File No. S360-21
Form C28-6811-1

44PS

IBM Systems Reference Library

IBM System/360 Model 44

Programming System

Assembler Language

This publication contains specifications for the IBM System/360 Model 44 Programming System Assembler Language.

This assembler language is used to write programs for the Model 44. The IBM System/360 Model 44 Programming System Assembler program processes the language and provides auxiliary functions useful in the preparation and documentation of a program.

PREFACE

This publication is a reference manual for the programmer using the assembler language and its features.

This publication presents information common to all parts of the language, followed by specific information concerning the symbolic machine instruction codes and the assembler program functions provided for the programmer's use.

Appendixes A through F present such items as a summary chart for constants, instruction listings, character set representations, and other aids to programming. Appendix G is a features comparison chart of System/360 assemblers.

Knowledge of IBM System/360 machine operations, particularly storage addressing, data formats, and machine instruction formats and functions, is prerequisite to using this publication. It is assumed that the reader has experience with programming concepts and techniques or has completed basic courses of instruction in these areas.

The publications most closely supplemental to this one are:

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

IBM System/360: System Summary, Form A22-6810

IBM System/360 Model 44: Functional Characteristics, Form A22-6875

IBM System/360 Model 44 Programming System: Concepts and Facilities, Form C28-6810

IBM System/360 Model 44 Programming System: Guide to System Use, Form C28-6812

Data Acquisition Special Features for IBM System/360 Model 44, Form A22-6900

Second Edition

This is a major revision of, and makes obsolete, C28-6811-0. Section 7, "Update Feature," has been substantially revised to include additional information and examples of update operations. Appendix G, "Features Comparison Checklists," has been rewritten to define more specifically the relationship between the IBM System/360 Model 44 Programming System Assembler Language and the other System/360 programming support system assembler languages. Because of a change in specifications, all references to the length attribute of a symbol have been deleted. Changes to the text other than these are indicated by a vertical line to the left of the change.

Specifications contained herein are subject to change from time to time. Any such change will be reported in subsequent revisions or Technical Newsletters.

Copies of this and other IBM publications can be obtained through IBM Branch Offices.

A form is provided at the back of this publication for reader's comments. If the form has been removed, comments may be addressed to IBM Corporation, Programming Publications, 1271 Avenue of the Americas, New York, N.Y., 10020.

FIGURES

TABLES

Computer programs may be expressed in machine language, i.e., language interpreted directly 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 a 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 publication is a symbolic programming language for the IBM System/360 Model 44. It enables the programmer to use IBM System/360 machine functions as though he were coding in System/360 Model 44 machine language.

The assembler program that processes the language translates symbolic instructions into machine-language instructions, assigns storage locations, and performs auxiliary functions necessary to produce an executable machine-language program.

COMPATIBILITY

The IBM System/360 Model 44 Programming System Assembler Language is a selected subset of the language available in the IBM System/360 programming support systems designed for the Models 30, 40, 50, 65, and 75 -- specifically:

- System/360 Operating System (OS/360)
- System/360 Disk Operating System (DOS/360)
- System/360 Tape Operating System (TOS/360)

Thus, source programs written in the Model 44 assembