256	binary digits	one			left
F	4	word	1 to 8	decimal digits	multi-ple	-85 to +75	-187 to +346	left
H	2	half word	1 to 8	decimal digits	multi-ple	-85 to +75	-187 to +346	left
E	4	word	1 to 8	decimal digits	multi-ple	-85 to +75	0-13	right
D	8	double word	1 to 8	decimal digits	multi-ple	-85 to +75	0-13	right
P *)	as needed	byte	1 to 16	decimal digits	multi-ple			left
Z *)	as needed	byte	1 to 16	decimal digits	multi-ple			left
A	4	word	1 to 4	any expression	one			left
V	4	word	3 or 4	relocatable symbol	one			left
S	2	half word	2 only	one absolute or relocatab-le expression or two absol-ute express-ions: exp(exp)	one			
Y	2	half word	1 to 4	any expression	one			left

*) Length modifier range in DS from 1 to 256.

APPENDIX G : IOCS EXAMPLE

This example (Figure 53) illustrates the files and main-storage area assignments for one tape file, one card file, and one disk file. It is a simplified order and inventory job in which a master tape is updated and written onto a disk file, and a card file of detail orders is processed. The following assumptions are made.

- The old master inventory tape contains quantity on hand and unit price in addition to the identifying information.
- The card file reflects quantities ordered. It is to be completed with quantity available for shipment, unit price, and the extension of quantity shipped x unit price.
- The new master inventory disk file reflects the decrease in quantity on hand due to the current orders, or an increase when items are returned.

The illustration shows this setup:

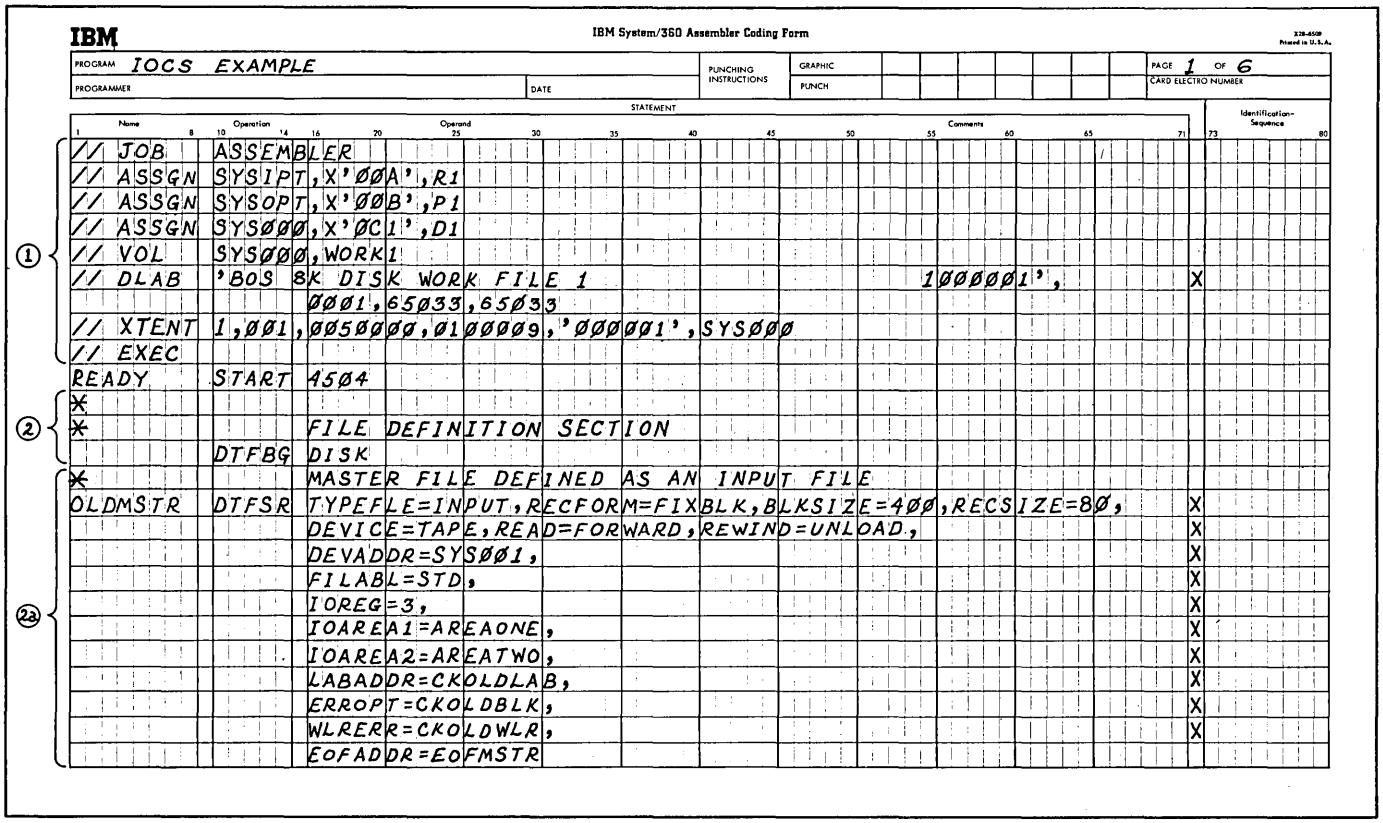
1. Job Control cards to assign devices for assembly. It is assumed that a Supervisor for object program execution is assembled separately.
2. Declarative macro instructions to define the three files. The first definition is preceded by a begin-definition card.
   a. Old master tape file. This is an input file to be read forward. It contains standard volume and file labels and additional user 80-character file labels. It is a card-image file with a blocking factor of 5, and uses two input areas with general purpose register 3 assigned for locating individual records in the input areas.
   b. New master disk file. This is a disk output file with the same characteristics as the tape input file. General purpose register 4 is assigned for locating the next available output-record area.
   c. Detail card file. This is an input file to be updated. It is read at the punch-feed-read station of an IBM 2540 Card Read-Punch. Records are read in, updated, and punched from the same I/O area.
   d. End of the three file definition macro instructions. The user's source program follows this macro instruction.
3. Assembler instructions to define the input and output areas for the three files.
   a. Two input and two output areas are reserved for the master files. Input records, and fields within the records, are defined in a dummy section (DSECT). See part 3c.
   b. One I/O area is reserved for detail card records. Because these are single unblocked records, the individual fields within the records may be defined along with the allocation of the I/O area as shown. Note the use of the zero-duplication factor.
   c. Use of the DSECT (dummy section) assembler instruction is illustrated here. This permits the individual fields within a record to be addressed symbolically. Whenever these fields are used as an operand of an instruction, their length may be implied and their location is expressed as a displacement relative to the contents of the register specified for addressing the dummy section. In this example, register 3 is assigned to the master input file by DTFSR entry IOREG. Therefore, IOCS makes the starting address of the current record available to the problem program in register 3. Because this register is also used to address the dummy section (USING statement in section 6), individual fields can be addressed regardless of which record in the input area(s) is being processed.
4. Assembler instruction to resume coding of original control section. Because the DSECT coding interrupted the control section defined by the START instruction, a CSECT instruction is required to resume the coding of the

original control section. The user may include here the remainder of his storage assignment instructions and any constants, tables, or work areas for calculations.

5. User routines required for processing additional user-standard labels.

6. Sample instructions to open files and locate master records that have current activity. This illustrates the following:

   - All files to be processed by logical IOCS must be opened.
   - A GET for an unblocked record to be processed in a single I/O area causes the record to be physically transferred from the I/O device to main storage . This makes it available to the problem program.
   - The first GET for a record in a blocked file causes the physical transfer of the block(s) of data from the I/O device to the I/O area(s). It also initializes the specified I/O register with the address of the first record. Each succeeding GET causes the address of the currently available record to be placed in the I/O register, and may or may not cause a physical transfer of data.
   - When output files (with labels) specify an I/O register, the register is initialized by the OPEN routine (after label processing). A PUT to an output file with no work area specified merely causes the address of the next available record area to be placed in the specified I/O register. No data is moved within storage, and physical transfer of data may or may not occur.
   - A dummy section is addressed by means of the USING statement, and fields (for example, MITEM) within an individual record that have been defined in the dummy section may be referred to by symbolic names.
   - A record can be moved from an input file to an output file, both of which have I/O registers specified, by the MVC instruction with explicit length and explicit base registers. These base registers must be the same registers as those specified by the IOREG entries in the corresponding file definitions.
   - All files that have been opened must be closed.



IBM

IBM System/360 Assembler Coding Form

X28-6509
Printed in U.S.A.

PROGRAM IOCS EXAMPLE | PUNCHING INSTRUCTIONS | GRAPHIC | PUNCH | PAGE 1 OF 6

PROGRAMMER | DATE | CARD ELECTRO NUMBER

STATEMENT

	Name (1–8)	Operation (10–14)	Operand (16–71)	Comments	Identification-Sequence (73–80)
①	// JOB	ASSEMBLER			
	// ASSGN	SYSIPT,X'ØØA',R1			
	// ASSGN	SYSOPT,X'ØØB',P1			
	// ASSGN	SYSØØØ,X'ØC1',D1			
	// VOL	SYSØØØ,WORK1			
	// DLAB	'BOS 8K DISK WORK FILE 1	1ØØØØØ1',	X	
		ØØØ1,65Ø33,65Ø33			
	// XTENT	1,ØØ1,ØØ5ØØØ,Ø1ØØØØ9,'ØØØØØ1',SYSØØØ			
	// EXEC				
	READY	START 45Ø4			
②	*				
	*		FILE DEFINITION SECTION		
		DTFBG	DISK		
	*		MASTER FILE DEFINED AS AN INPUT FILE		
	OLDMSTR	DTFSR	TYPEFLE=INPUT,RECFORM=FIXBLK,BLKSIZE=4ØØ,RECSIZE=8Ø,	X	
			DEVICE=TAPE,READ=FORWARD,REWIND=UNLOAD,	X	
			DEVADDR=SYSØØ1,	X	
			FILABL=STD,	X	
②			IOREG=3,	X	
			IOAREA1=AREAONE,	X	
			IOAREA2=AREATWO,	X	
			LABADDR=CKOLDLAB,	X	
			ERROPT=CKOLDBLK,	X	
			WLRERR=CKOLDWLR,	X	
			EOFADDR=EOFMSTR		

Figure 53. IOCS Example (Part 1 of 6)

IBM System/360 Assembler Coding Form

PROGRAM IOCS EXAMPLE — PAGE 2 OF 6

```
*
*              MASTER FILE DEFINED AS AN OUTPUT FILE
*
```

(2b)

```
NEWMSTR  DTFSR DEVICE=DISK11,IOAREA1=AREATHRE,                          X
               IOAREA2=AREAFOUR,                                        X
               IOREG=4,                                                 X
               RECFORM=FIXBLK,                                          X
               BLKSIZE=4Ø8,                                             X
               RECSIZE=8Ø,                                              X
               TYPEFLE=OUTPUT,                                          X
               VERIFY=YES
```

(2c)

```
*
*              DETAIL FILE DEFINED AS A COMBINED FILE
*
DETAIL   DTFSR TYPEFLE=CMBND,RECFORM=FIXUNB,BLKSIZE=8Ø,                 X
               DEVICE=READ4Ø,                                           X
               DEVADDR=SYSØØØ,                                          X
               IOAREA1=DETAILCD,                                        X
               EOFADDR=EOFDTAIL
```

(2d)

```
         DTFEN
```

Figure 53. IOCS Example (Part 2 of 6)

IBM System/360 Assembler Coding Form

PROGRAM IOCS EXAMPLE — PAGE 3 OF 6

(3a)

```
*              INPUT / OUTPUT AREAS FOR MASTER FILE USAGE
         USING *,6      ESTABLISH REGISTER FOR ADDRESSING ALL I/O AREAS
AREAONE  DS    CL2ØØ    MASTER FILE INPUT AREA 1 - FIRST 2ØØ POSITIONS
         DS    CL2ØØ    SECOND 2ØØ POSITIONS
AREATWO  DS    CL2ØØ    MASTER FILE INPUT AREA 2 - FIRST 2ØØ POSITIONS
         DS    CL2ØØ    SECOND 2ØØ POSITIONS
AREATHRE DS    CL2Ø8    MASTER FILE OUTPUT AREA 1 - FIRST 2ØØ POSITIONS
         DS    CL2ØØ    SECOND 2ØØ POSITIONS
AREAFOUR DS    CL2Ø8    MASTER FILE OUTPUT AREA 2 - FIRST 2ØØ POSITIONS
         DS    CL2ØØ    SECOND 2ØØ POSITIONS
```

(3b)

```
*              DETAIL FILE INPUT, OUTPUT, AND UPDATING AREA
*                   THE LAST THREE FIELDS OF THE DETAIL CARD WILL BE
*                   CALCULATED AND PUNCHED BY THIS PROGRAM
DETAILCD DS    ØCL8Ø    DETAIL CARD IMAGE
DITEM    DS    CL1Ø     ITEM NUMBER IN STORES CATALOG
         DS    CL15     ITEM DESCRIPTION    - NOT USED IN THIS PROGRAM
         DS    CL11     CUSTOMER NUMBER     - NOT USED IN THIS PROGRAM
         DS    CL11     INVOICE NUMBER      - NOT USED IN THIS PROGRAM
         DS    CL2      LINE NO. ON INVOICE - NOT USED IN THIS PROGRAM
DCODE    DS    CL1      TRANSACTION CODE - ISSUE, RETURN TO STORES, ETC
QUANTITY DS    CL5      QUANTITY ORDERED OR RETURNED
QTYFILLD DS    CL5      QUANTITY TO BE SHIPPED AT THIS TIME -CALCULATED
DUNITCST DS    CL7      UNIT COST OF ITEM -FROM MASTER FILE RECORD
AMOUNT   DS    CL13     EXTENSION OF UNIT COST BY QUANTITY SHIPPED      X
                        NOTE-ON A RETURN PUNCH A CREDIT X IN THIS FIELD
```

Figure 53. IOCS Example (Part 3 of 6)

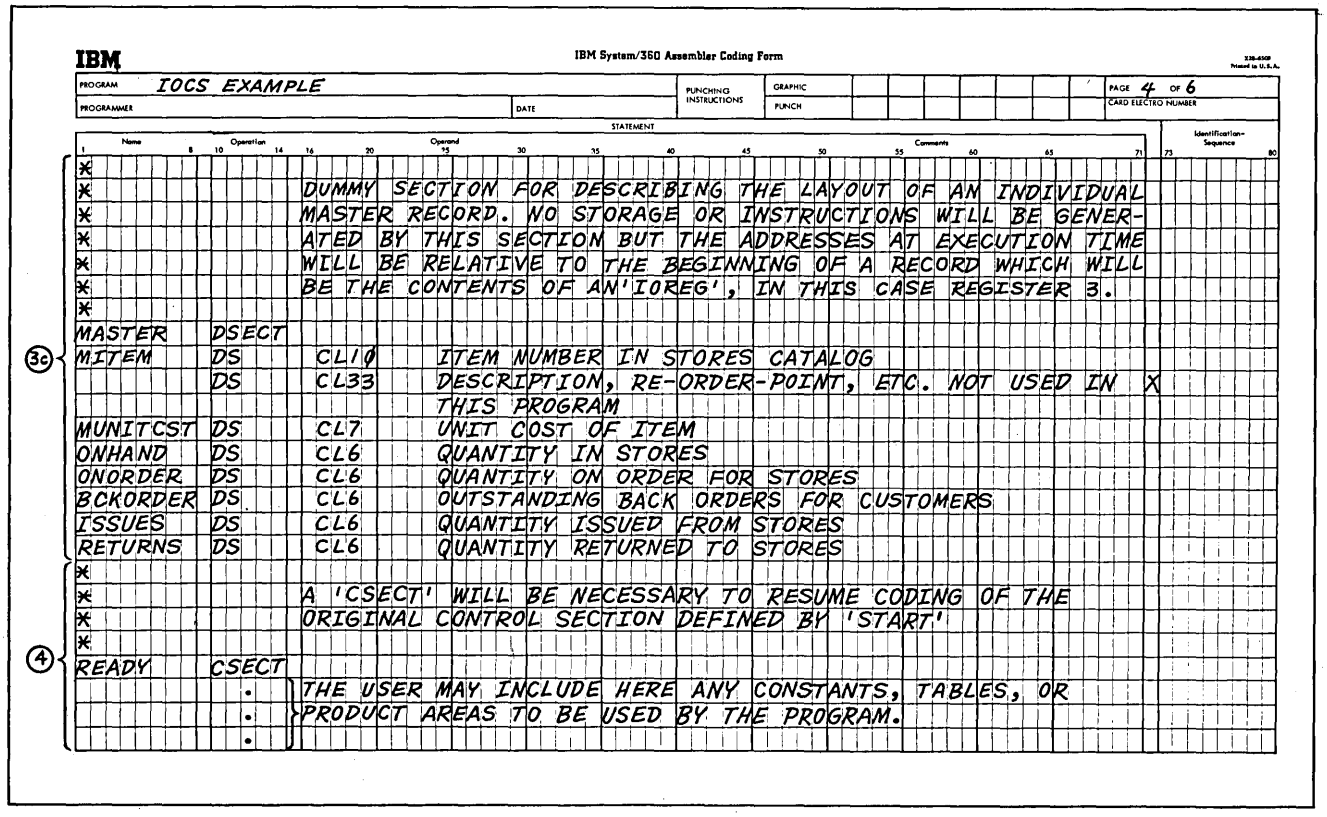


Figure 53. IOCS Example (Part 4 of 6)

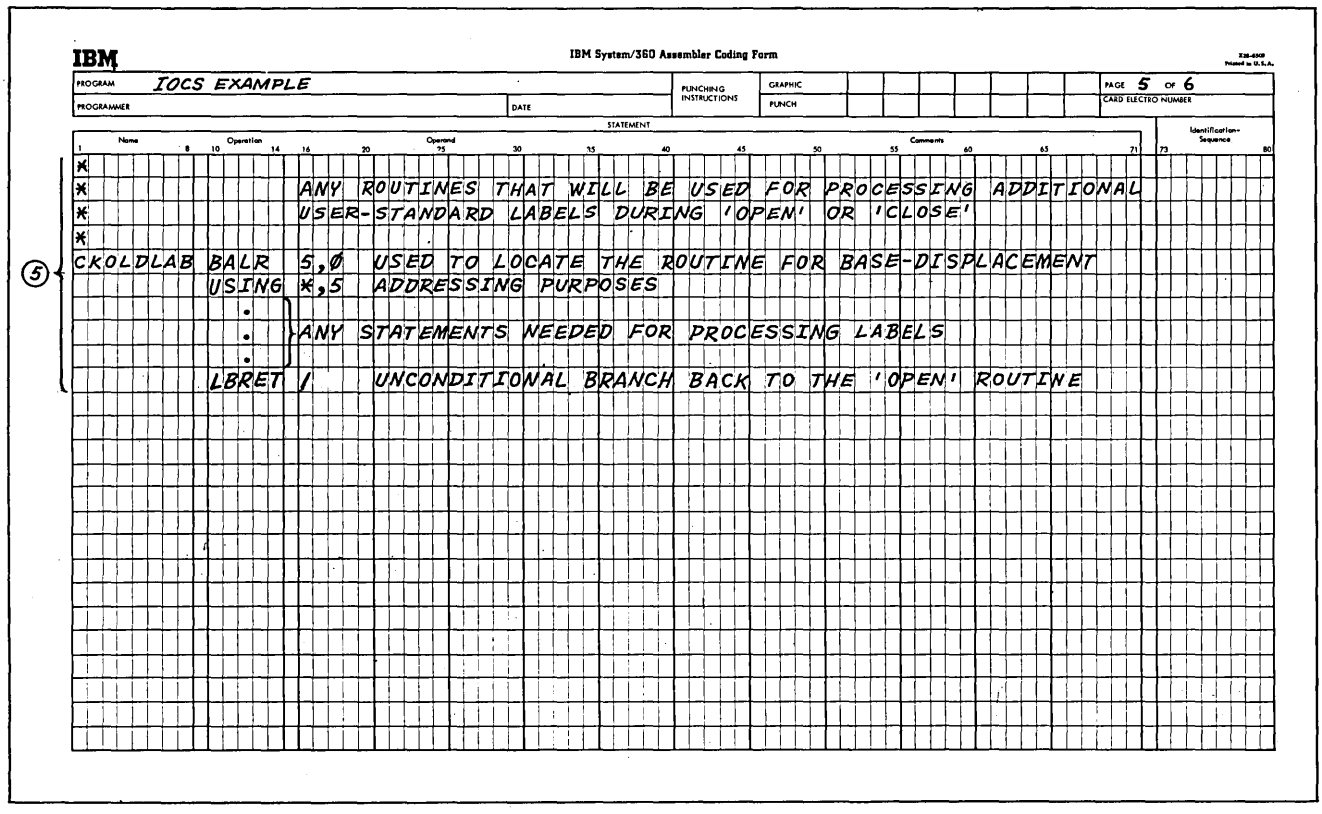


Figure 53. IOCS Example (Part 5 of 6)

IBM System/360 Assembler Coding Form

PROGRAM: IOCS EXAMPLE — PAGE 6 OF 6

(6)

```
*
*              USER'S MAIN-LINE PROGRAM
*
LOCATE   BALR  2,Ø      LOCATE PROGRAM BY LOADING REGISTER 2 WITH LINKAGE
         USING *,2      INFORMATION AND USING THAT AS A BASE REFERENCE
          .    }HOUSEKEEPING, INCLUDING LOADING REGISTER 6 FOR ADDRESS-
          .    }ING I/O AREAS
BEGIN    OPEN  OLDMSTR,NEWMSTR,DETAIL   OPEN FILES AND PROCESS LABELS
GETDTAIL GET   DETAIL     READ IN A DETAIL CARD
GETMSTR  GET   OLDMSTR    LOCATE A MASTER RECORD (USING REGISTER 3)
         USING MASTER,3    USE REGISTER 3 FOR ADDRESSING THE 'MASTER'    X
                               DUMMY SECTION AND FIELDS WITHIN IT
         CLC   MITEM,DITEM     COMPARE MASTER TO DETAIL ON ITEM NUMBER
         BH    ERROR       INDICATES A MISSING MASTER OR NEW DETAIL NO.
         BE    UPDATEM     TO ROUTINE FOR UPDATING MASTERS FROM DETAILS
         MVC   Ø(8Ø,4),Ø(3) OTHERWISE, MOVE MASTER FROM INPUT TO OUTPUT
         PUT   NEWMSTR         AND BUMP REGISTER 4 TO NEXT OUTPUT RECORD
         B     GETMSTR         THEN GET THE NEXT MASTER FOR COMPARING
*
*              END-OF-FILE ROUTINES
*
         CLOSE OLDMSTR,NEWMSTR,DETAIL   CLOSE FILES
         EOJ
         END   LOCATE
```

Figure 53. IOCS Example (Part 6 of 6)

APPENDIX H: ASSEMBLER LANGUAGES -- FEATURES COMPARISON CHART

Features not shown below are common to all assemblers. In the chart:

Dash = Not allowed.
X = as defined in System/360 Operating System Assembler Language Manual.

Feature	BPS/360: Card Basic Assembler	7090/7094 Support Package Assembler	BPS/360 and BOS/360 Assemblers[1]	OS/360 Assembler
No. of Continuation Cards/Statement (exclusive of macro-instructions)	0	0	1	2
Input Character Code	EBCDIC	BCD & EBCDIC	EBCDIC	EBCDIC
ELEMENTS:				
Maximum Characters per symbol	6	6	8	8
Character self-defining terms	1 Char. only	X	X	X
Binary self-defining terms	--	--	X	X
Length attribute reference	--	--	X	X
Literals	--	--	X	X
Extended mnemonics	--	X	X	X
Maximum Location Counter value	$2^{16}-1$	$2^{24}-1$	$2^{24}-1$	$2^{24}-1$
Multiple Control Sections per assembly	--	--	X	X
EXPRESSIONS:				
Operators	+-*	+-*/	+-*/	+-*/
Number of terms	3	16	3	16
Number of parentheses	--	--	1 Level	5 Levels
Complex relocatability	--	--	X	X
ASSEMBLER INSTRUCTIONS:				
DC and DS				
Expressions allowed as modifiers	--	--	--	X
Multiple operands	--	--	--	X
Multiple constants in an operand	--	--	Except Address Consts.	X

(Continued)

Feature	BPS/360: Card Basic Assembler	7090/7094 Support Package Assembler	BPS/360 and BOS/360 Assemblers[1]	OS/360 Assembler
Bit length specifications	--	--	--	X
Scale modifier	--	--	X	X
Exponent Modifier	--	--	X	X
DC types	Except B, P, Z, V, Y, S	Except B, V	X	X
DC duplication factor	Except A	X	Except S	X
DC duplication factor of zero	--	--	Except S	X
DC length modifier	Except H, E, D	X	X	X
DS types	Only C, H, F, D	Only C, H, F, D	X	X
DS length modifier	Only C	Only C	X	X
DS maximum length modifier	256	256	256	65,535
DS constant subfield permitted	--	--	X	X
COPY	--	--	--	X
CSECT	--	--	X	X
DSECT	--	--	X	X
ISEQ	--	--	X	X
LTORG	--	--	X	X
PRINT	--	--	X	X
TITLE	--	X	X	X
COM	--	--	--	X
ICTL	1 oprnd 1 or 25 only	1 oprnd	X	X
USING	2 oprnds oprnd 1 reloc only	2-17 oprnds oprnd 1 reloc only	6 oprnds	X
DROP	1 oprnd only	X	5 oprnds	X

(Continued)

Appendix H: Assembler Languages--Features Comparison Chart (Continued)

Feature	BPS/360: Card Basic Assembler	7090/7094 Support Package Assembler	BPS/360 and BOS/360 Assemblers[1]	OS/360 Assembler
CCW	oprnd 2 reloc only	X	X	X
ORG	no blank oprnd	no blank oprnd	X	X
ENTRY	1 oprnd only	1 oprnd only	1 oprnd only	X
EXTRN	max 14 1 oprnd only	1 oprnd only	1 oprnd only	X
CNOP	2 dec digits	2 dec digits	2 dec digits	X
PUNCH	--	--	X	X
REPRO	--	--	X	X
Macro Instructions	--	--	X	X

[1] Not including Model 20.

(Continued)

Appendix H: Assembler Languages--Features Comparison Chart (Continued)

Macro Language Features	BPS/360 and BOS/360 Assemblers[1]	OS/360 Assembler
Operand Sublists	--	X
Attributes of macro-instruction operands inside macro definitions and symbols used in conditional assembly instructions outside macro definitions.	--	X
Substitution in the operation field	--	X
Subscripted SET symbols	--	X
Maximum number of operands	49	200
Conditional assembly instructions outside macro definitions.	--	X
Maximum number of SET symbols		
global SETA	16	[2]
global SETB	128	[2]
global SETC	16	[2]
local SETA	16	[2]
local SETB	128	[2]
local SETC	0	[2]

[1] Not including Model 20.
[2] The number of SET symbols permitted by the Operating System/360 Assembler is variable, dependent upon available main storage.

This table lists the card groups that make up the source deck produced by the programmer, with an explanation of each group. The groups are listed in the order in which they appear in the source deck.
Note: All job-control cards must enter the system via SYSRDR, all others via SYSIPT. The same device may be assigned to both SYSRDR and SYSIPT.

CARD GROUP	CONTENTS	REMARKS
Job Control Cards	// JOB ASSEMBLER	First card in group (always required) Assemble-and-execute is initiated when name of problem program appears as the second operand.)
	// DATE	Required on first job after IPL.
	// ASSGN	See Figure 50 for device assignments. (Only those assignments not already in effect are required.)

	// VOL
	// DLAB
	// XTENT
		Required for SYS000 (all assemblies). See Figure 56 for operands.

	// VOL
	// DLAB
	// XTENT
		Required for SYS001 (when the AWORK 2 option is used).

	// EXEC	Last card in group (always required).
Assembler Control Cards	ALOG AOPTN AWORK AFILE	See Control Cards section for explanation and format. These cards may appear in any order.
Supervisor-Assembly Source Cards	Supervisor-assembly macro statements.	Either a BOS or an independent Supervisor may be assembled.
Problem-Program Source Cards	Assembler, Machine, and Macro statements.	START card may be omitted if a problem program is assembled alone, (START 0 assumed). START card must be omitted if a problem program and a supervisor are assembled together.
End Card	END	Last card of source deck.

Figure 54. Assembler Source Deck

Symbolic Unit	Function and Device
	REQUIRED DEVICE ASSIGNMENTS
SYSRES	System residence device. IBM 2311 Disk Storage Drive.
SYSRDR	Job-control input device. May be the same device as SYSIPT. IBM 1442, 2520, or 2540 Card Read Punch, or 2501 Card Reader.
SYSIPT	Source program input device. May be the same device as SYSRDR, IBM 1442, 2520 (Model A1 or B1), or 2540 Card Read Punch, or 2501 Card Reader. IBM 2400-series Magnetic Tape Unit (7- or 9-track) may be used. (If the data-conversion feature was used to prepare the 7-track tape, it must also be used to read the tape.) Information appearing on tape must be 80-byte unblocked records.
SYSLST	Program listing device. IBM 1403, 1404 (continuous forms only), or 1443 Printer. IBM 2400-series Magnetic Tape Unit (9-track, or 7-track with or without the data-conversion feature) may be used. Listing on tape appears as 121-character print images (a single forms-control byte followed by a 120-character line image). (This forms-control byte is the command code portion of the CCW used during printout.)
SYS000	Used for temporary work area during assembly. IBM 2311 Disk Storage Drive. May be same device as SYSRES.
	OPTIONAL DEVICE ASSIGNMENTS
SYSLOG	Operator message logging device. IBM 1052 Printer-Keyboard, or IBM 1403, 1404 (continuous forms only), or 1443 Printer.
SYSOPT	Object program output device. IBM 1442, 2520, or 2540 Card Read Punch. IBM 2400-series Magnetic Tape Unit (9-track, or 7-track with data-conversion feature) may be used. Output on tape appears as 80-byte unblocked records. Not required when using assemble-and-execute option.
SYS001	Used for temporary work area during assembly. IBM 2311 Disk Storage Drive. Reduces assembly time by providing additional work area on a separate disk drive.

Note 1: When an IBM 2404 Magnetic Tape Unit and Control is used, all tape drives must be assigned to the same channel.

Note 2: For any of the above device assignments, the Supervisor must contain the corresponding error routines.

Figure 55. Device Assignments

Symbolic Unit	Function and Device
	REQUIRED DEVICE ASSIGNMENTS
SYSRES	System residence device. IBM 2311 Disk Storage Drive.
SYSRDR	Job-control input device. May be the same device as SYSIPT. IBM 1442, 2520, or 2540 Card Read Punch, or 2501 Card Reader.
SYSIPT	Source program input device. May be the same device as SYSRDR, IBM 1442, 2520 (Model A1 or B1), or 2540 Card Read Punch, or 2501 Card Reader. IBM 2400-series Magnetic Tape Unit (7- or 9-track) may be used. (If the data-conversion feature was used to prepare the 7-track tape, it must also be used to read the tape.) Information appearing on tape must be 80-byte unblocked records.
SYSLST	Program listing device. IBM 1403, 1404 (continuous forms only), or 1443 Printer. IBM 2400-series Magnetic Tape Unit (9-track, or 7-track with or without the data-conversion feature) may be used. Listing on tape appears as 121-character print images (a single forms-control byte followed by a 120-character line image). (This forms-control byte is the command code portion of the CCW used during printout.)
SYS000	Used for temporary work area during assembly. IBM 2311 Disk Storage Drive. May be same device as SYSRES.
	OPTIONAL DEVICE ASSIGNMENTS
SYSLOG	Operator message logging device. IBM 1052 Printer-Keyboard, or IBM 1403, 1404 (continuous forms only), or 1443 Printer.
SYSOPT	Object program output device. IBM 1442, 2520, or 2540 Card Read Punch. IBM 2400-series Magnetic Tape Unit (9-track, or 7-track with data-conversion feature) may be used. Output on tape appears as 80-byte unblocked records. Not required when using assemble-and-execute option.
SYS001	Used for temporary work area during assembly. IBM 2311 Disk Storage Drive. Reduces assembly time by providing additional work area on a separate disk drive.

Note 1: When an IBM 2404 Magnetic Tape Unit and Control is used, all tape drives must be assigned to the same channel.

Note 2: For any of the above device assignments, the Supervisor must contain the corresponding error routines.

Figure 55. Device Assignments

This table lists the Disk-label cards, and their contents.		
Card	Field	Contents
Required for all assemblies		
VOL	Symbolic Unit	SYS000
	File Name	WORK1
DLAB	File Name	BOS 8K DISK WORK FILE 1
	Format Identifier	1
	File Serial Number	Required
	Volume Sequence Number	0001
	Creation Date	Today's Date
	Expiration Date	Today's Date
	System Code	Optional
XTENT	Extent Type	1
	Extent Sequence Number	000
	Lower Limit of Extent	(Define the area, on the disk pack assigned
	Upper Limit of Extent	to SYS000, to be used by the assembler).
	Volume Serial Number	Must be the same as DLAB File Serial Number.
	Symbolic Unit	SYS000
Required only when the AWORK 2 option is used.		
VOL	Symbolic Unit	SYS001
	File Name	WORK2
DLAB	File Name	BOS 8K DISK WORK FILE 2
	Format Identifier	1
	File Serial Number	Required
	Volume Sequence Number	0001
	Creation Date	Today's Date
	Expiration Date	Today's Date
	System Code	Optional
XTENT	Extent Type	1
	Extent Sequence Number	000
	Lower Limit of Extent	(Define the area, on the disk pack assigned
	Upper Limit of Extent	to SYS001, to be used by the assembler).
	Volume Serial Number	Must be same as DLAB File Serial Number.
	Symbolic Unit	SYS001

Figure 56. Disk Label Cards

This table lists the card groups that make up the output deck produced by the assembler, and the conditions of their assembly. The groups are listed in the order in which they appear in the output deck.
Note: No output deck will be produced when NODECK appears in AOPTN.

Card Group	Remarks
IPL Loader	Used for loading an independent supervisor from a card reader. Produced when IPL appears in AOPTN.
Reproduced Cards	These reproduced cards result from REPRO or PUNCH instructions located before START.
Symbol Table	Produced when PCHSYM appears in AOPTN.
External Symbol Dictionary (ESD)	Not produced when NOESD appears in AOPTN.
Supervisor	Either a BOS or an independent supervisor may be produced.
Program Loader	Produced as part of the independent supervisor.
Problem Program	Consists of TXT, XFR, and reproduced cards. The reproduced cards result from REPRO or PUNCH instructions located after START.
Relocation Dictionary (RLD)	Produced if relocatable constants are present, except when NORLD appears in AOPTN.
End Card	Produced as the last card of the output deck. (Next to last if an Entry card is produced.)
Entry Card	Produced as the last card of the output deck when ENTRY appears in AOPTN.

Figure 57. Assembler Output Deck

The information in each card is in EBCDIC--Extended Binary Coded Decimal Interchange Code.

For each card, the first column indicates the numbers of the columns to be punched. The second column indicates the information to be punched.

ESD Card	
1	Multiple punch (12-2-9). Identifies this as a loader card.
2-4	ESD--External Symbol Dictionary card.
11-12	Number of bytes of information contained in this card.
15-16	External symbol identification number (ESID) of the first SD, PC, or ER on this card. Relates the SD, PC, or ER to a particular control section.
17-72	Variable information. 8 positions. Name. 1 position. Type code: 0=SD, 1=LD, 2=ER, or 4=PC. 3 positions. Assembled origin. 1 position. Blank. 3 positions. Control section length, if an SD-type or a PC-type. If an LD-type, this field contains the external symbol identification number (ESID) of the SD or PC containing the label.
73-76	Program identification taken from the name field of the first TITLE statement before the START card.
77-80	Sequence number starting with 0001.

TXT Card	
1	Multiple punch (12-2-9). Identifies this as a loader card.
2-4	TXT--Text card.
6-8	Assembled origin (address of first byte to be loaded from this card).
11-12	Number of bytes of text to be loaded.
15-16	External symbol identification number (ESID) of the control section (SD) containing the text.
17-72	Up to 56 bytes of text--data or instructions to be loaded.
73-76	Program identification taken from the name field of the first TITLE statement before the START card.
77-80	Sequence number starting with 0001.

RLD Card	
1	Multiple punch (12-2-9).
2-4	RLD--Relocation Dictionary card.
11-12	Number of bytes of information contained in the card.
17-72	Variable information (multiple items). 2 positions. Pointer to the relocation factor of the contents of the load constant. 2 positions. Pointer to the relocation factor of the control sections in which the load constant occurs. 1 position. Flag indicating type of constant. 3 positions. Assembled address of load constant.
73-76	Program identification taken from the name field of the first TITLE statement before the START card.
77-80	Sequence number starting with 0001.

Figure 58. Format of Assembler Output Cards (Part 1 of 2)

END Card	
1	Multiple punch (12-2-9).
2-4	END
6-8	Assembled origin of the label supplied to the Assembler in the END card (optional).
15-16	ESID number of the control section to which this END card refers.
17-22	Symbolic label supplied to the Assembler if this label was not defined within the assembly.
73-76	Program identification taken from the name field of the first TITLE statement before the START card.
77-80	Sequence number starting with 0001.

XFR Card	
1	Multiple punch (12-2-9).
2-4	XFR. Transfer card.
6-8	Assembled origin of entry point (after the program is loaded, it receives control at this point).
15-16	External symbol identification number (ESID) of the control section in which the transfer occurs.
17-22	Symbolic label of the entry point.
73-76	Program identification taken from the name field of the first TITLE statement before the START card.
77-80	Sequence number starting with 0001.

SYM Card	
1	Multiple punch (12-2-9).
2-4	SYM
11-12	Number of bytes of information contained in this card.
14-16	External symbol identification number (ESID) of the control section (SD) containing the text.
17-72	Variable information. 12 columns 8 positions - symbol name. 1 position - type identification (machine or assembler instruction other than EQU, DC, or DS). 3 positions - Value attribute (the displacement within the CSECT). or 17 columns 8 positions - symbol. 1 position - type identification (EQU, DC, or DS). 3 positions - Value attribute (the displacement within the CSECT). 1 position - constant type. 1 position - length (one byte less than the constant). 3 positions - multiplicity.
73-76	Program identification taken from the name field of the first TITLE statement preceding the START card.
77-80	Sequence number starting with 0001.

Figure 58. Format of Assembler Output Cards (Part 2 of 2)

Field	Print Positions	Meaning
External Symbol Dictionary (ESD)		
SYMBOL	01-08	External label.
TYPE	11-12	Symbol type. SD - Section Definition (named control section) LD - Label Definition (from ENTRY statement) ER - External Reference (from EXTRN statement or V-type constant) PC - Private Code (unnamed control section)
ID	15-16	ESD entry number.
ADDR	18-23	Address of symbol before relocation.
LENGTH	25-30	Length attribute of symbol.
LD ID	34-35	ESD entry number of control section where label appears.
Instructions		
LOCATN	01-06	Location of assembled instruction (hexadecimal).
OBJECT CODE	08-21, or 08-23	Assembled instruction (hexadecimal). Constant generated (hexadecimal).
WRN	03-05	Flag: Possible error in previous statement.
SEQ	08-10	Flag: Sequence error in previous statement.
ERR	13-15	Flag: Error in previous statement.
CAT	18-20	Flag: Catastrophic error in previous statement.
ADDR 1	23-28	Effective address of 1st operand (hexadecimal).
ADDR 2	29-34	Effective address of 2nd operand (hexadecimal).
STMNT	36-39	Statement number (decimal).
SOURCE STATEMENT	41-120	Source statement (card image). Column 40 contains an asterisk if a source statement is generated.
Relocation Dictionary (RLD)		
POS.ID	03-04	ESD ID number of control section containing the constant.
REL.ID	10-11	ESD ID number of control sections containing the address in the constant.
FLGS	15-16	Type of constant.
ADDR	18-23	Address of constant before relocation.

Figure 59. Assembler Output Listing (Part 1 of 2)

Field	Print Positions	Meaning
Diagnostics		
STATEMENT NO.	05-08	Listing sequence number of statement in error.
ERROR MESSAGES	20-79	Explanation of error
ACTION	80-120	Action taken by assembler.
Table of Defined Symbols		
SYMBOL	01-08 65-72	Source Label
LEN	11-13 75-77	Length Attribute (decimal)
VALUE	15-20 79-84	Value Attribute (hexadecimal)
TYPE	24 88	Type Attribute (decimal)
Cross-Reference List (Replaces Table of Defined Symbols when CROSSREF appears in AOPTN.)		
SYMBOL	01-08	Source Label
LEN	10-13	Length Attribute (decimal)
VALUE	15-20	Value Attribute (hexadecimal)
DEF*	22-26	Listing sequence number of statement which defines label.
CROSS-REFERENCE*	30-34 36-40 42-46 48-52 54-58 60-64 66-70 72-76 78-82 84-88 90-94 96-100 102-106 108-112 114-118	Listing sequence numbers of statements which contain the label. * The maximum number of symbols and cross references to symbols that can be listed is 20,625.

Figure 59. Assembler Output Listing (Part 2 of 2)

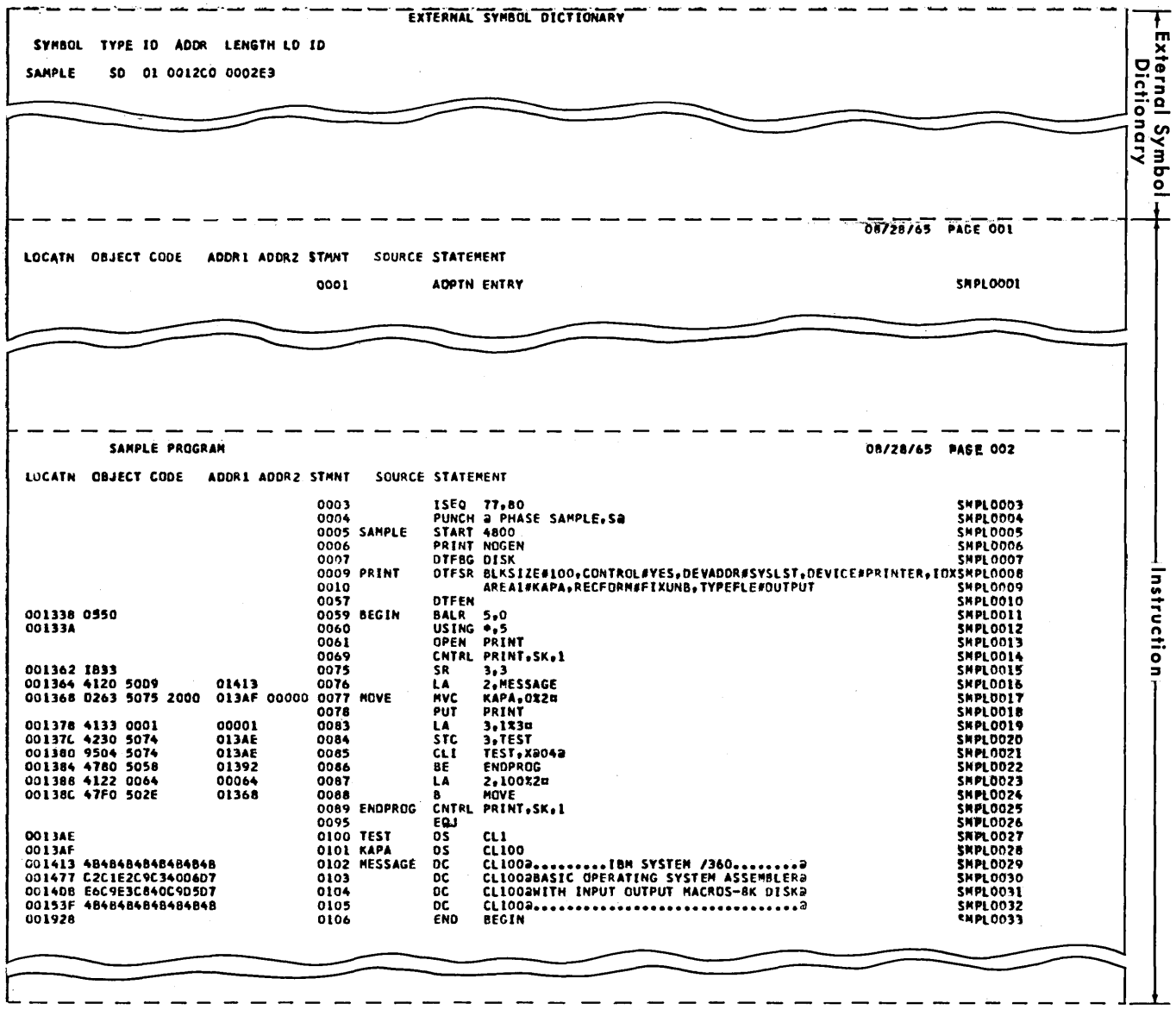


Figure 60. Example of an Assembler Listing (Part 1 of 2)

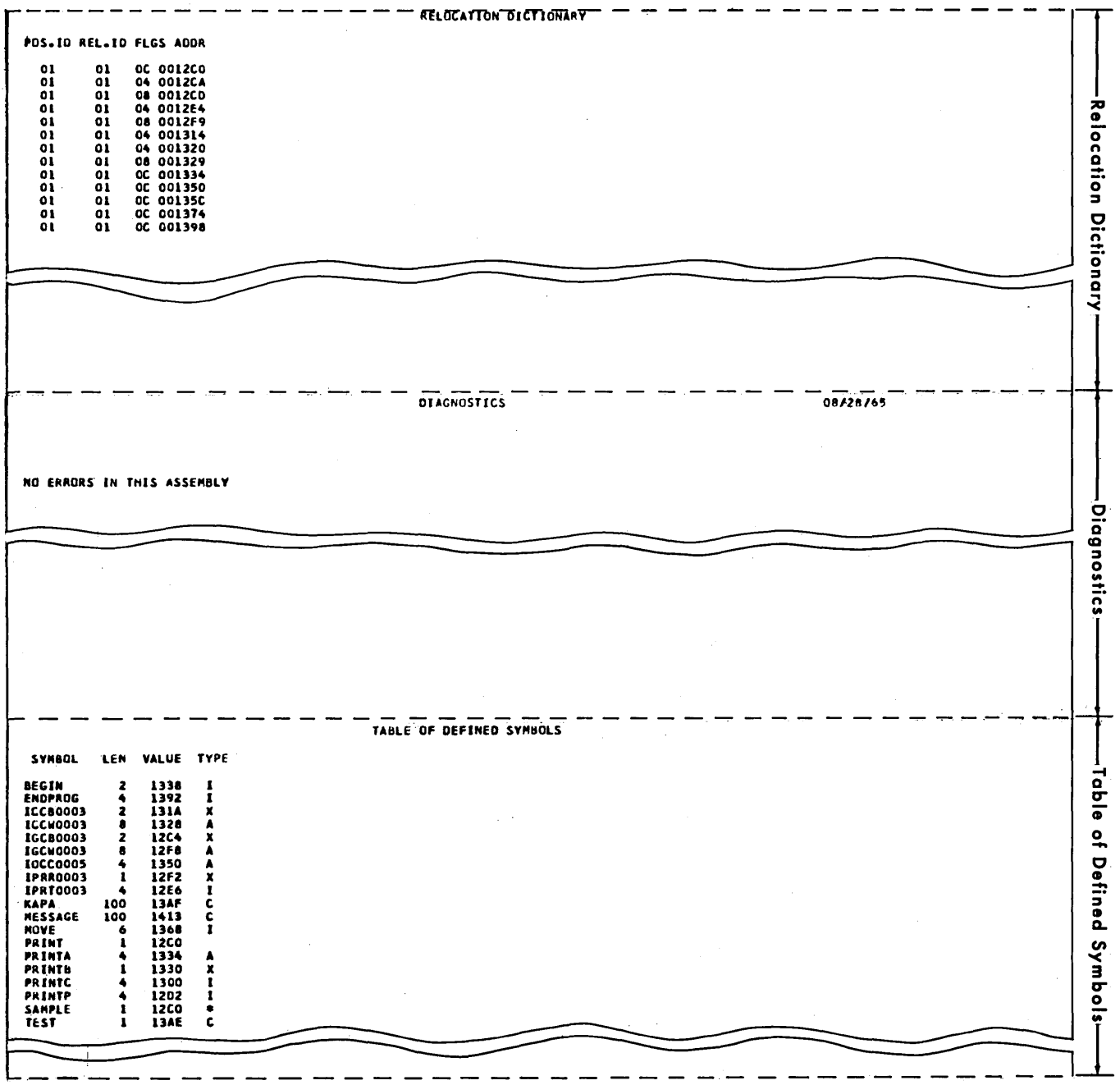


Figure 60. Example of an Assembler Listing (Part 2 of 2)

The following is a listing of the card groups required for the execution of an object program when a disk-resident system is not used. Card groups produced by the Assembler in addition to those listed here are the External Symbol Dictionary (ESD) and the Relocation Dictionary (RLD), which are used only if a Linkage Editor run is required before execution.

Card Groups (in order of loading)	Logic of Execution of the Object Program
IPL Loader	The operator initiates the loading of the IPL Loader with the load key on the console.
Supervisor Program Loader	The IPL Loader loads the Supervisor and the Program Loader, and then transfers control to the Program Loader.
First Problem Program; End Card	The Program Loader loads the First Problem Program, and then transfers control to it.
Data (if present in card form); Data End Card	The First Problem Program runs to end-of-job, Sharing control with the Supervisor.
Second Problem Program; End Card	When the First Problem Program reaches end-of-job, control is transferred back to the Program Loader which loads the next Problem Program and transfers control to it
Data (if present in card form) : : (Any one or several of these Problem Programs may be a Job Control program. See Operating Considerations.) : : Last Problem Program; End Card	
Data (if present in card form); Data End Card	When the hopper becomes empty, an error message will be transmitted to the operator, and the system will enter the wait state.

If the operator has more Problem Programs or data cards to run, he must place these cards in the hopper and press the interruption key to resume the operation.

If another Supervisor is to be used, that Supervisor and a Program Loader must be loaded together by the IPL Loader.

If the execution of a Problem Program causes the Program Loader to be overlaid or otherwise disturbed, the Supervisor and the Program Loader must be reloaded together by an IPL Loader before the next Problem Program can be executed.

Figure 61. The Object Run When a Disk-Resident System Is Not Used

Diagnostic messages are printed if violations of the assembler language are detected by the Assembler program. With each message is printed the number of the related source program statement, and the associated action taken by the Assembler program (Figure 62). If more than one error is detected for a source statement, the diagnostic messages for all errors detected are listed. In this case, the Assembler takes the action that is the most severe (for the errors detected), and that action is printed with the diagnostic messages.

The following list gives the types of assembler actions in the order of their severity. The most severe action is given first.

ASSEMBLY IN ERROR (AIE) - The user's program cannot be executed. The assembly runs to completion, but only the diagnostic messages preceding the ASSEMBLY IN ERROR message may be considered valid.

GENERATION TERMINATED - Generation of one macro is terminated. This does not affect the generation of other macros.

STATEMENT TREATED AS COMMENTS (STC)

STATEMENT INCOMPLETELY ASSEMBLED (SIA) - If the statement is a machine instruction, the amount of main storage necessary for the instruction is reserved and filled with zeros.

STATEMENT ASSEMBLED (SA)

Diagnostic Message	Meaning	Associated Action	Notes
ASSEM CTL CD-INVALID NAME FIELD	An AOPTN, AWORK, or an ALOG card has been detected with an entry in the name field. The name field was ignored, but the rest of the card has been processed.	SA	
ASSEM CTL CD-INVALID OR MISSING OPERAND IN CONTROL CARD	An AOPTN card was given with no operands or with an operand function which does not equal one of the valid options that can be specified. The invalid operand is ignored and any other operands present are processed.	SIA	
BITS 37 THROUGH 39 IN CCW ARE NON-ZERO	Bits 37, 38, and 39 in a CCW statement are not zero.	SA	
CONSTANT TRUNCATED	Specified constant length is less than the actual constant length.	SA	
ILLEGAL FORMAT	1. Invalid delimiter. 2. Missing field in CCW statement. 3. Extra field(s) appear in statement operand. 4. First operand is a literal. 5. More than one literal operand encountered. 6. Blank operand in a machine instruction or in a CCW, DROP, ICTL, USING, or XFR assembler instruction.	SIA	

Figure 62. Assembler Diagnostic Messages (Part 1 of 7)

Diagnostic Message	Meaning	Associated Action	Notes
ILLEGAL MODIFIER IN DC	1. Incorrect modifier or modifier value in a DC statement. 2. Incorrect combination of exponent modifiers. 3. Exponent and/or scale cannot be applied to the constant provided.	SIA	
IMMEDIATE DATA INVALID	The immediate operand in an SI instruction is: 1. Not a self-defining term. 2. An invalid self-defining term (e.g., more than one byte)	SIA	
IMPROPER START VALUE	The value in a START statement is not a multiple of eight. (The value given is increased to the next higher multiple of eight).	SA	
INVALID ASSEM CTL CD	1. The operand of an AWORK card is not a single operand of either 1 or 2. 2. The name field of an AFILE card is blank or contains ALL. 3. The operand field of an AFILE card is not LIBRARY or LIBRARY, RETAIN.	STC	
INVALID BOUNDARY ALIGNMENT	Instruction is not aligned on the proper boundary, ie, for use with an execute instruction.	SA	
INVALID EXPRESSION	1. For all instructions except CCW, DC, ORG, EQU, and USING: the value of the expression is negative. 2. The expression has more than three terms. 3. The expression is complexly relocatable, but it is not allowed for this instruction. 4. The terms in a multiply or divide expression are not absolute. 5. An arithmetic operator begins or ends an expression.	SIA	
INVALID LENGTH VALUE	For SS type instructions only. 1. Length is negative. 2. Length is greater than 256 3. For two-length instructions: a. L1 or L2 is negative. b. L1 or L2 is greater than 16. 4. For multiply and divide instructions: a. L1 or L2 is negative, b. L1 greater than 16, or L2 greater than 8. c. L2 is greater than L1.	SIA	

Figure 62. Assembler Diagnostic Messages (Part 2 of 7)

Diagnostic Message	Meaning	Associated Action	Notes
INVALID LITERAL SPECIFICATION	1. A literal appears in the first operand of an instruction. 2. The literal is incorrect. (The rules for literals are identical to the rules for DC constants.) 3. Literal operand has no left parenthesis or apostrophe.	SIA	
INVALID NAME FIELD	1. For all instructions except TITLE: The statement name begins with a non-alphabetic character. ($, @, and # are considered alphabetic characters.) 2. The statement name is longer than eight characters. 3. Nonalphameric characters appear within the statement name. 4. A statement name is present in a statement which may not have a name. 5. A DSECT statement has no name.	SA	
INVALID OCCURRENCE OF ASSEMBLER OPERATION	1. Program has more than one START card. 2. A START card is improperly placed in the program. 3. A LTORG appears within a dummy control section.	STC	
INVALID REGISTER IN USING OR DROP	1. Register specified in a USING or DROP statement is other than 0 to 15. 2. Register 0 is specified in a USING statement with a non-zero absolute value. 3. Register 0, when used with a USING statement, is not the first register specified.		
INVALID REGISTER SPECIFICATION	1. Register specified is other than 0 to 15. 2. Floating point register specified must be 0, 2, 4, or 6. 3. Base register specified in a relocatable operand. 4. In a USING or DROP statement, a relocatable expression appears in an R1, R2, R3, R4, or R5 field.	SIA	
INVALID SELF DEFINING TERM	The self-defining term: 1. is too large. 2. is too long. 3. contains an invalid character.	SIA	

Figure 62. Assembler Diagnostic Messages (Part 3 of 7)

Diagnostic Message	Meaning	Associated Action	Notes
LIMIT EXCEEDED	1. The value of the location counter has exceeded $2^{24}-1$ 2. The value of the location counter set by the ORG statement has gone below the initial value of the control section. 3. The total number of CSECT, DSECT, EXTRN, and DC (V-type) statements exceeds 255. 4. The total number of CSECT and DSECT statements exceeds 32.	AIE	
MACRO-INNER MACRO NESTING DEPTH EXCEEDED	An inner macro instruction has been given within a third level macro. (This macro will not be expanded.)	STC	1, 2
MACRO INST-INVALID OPERAND	1. Operand longer than 8 characters. 2. Operand has an equal sign in other than the first position.	STC	
MACRO INST-TOO MANY OPERANDS	More than 49 operands in a macro instruction. The extra operands are ignored.	SIA	
MACRO INST-UNDEFINED KEYWORD	Keyword in macro instruction does not match any keyword defined in the prototype. This operand is ignored; all other operands are processed. NOTE: Only one message appears when more than one undefined keyword appears in the same card of a macro instruction.	SIA	
MACRO-INVALID DATA IN ARITHMETIC OPERATION	Non-numeric character encountered in an AIF, AIFB, or SETB statement with an arithmetic relation or a SETA statement.	STC	1,2
MACRO-INVALID RESULT IN ARITHMETIC OPERATION	For an AIF, AIFB, or SETB statement with an arithmetic relation or a SETA statement: 1. Result is negative. 2. Result is greater than $2^{24}-1$.	STC	1,2
MACRO-INVALID SUBSTRING	Specified substring not wholly contained in the character string of a SETC instruction.	STC	1, 2
MACRO-INVALID SYSLST REFERENCE	SYSLST reference to a parameter number greater than 49 was encountered.	STC	1, 2
MACRO-LONG FINAL RESULT IN A CHAR. OPERATION	Character string result has exceeded eight characters in an AIF, AIFB, or SETB statement with a character relation or a SETC statement.	STC	1,2

Figure 62. Assembler Diagnostic Messages (Part 4 of 7)

Diagnostic Message	Meaning	Associated Action	Notes
MACRO-LONG INTERMEDIATE RESULT IN A CHAR. OPERATION	Character string has exceeded sixteen characters for an AIF, AIFB, or SETB statement with a character relation or a SETC statement.	STC	1, 2
MACRO SEQUENCE SYMBOL NOT FOUND	No sequence symbol found in a macro-definition for an AGO, AGOB, AIF, or AIFB statement.	generation terminated	1, 2
MACRO-UNDEFINED OP CODE	An instruction with an operation code which is not recognized as a valid System/360 operation code, and is not contained in the macro library, was found during macro generation.	STC	2
MNOTE (plus a message contained in the macro definition)	A message from the macro coder to the macro user, generally identifying an error in the macro instruction.	None	
MULTIPLE DEFINITION	1. Identical symbols appear in the name fields of two or more statements. 2. For an EXTRN statement: a. Operand is identical to the name field of another statement. b. Two or more statements have identical operands 3. The name fields of a CSECT and a DSECT statement are identical. The statement encountered second is considered unnamed.	SA	
NOT ADDRESSABLE	1. Base register(s) specified in USING statement(s) can not be applied. 2. A base register has been specified in a relocatable operand. 3. An absolute displacement is: a. Negative. b. Greater than 4095. 4. No base register specified.	SIA	
OPERATION TERMINATED ON UPSI BIT 7	Operator intervention due to loop during macro generation (UPSI bit 7 was turned on). AGOB, AIFB, and inner macro calls are not processed.	STC	1, 2

Figure 62. Assembler Diagnostic Messages (Part 5 of 7)

Diagnostic Message	Meaning	Associated Action	Notes
STATEMENT FORMAT CANNOT BE ANALYZED	1. Invalid term in a CNOP statement. 2. Erroneous operand in an ICTL, ISEQ, ORG, or EQU statement. 3. Blank operand in a DC, DS, EXTRN, or ENTRY statement. 4. For a DC or DS: a. First character in operand field is not alphabetic, numeric, or an apostrophe. b. No apostrophe or alphabetic character follows the duplication factor. c. No terminating apostrophe. d. Terminating apostrophe followed by a non-blank character. e. Length specified is non-numeric. f. Duplication factor is longer than eight decimal digits. g. Invalid constant type. h. Explicit length greater than 256 or equal to 0. i. Explicit length for a V-type constant is 1 or 2. 5. DC or literal has no fourth subfield. 6. Parentheses are not paired. 7. In a DC or CCW statement: a. A field is missing. b. Bad data is given. 8. No initial apostrophe or no operand in a PUNCH or TITLE statement.	STC	
SYMBOL NOT PREVIOUSLY DEFINED	A symbol in the operand of an ORG or EQU statement is not defined in a previously encountered instruction.	STC	
TOO MANY CONTINUATION CARDS	An instruction (other than a macro instruction) has more than one continuation card.	STC	
TOO MANY REGISTERS SPECIFIED IN USING OR DROP STATEMENT	More than five registers are specified in a USING or DROP statement.	SIA	
UNDEFINED OPERATION CODE	Mnemonic operation code is not recognized as a valid IBM System/360 operation code, and is not contained in the macro library.	STC	
UNDEFINED SYMBOL	A symbol has been referenced but it is not defined in the name field of any instruction.	SIA	

Figure 62. Assembler Diagnostic Messages (Part 6 of 7)

Diagnostic Message	Meaning	Associated Action	Notes
UNPAIRED AMPERSAND	Odd number of ampersands encountered in a constant. (Two ampersands must be specified for every ampersand wanted in a constant.)	SA	

NOTES:

1. These messages refer to statements in the assembly which contain additional information. This additional information is: the macro name, the operation code of the instruction in error within the generated macro, and a pointer to that instruction.

2. Macro diagnostic messages may be caused by improper coding of a macro-definition, or by improper data in the macro instruction (for example, alphabetic characters supplied for a length specification).

Figure 62. Assembler Diagnostic Messages (Part 7 of 7)

Macro Format		
Input/Output Control Macros		
Name	Operation	Operand
	BCLOS	dtfname
	BOPEN	(See Binary Synchronous Communication for operands to be used with the BOPEN macro.)
blockname	CCB	SYSnnn,command-list-name,X'yyyy'
	CDCNV	type,startaddr,length
	CHKPT	n,restart-name,SYSnnn,DISK
	CHNG	SYSnnn
	CLOSE	filename filename1,filename2,filename3,...
	CNTRL	filename,code,n,m
		filename,SEEK
		dtfname,code,n
		dtfname,code
	DSPLY	filename,r,r
	DIALO	7See Processing with STR Devices for the operand that may be used with the DIALO macro.)
	ENDFL	filename
	ERRPT	dtfname
	ESETL	filename
	EXCP	blockname
	FEOV	filename
	GET	filename filename,workname
		filename,,IS filename,workname,IS
	IDIAL	(See Binary Synchronous Communication for operands to be used with the IDIAL macro.)
	LBRET	1 2
	OPEN	filename filename1,filename2,....

Name	Operation	Operand
	PRTOV	filename,n,routine-name
	PUT	filename filename,workname
		filename,,IS filename,workname,IS
	RDLNE	filename
	READ	filename,KEY filename,ID
		filename,KEY,IS
		filename,OR,{name / (r)}
		dtfname,STR
		dtfname,BSC,type-code
	RELSE	filename
	RESCN	filename,r,r,n,F
	SCLOS	dtfname
	SETFL	filename
	SETL	filename,BOF filename,KEY filename,idname
	SOPEN	(See Processing with STR Devices for the operands that may be used with the SOPEN macro.)
	TRUNC	filename
	WAIT	blockname
	WAITF	filename
	WAITM	blockname1,blockname2,... blocknamen,reg
	WRITE	filename,KEY filename,ID filename,RZERO filename,AFTER
		filename,NEWKEY,IS
		filename,KEY,IS
		dtfname,STR
		dtfname,BSC,type-code

Supervisor Communication Macros		
Name	Operation	Operand
	COMRG	
	DUMP	
	EOJ	
	EXCP	blockname
	EXIT	TR CR
	FETCH	name
	MSG	code,REPLY
	MVCOM	byte,n,location
	STXIT	n,pc-name,it-name,oc-name
Supervisor Assembly Macros		
	IOCFG	keyword=YES,keyword=YES,...
	SEND	n,REP
	SUPVR	DISK=YES,CONFG=nnnnnnnn,TR=YES,CR=YES,SAVEREG=yes CONFG=nnnnnnnn,TR=YES,CR=YES,CHKPT=YES
	SYMUN	n,X'ccuu',X'ddss',X'ccuu',X'ddss'
Job Control Assembly Macros		
	JBCTL	
	RSTRT	

	BOS/BPS STR	7702 BINARY	7702 BCD	7711 BINARY	7711 BCD	1978 BINARY	1978 BCD	1013	1009 BINARY	1009 BCD
Transmits Blank	ROXN	ROXN	INVALID	ROXN	INVALID	ROXN	2480	2480	ROXN	2480
Transmits Substitute Blank	2480	2480	2480	2480	2480	2480	INVALID	INVALID	2480	2480
Tranmits and receives intermed. LRC	YES or NO	NO	NO	NO	NO	YES	YES	YES	NO	YES
Receives ROXN	Blank	Blank	INVALID	Blank	Subst. Blank	Blank	INVALID	INVALID	Blank	INVALID
Receives 2480	Subst. Blank	Subst. Blank	Subst. Blank	Subst. Blank	Subst. Blank	Blank. Blank	Blank	Blank	Subst. Blank	Blank

	Card Code	7 bit Tape Code (Even Parity)	7 bit Tape Code (Odd Parity)
Blank	No Punches	Invalid	C bit
Subst. Blank	2-8 Punches	C-A Bits	A Bit

The 7702, 7711, 1978, and 1009, all in binary mode, handle blank and substitute blank compatibly with BCS/BPS STR support.

THE CDCNV macro instruction may be modified to transmit a blank as 2-4-8-0, or to receive a 2-4-8-0 as a blank. This modification may be used to provide compatibility with the 7702, 1009, and 1978 in BCD, and the 1013.

To modify CDCNV, type E, for transmitting a blank as 2-4-8-0, execute the following instruction before converting the first record:

```
MVI IOCTBLE+64,X'2E
```

To modify CDCNV, type F, for receiving a 2-4-8-0 as a blank, execute the following instruction before converting the first record:

```
MVI IOCTBLF+46,X'40'
```

The CDCNV macro expansion provides the table for translation on the first expansion of a particular type. Therefore, if the problem program uses types E and F, modified and not modified, in the same program, the byte in question must be modified before each CDCNV (type E or F) used.

APPENDIX M: BINARY SYNCHRONOUS COMMUNICATION

PART 1 - EBCDIC SPECIAL CHARACTER TABLE (IOBSCT)

CHARACTER	HEXADECIMAL REPRESENTATION	LENGTH IN BYTES	MEANING	PROVIDED BY *
ACK-0	1070	2	Even-odd positive alternating acknowledgments	IOCS (pr.pr.)
ACK-1	1061	2		
ENQ	2D	1	Inquiry--requests permission to send	IOCS (pr.pr.)
EOT	37	1	End-of-transmission character	IOCS
DLE STX	1002	2	Start-of-transparent-text sequence	pr.pr.
DLE EOT	1037	2	Disconnect sequence	IOCS
WABT	107F	2	Wait-before-transmitting sequence	IOCS
DLE ETX	1003	2	End-of-transparent-text sequence	IOCS
DLE ETB	1026	2	End-of-transparent-block sequence	IOCS
STX EOT	0237	2	Positive acknowledgment but signifies inability to continue	pr.pr.
DLE ENQ	102D	2	Disregard this record	IOCS
NAK	3D	1	Negative acknowledgment - requests retransmission	IOCS
SOH	01	1	Start-of-header character	pr.pr.

*pr.pr. - This character is supplied by the problem program as a part of the text message.

(pr.pr.) - This character is included by the problem program after the last ID-character in each ID-sequence.

IOCS-BSC support (macro routines or error recovery procedures) supplies this character.

PART 2 -- SUMMARY OF CHECKS TO BE MADE AT I/O COMPLETION

CHECK	LOCATION	IF ON, CHECK FURTHER FOR	LOCATION
Normal Completion	BSC flag byte I--COMPLETION bit 0	No further checking is necessary unless a READ operation--then check EOT or DLE EOT received (which are normal completions).	BSC FLAGS byte II--RECEIVED flags--bit 6
I/O Error	BSC flag byte I--COMPLETION bit 1	Lost data (READ macros)	TRANSMISSION FLAGS--byte II (dtfname + 7) bit 0
		Bus-out check	bit 1
		Intervention required	bit 3
		Time out	bit 6
		Unit exception*	bit 5
Message Format Error	BSC flag byte I--COMPLETION bit 2	This bit alone: 1. unrecognizable response characters 2. text messages with invalid text-framing characters	------ ------
		Invalid ID (IDIAL only)	BSC flag byte I COMPLETION bit 5
		Invalid ACK (WRITE macros or (DIAL)	BSC flag byte I COMPLETION bit 7
		Unexpected response and a bit in byte II--RECEIVED flags (EOT, WABT, etc.)**	BSC flag byte I COMPLETION bit 6 and byte II RECEIVED flags

* Set by PIOCS to indicate unit exception on a WRITE command, which occurred because the CPU attempted to transmit while in receive mode. This normally is a contention situation, occurring when both CPUs attempt to transmit at the same time. Because the line has probably been "cleared" by a READ command with the skip flag on, issued by PIOCS, a retry of the failing macro will probably be successful.

** If an EOT is received as an unexpected response to a WRITE macro on a switched line, the transmitting CPU (which issued the WRITE macro) should issue a CNTRL EOT macro. This allows the remote (receiving) CPU to issue CNTRL ENQ and to transmit.

PART 3 - SAMPLE PROGRAM

The BOS/BSC sample program (Figures 63 and 64) illustrates line control procedures for a leased line and error checking after the completion of BSC macros. The listings are self-explanatory.

Figure 63: 80-character records are read from a card reader and sent to the remote CPU (Figure 63). At EOF on the reader, EOT is sent to the remote CPU, signifying end-of-transmission. READ TQ (inquiry) is issued to prepare to receive records from the remote CPU. The 80-character records received are printed. At end-of-job, the ERRPT macro is issued to display the error statistics and transmission count.

Figure 64: CNTRL Prepare is issued to prepare to receive records from the remote CPU. The 80 characters received are printed. When EOT is received, CNTRL ENQ is issued. 80-character records are read from a card reader and sent to the remote CPU (Figure 63). At end-of-job, the ERRPT macro is issued to display the error statistics and transmission count.

```
// JOB ASSEMBLER
// DATE 67130
// VOL          SYS000,WORK1
// DLAB         'BOS 8K DISK                                    111111',  *
               0001,65099,65099,'0000000000000'
// XTENT        1,000,0116000,0195009,'111111',SYS000
// EXEC
          AOPTN ENTRY
          REPRO
          PHASE TESTA,S
          TITLE 'SAMPLE PROGRAM - CPU-A'
          START X'1F08'
*
*  FILE DEFINITION SECTION--CARD READER AND PRINTER
*
          DTFBG DISK
RDFILE    DTFSR BLKSIZE=80,DEVADDR=SYSIPT,DEVICE=READ40,EOFADDR=EOT,     *
               IOAREA1=RDCARD,TYPEFLE=INPUT
WTFILE    DTFSR BLKSIZE=80,DEVADDR=SYSLST,DEVICE=PRINTER,IOAREA1=RDCARD,*
               TYPEFLE=OUTPUT
          DTFEN
*
*  BSC FILE DEFINITION SECTION
*
BASE      DTFRF BEGIN,3
BSCFILE   DTFBS DEVADDR=SYS002,AREA=WTAREA,LENGTH=WTLEN,BSCFLAG=ERRFLG, *
               RCOUNT=7
*
*  INITIALIZATION--OPEN'S AND BOPEN
*
BEGIN     BALR  REG6,0
          USING *,REG6
          OPEN  RDFILE,WTFILE
          BOPEN DTFNAME=BSCFILE,INTRFC=B,DIAL=NO
          WAIT  BSCFILED
          TM    ERRFLG,X'80'          TEST FOR NORMAL COMPLETION OF BOPEN
          BZ    ERPT                  NO---ERRPT EXIT  YES---CONTINUE
*
*  SET UP BSC CCB FOR WRITE (AND READ) OPERATION
*
          L     REG5,THREE            SET UP REGISTER FOR RETRY LOOP
          LA    REG4,SDCARD           SET UP AREA ADDRESS FIELD OF CCB
          ST    REG4,WTAREA                WITH DATA AREA ADDRESS
          LH    REG4,LENGTH           SET UP LENGTH FIELD OF CCB WITH
          STH   REG4,WTLEN                 DATA LENGTH
*
*  REQUEST PERMISSION OF REMOTE CPU TO SEND (CNTRL ENQ)
*
ENQSND    CNTRL BSCFILE,ENQ
          WAIT  BSCFILED
          TM    ERRFLG,X'80'          TEST FOR NORMAL COMPLETION OF CNTRL
          BO    PROCEDE               YES---CONTINUE  NO---
          TM    ERRFLG,X'22'          TEST FOR MESSAGE FORMAT ERROR AND
*                                           UNEXPECTED RESPONSE
          BO    CHK1                  BOTH CONDITIONS--CONTINUE CHECKING
          LA    REG7,ENQSND           SET UP RETURN ADDRESS FOR TIME OUT
```

Figure 63. (Part 1 of 3)

```
         B     ERROR                ANY OTHER--ERROR EXIT
CHK1     TM    ERRFLG+1,X'08'       WAS UNEXPT. RESP. AN ENQ(CONTENTION)
         BO    READ                 YES--TRY A READ  NO--
         TM    ERRFLG+1,X'02'       WAS THE UNEXPT. RESP. AN EOT
         BO    ENDOJ                YES---END OF JOB  NO---
         TM    ERRFLG+1,X'01'       WAS THE UNEXPT. RESP. A WABT
         BZ    ERROR1
         BCT   REG5,ENQSND          YES---RETRY THE CNTRL ENQ
         B     EWABT                STILL GETTING WABT---WABT EXIT
*
*  GET A RECORD FROM THE CARD READER--SEND IT TO CPU-B (REMOTE)
*
PROCEDE  L     REG5,THREE           RESET REGISTER FOR RETRY LOOP
         TM    PGMSW,X'01'          WAS THE CNTRL ENQ REISSUED BY THE
*                                        WRITE ROUTINE
         BO    SNDCD                YES--SKIP GETTING A NEW RECORD
RDACD    GET   RDFILE               GET A RECORD FROM THE CARD READER
SNDCD    WRITE BSCFILE,BSC,TT       SEND IT---WRITE-TYPE IS 'CONTINUE'
         WAIT  BSCFILED
         TM    ERRFLG,X'80'         TEST FOR NORMAL COMPLETION OF WRITE
         BO    RDACD                YES--GET ANOTHER RECORD TO SEND  NO-
         TM    ERRFLG,X'22'         TEST FOR MESSAGE FORMAT ERROR AND
*                                        UNEXPECTED RESPONSE
         BO    CHK2                 BOTH CONDITIONS--CONTINUE CHECKING
         LA    REG7,SNDCD           SET UP RETURN ADDRESS FOR TIMEOUT
         B     ERROR                ANY OTHER--ERROR EXIT
CHK2     TM    ERRFLG+1,X'02'       WAS THE UNEXPT. RESP. AN EOT
         BO    ENDOJ                YES--END OF JOB  NO---
         TM    ERRFLG+1,X'01'       WAS THE UNEXPT. RESP. A WABT
         BZ    ERROR1               NO---ERROR EXIT  YES--
         OI    PGMSW,X'01'          SET SWITCH TO RESEND THIS RECORD
         BCT   REG5,ENQSND          SEND ENQ AGAIN
*
*  END-OF-FILE---SEND EOT (CNTRL EOT) AND PREPARE TO RECEIVE ENQ
*  (CNTRL PREPARE)
*
EOT      CNTRL BSCFILE,EOT
         WAIT  BSCFILED
         TM    ERRFLG,X'80'         TEST FOR NORMAL COMPLETION OF CNTRL
         BZ    ERROR1               NO--ERROR EXIT  YES--
PREPARE  READ  BSCFILE,BSC,TQ
         WAIT  BSCFILED
         TM    ERRFLG,X'80'         TEST FOR NORMAL COMPLETION OF CNTRL
         BO    READ                 YES--CONTINUE  NO--
         TM    ERRFLG,X'22'         TEST FOR MESSAGE FORMAT ERROR AND
*                                        UNEXPECTED RESPONSE
         BO    CHK3                 BOTH CONDITIONS--CONTINUE CHECKING
         LA    REG7,PREPARE         SET UP RETURN ADDRESS FOR TIMEOUT
         B     ERROR                ANY OTHER--ERROR EXIT
CHK3     TM    ERRFLG+1,X'02'       WAS THE UNEXPT. RESP. AN EOT
         BO    ENDOJ                YES--NORMAL END OF JOB
         B     ERROR1               NO--ERROR EXIT
*
*  RECEIVE A RECORD AND PRINT IT USING SAME DATA AREA AND LENGTH
*
READ     READ  BSCFILE,BSC,TN       RECEIVE A RECORD--READ-TYPE CONTINUE
```

Figure 63. (Part 2 of 3)

```
         WAIT  BSCFILED
         TM    ERRFLG,X'80'          TEST FOR NORMAL COMPLETION OF READ
         LA    REG7,READ             SET UP RETURN ADDRESS FOR TIMEOUT
         BZ    ERROR                 NO--ERROR EXIT  YES--
         TM    ERRFLG+1,X'02'        TEST FOR EOT RECEIVED WITH NORMAL
*                                          COMPLETION
         BO    ENDOJ                 YES--NORMAL END OF JOB
         PUT   WTFILE                NO--PRINT THE RECORD
         B     READ                  RECEIVE ANOTHER RECORD
*
*  ERROR EXIT
*
ERROR    TM    ERRFLG,X'40'          TEST FOR I/O ERROR
         BZ    ERROR1                NO--EXIT  YES--
         TM    BSCFILED+3,X'02'      TEST FOR TIMEOUT
         BO    TIMEOUT               YES--GO TO ROUTINE  ELSE
ERROR1   MSG   ERR                   ISSUE INDICATIVE MESSAGE
         B     ENDOJB                SEND AN EOT INDICATING END OF JOB
*
*  TIME OUT
*
TIMEOUT  TM    PGMSW,X'10'           HAS TIMEOUT BEEN RETRIED
         BO    *+10                  YES--EXIT
         OI    PGMSW,X'10'           NO--TURN ON SWITCH INDICATING RETRY
         BR    REG7                  RETRY
         MSG   TIME
         B     ENDOJB
*
*  WABT EXIT
*
EWABT    MSG   WABT                  ISSUE INDICATIVE MESSAGE
         B     ENDOJB                GO TO END OF JOB
*
*  NORMAL END-OF-JOB EXIT
*
ENDOJ    MSG   EOJ                   ISSUE EOJ MESSAGE
ENDOJB   EQU   *
         CLOSE RDFILE,WTFILE         CLOSE THE FILES
         BCLOS BSCFILE
         WAIT  BSCFILED
ERPT     ERRPT BSCFILE
*                                    PRINT THE ERROR STATISTICS
         EOJ
*
*
*  REGISTER EQUATES
*
REG4     EQU   4                     WORK REGISTER
REG5     EQU   5                     COUNT-CONTROLLED LOOP REGISTER
REG6     EQU   6                     BASE REGISTER
REG7     EQU   7                     TIMEOUT RETURN ADDRESS
*
*  CONSTANTS
*
THREE    DC    F'3'                  LOOP COUNT
SDCARD    DC    X'02'                STX
RDCARD   DS    CL80                  DATA AREA
         DC    X'03'                 ETX
LENGTH   DC    H'82'                 DATA LENGTH
PGMSW    DC    X'00'                 PROGRAM SWITCH--CONTROLS EXIT FROM
*                                          WABT LOOP
*
         END   BASE
// END
```

Figure 63. (Part 3 of 3)

```
// JOB ASSEMBLER
// DATE 67130
// VOL          SYS000,WORK1
// DLAB         'BOS 8K DISK                                    1111111',  *
               0001,65099,65099,'0000000000000'
// XTENT        1,000,0116000,0195009,'111111',SYS000
// EXEC
         AOPTN ENTRY
         REPRO
         PHASE TESTA,S
         TITLE 'SAMPLE PROGRAM - CPU-B'
         START X'1F08'
*
*  FILE DEFINITION SECTION--PRINTER AND CARD READER
*
         DTFBG DISK
RDFLE    DTFSR BLKSIZE=80,DEVADDR=SYSIPT,DEVICE=READ40,EOFADDR=ENDOJ,  *
               IOAREA1=WTCARD,TYPEFLE=INPUT
WTFLE    DTFSR BLKSIZE=80,DEVADDR=SYSLST,DEVICE=PRINTER,IOAREA1=WTCARD,*
               TYPEFLE=OUTPUT
         DTFEN
*
*  BSC FILE DEFINITION SECTION
*
BASE     DTFRF BEGIN,3
BSCFLE   DTFBS DEVADDR=SYS002,AREA=RDAREA,LENGTH=RDLEN,BSCFLAG=ERRFG,  *
               RCOUNT=7
*
*  INITIALIZATION--OPEN'S AND BOPEN
*
BEGIN    BALR  REG6,0
         USING *,REG6
         OPEN  RDFLE,WTFLE
         BOPEN DTFNAME=BSCFLE,INTRFC=B,DIAL=NO
         WAIT  BSCFLED
         TM    ERRFG,X'80'          TEST FOR NORMAL COMPLETION OF BOPEN
         BZ    ERPT                 NO---ERRPT EXIT, YES---CONTINUE
*
*  SET UP BSC CCB FOR READ (AND WRITE) OPERATION
*
         L     REG5,THREE           SET UP REGISTER FOR RETRY LOOP
         LA    REG4,STCARD          SET UP AREA ADDRESS FIELD OF CCB
         ST    REG4,RDAREA               WITH DATA AREA ADDRESS
         LH    REG4,LENGTH          SET UP LENGTH FIELD OF CCB WITH
         STH   REG4,RDLEN                DATA LENGTH
*
*  PREPARE TO READ BY RECEIVING ENQ FROM REMOTE CPU
*
PREPARE  CNTRL BSCFLE,PRP
         WAIT  BSCFLED
         TM    ERRFG,X'80'          TEST FOR NORMAL COMPLETION OF CNTRL
         BO    READ                 YES--CONTINUE  NO--
         TM    ERRFG,X'22'          TEST FOR MESSAGE FORMAT ERROR AND
*                                        UNEXPECTED RESPONSE
         BO    CHCK1                BOTH CONDITIONS--CONTINUE CHECKING
         LA    REG7,PREPARE         SET UP RETURN ADDRESS FOR TIMEOUT
```

Figure 64. (Part 1 of 3)

```
         B     ERROR              ANY OTHER--ERROR EXIT
CHCK1    TM    ERRFG+1,X'02'      WAS THE UNEXPT. RESP. AN EOT
         BO    ENQSND             YES--GO TO WRITE
         B     ERROR1             NO---ERROR EXIT
*
*  RECEIVE A RECORD AND PRINT IT
*
READ     READ  BSCFLE,BSC,TN      RECEIVE A RCD FROM CPU-A---READ-TYPE
*                                      CONTINUE
         WAIT  BSCFLED
         TM    ERRFG,X'80'        TEST FOR NORMAL COMPLETION OF READ
         LA    REG7,READ          SET UP RETURN ADDRESS FOR TIMEOUT
         BZ    ERROR              NO--ERROR EXIT  YES--
         TM    ERRFG+1,X'02'      TEST FOR EOT RECEIVED WITH NORMAL
*                                      COMPLETION
         BO    ENQSND             YES--BEGIN SENDING RECORDS
         PUT   WTFLE              NO--PRINT THE RECORD
         B     READ               RECEIVE ANOTHER RECORD
*
*  REQUEST PERMISSION OF REMOTE CPU TO SEND (CNTRL ENQ)
*
ENQSND   CNTRL BSCFLE,ENQ
         WAIT  BSCFLED
         TM    ERRFG,X'80'        TEST FOR NORMAL COMPLETION OF CNTRL
         BO    PROCEDE            YES--CONTINUE  NO--
         TM    ERRFG,X'22'        TEST FOR MESSAGE FORMAT ERROR AND
*                                      UNEXPECTED RESPONSE
         BO    CHCK2              BOTH CONDITIONS--CONTINUE CHECKING
         LA    REG7,ENQSND        SET UP RETURN ADDRESS FOR TIMEOUT
         B     ERROR              ANY OTHER--ERROR EXIT
CHCK2    TM    ERRFG+1,X'08'      WAS UNEXPT. RESP. AN ENQ(CONTENTION)
         BO    ENDOJQ             YES--END THE JOB  NO--
         TM    ERRFG+1,X'02'      WAS THE UNEXPT. RESP. AN EOT
         BO    ENDOJ              YES--NORMAL END OF JOB  NO--
         TM    ERRFG+1,X'01'      WAS THE UNEXPT. RESP. A WABT
         BZ    ERROR1             NO--ERROR EXIT
         BCT   REG5,ENQSND        YES--RETRY THE CNTRL ENQ
         B     EWABT              STILL GETTING WABT--WABT EXIT
*
*  GET A RECORD FROM THE CARD READER--SEND IT TO CPU-A (REMOTE)
*
PROCEDE  L     REG5,THREE         RESET REGISTER FOR RETRY LOOP
         MVI   STCARD,X'02'       RESTORE THE STX CHARACTER
         TM    PGMSW,X'01'        WAS THE CNTRL ENQ REISSUED BY THE
*                                      WRITE ROUTINE
         BO    SNDCD              YES--SKIP GETTING A NEW RECORD
RDACD    GET   RDFLE              GET A RECORD FROM THE CARD READER
SNDCD    WRITE BSCFLE,BSC,TT      SEND IT---WRITE-TYPE IS 'CONTINUE'
         WAIT  BSCFLED
         TM    ERRFG,X'80'        TEST FOR NORMAL COMPLETION OF WRITE
         BO    RDACD              YES--GET ANOTHER RECORD TO SEND  NO-
         TM    ERRFG,X'22'        TEST FOR MESSAGE FORMAT ERROR AND
*                                      UNEXPECTED RESPONSE
         BO    CHCK3              BOTH CONDITIONS--CONTINUE CHECKING
         LA    REG7,SNDCD         SET UP RETURN ADDRESS FOR TIMEOUT
         B     ERROR              ANY OTHER--ERROR EXIT
```

Figure 64. (Part 2 of 3)

```
CHCK3    TM    ERRFG+1,X'02'       WAS THE UNEXPT. RESP. AN EOT
         BO    ENDOJ               YES--END OF JOB  NO--
         TM    ERRFG+1,X'01'       WAS THE UNEXPT. RESP. A WABT
         BZ    ERROR1              NO--ERROR EXIT  YES--
         OI    PGMSW,X'01'         SET SWITCH TO RESEND THIS RECORD
         BCT   REG5,ENQSND         SEND ENQ AGAIN
*
*  CONTENTION EXIT
*
ENDOJQ   MSG   CENQ                ISSUE INDICATIVE MESSAGE
         B     ENDOJB              SEND AN EOT INDICATING END OF JOB
*
*  ERROR EXIT
*
ERROR    TM    ERRFG,X'40'         TEST FOR I/O ERROR
         BZ    ERROR1              NO--EXIT  YES--
         TM    BSCFLED+3,X'02'     TEST FOR TIMEOUT
         BO    TIMEOUT             YES--GO TO ROUTINE  ELSE
ERROR1   MSG   ERR                 ISSUE INDICATIVE MESSAGE
         B     ENDOJB              SEND AN EOT INDICATING END OF JOB
*
*  TIMEOUT
*
TIMEOUT  TM    PGMSW,X'10'         HAS TIMEOUT BEEN RETRIED
         BO    *+10                YES--EXIT  NO--
         OI    PGMSW,X'10'         TURN ON SWITCH INDICATING RETRY
         BR    REG7                GO TO RETRY
         MSG   TIME
         B     ENDOJB
*
*  WABT EXIT
*
EWABT    MSG   WABT                ISSUE INDICATIVE MESSAGE
         B     ENDOJB              GO TO END OF JOB
*
*  NORMAL END-OF-JOB EXIT
*
ENDOJ    MSG   EOJ                 ISSUE EOJ MESSAGE
ENDOJB   CNTRL BSCFLE,EOT          SEND AN EOT
         WAIT  BSCFLED
         CLOSE RDFLE,WTFLE         CLOSE THE FILES
         BCLOS BSCFLE
         WAIT  BSCFLED
ERPT     ERRPT BSCFLE
*                                  PRINT THE ERROR STATISTICS
         EOJ
*
*
*  REGISTER EQUATES
*
REG4     EQU   4                   WORK REGISTER
REG5     EQU   5                   COUNT-CONTROLLED LOOP REGISTER
REG6     EQU   6                   BASE REGISTER
REG7     EQU   7                   RETURN ADDRESS FOR TIMEOUT
*
*  CONSTANTS
*
THREE    DC    F'3'                LOOP COUNT
STCARD   DC    X'02'               STX
WTCARD   DS    CL80                DATA AREA
         DC    X'03'               ETX
LENGTH   DC    H'82'               DATA LENGTH
PGMSW    DC    X'00'               PROGRAM SWITCH--CONTROLS EXIT FROM
*                                        WABT LOOP
*
         END   BASE
// END
```

Figure 64. (Part 3 of 3)

PART 4 - BOS/BSC SUPPORT CHANNEL PROGRAMS

The BOS/BSC Support channel programs (CCW lists) set up by BSC macros to facilitate the sending and receiving of data to and from a remote CPU follow. For a description of each macro and/or type operation, refer to Binary Synchronous Communication.

These command sequences may be useful to the problem program which must interface with some other BSC support, such as DOS BTAM BSC.

BOPEN Macro

Using the DIAL keyboard operand coded on the BOPEN macro instruction, one of the following BSC channel programs is set up.

DIAL operand	channel commands
DIAL=YES	Disable Set Mode
DIAL=NO	Disable Set Mode Enable

IDIAL Macro

Using the parameters on the IDIAL macro instruction and the problem program-supplied IDLST, one of the following BSC channel programs is set up.

IDIAL operands	channel commands
DIAL-ANS or DIAL=MAN and OPTYPE=RD	Enable READ [RCVID] ENQ WRITE [SNDID] ACK-0 READ data
DIAL=CALL and OPTYPE=WT or DIAL=MAN and OPTYPE=WT	Dial or Enable WRITE [SNDID] ENQ READ [RCVID] ACK-0 WRITE data READ response
DIAL=CALL and OPTYPE=WTX or DIAL=MAN and OPTYPE=WTX	Dial or Enable WRITE [SNDID] ENQ READ [RCVID] ACK-0 WRITE data WRITE DLE ETX READ response

BCLOS Macro

The following BSC channel program is set up.

	channel commands
BCLOS Macro	Disable Set Mode

READ Macro

The appropriate channel program for READ is set up according to the type-code specified on the READ macro instruction.

Type-code	channel commands
TN (Continue)	WRITE ACK READ data
TP (Repeat)	WRITE NAK READ data
TG (Continue with leading graphics)	WRITE graphics WRITE ACK READ data
TL (Repeat with leading graphics)	WRITE graphics WRITE NAK READ data
TQ (Inquiry)	READ ENQ

WRITE Macro

The appropriate channel program for WRITE is set up according to the type-code specified on the WRITE macro instruction.

type-code	channel commands
TT (Continue)	WRITE data READ response
TC (Conversational)	WRITE data READ data (or response)
TX (Transparent Text)	WRITE data WRITE DLE ETX READ response
TXB (Transparent Block)	WRITE data WRITE DLE ETB READ response
TV (Transparent Conversational)	WRITE data WRITE DLE ETX READ data (or response)

CNTRL MACRO

For each operation type (code parameter on the CNTRL macro instruction) the appropriate channel program is set up.

operation type	channel commands
WABT (Wait before Transmitting)	WRITE WABT Prepare READ response
PRP (Prepare)	Prepare READ response
ENQ (Inquiry	WRITE ENQ READ response
DSC (Disconnect)	WRITE Disconnect (DLE EOT)
EOT (leased line)	WRITE EOT
EOT (dial line)	WRITE EOT READ response

PART 5-BOS/BSC TP OP CODES

BOS/BSC support uses bytes 40-47 of the channel command word (CCW), which are normally not used, for a "TP op code", indentifying to PIOCS the operation being performed by the command.

TP CODE	MEANING
X'01'	Prepare, Enable, Dial, or Disable command
X'02'	WRITE special characters: DLE ETB, DLE ETX, graphics, or WABT
X'03'	WRITE [ID] ENQ
X'04'	WRITE [ID] ACK-0
X'05'	READ response to CNTRL ENQ or EOT
X'07'	READ [ID] ACK-0
X'09'	WRITE response to text or ENQ
X'11'	READ or WRITE text
X'20'	READ response to text
X'21'	WRITE EOT, DLE EOT, Set Mode
X'24'	READ ENQ (READ TQ or READ response after Prepare)
X'31'	READ Skip - error recovery
X'32'	WRITE ENQ - error recovery
X'34'	READ ENQ - error recovery
X'35'	WRITE DLE ENQ - error recovery
X'39'	WRITE NAK or current SNDACK - error recovery

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

READER'S COMMENT FORM

IBM System/360 Basic Operating System
Assembler with Input/Output Macros
Specifications

Form C24-3361-6

- How did you use this publication?

  As a reference source ..... ☐
  As a classroom text ..... ☐
  As a self-study text ..... ☐

- Based on your own experience, rate this publication . . .

	.....	.....	.....	.....	.....
As a reference source:	Very Good	Good	Fair	Poor	Very Poor
As a text:	Very Good	Good	Fair	Poor	Very Poor

- What is your occupation? .....

- We would appreciate your other comments; please give specific page and line references where appropriate. If you wish a reply, be sure to include your name and address.

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

YOUR COMMENTS PLEASE . . .

This SRL bulletin is one of a series which serves as reference sources for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

Fold

Fold

FIRST CLASS
PERMIT NO. 1359
WHITE PLAINS, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation
112 East Post Road
White Plains, N. Y. 10601

Attention: Department 813

Fold

Fold

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]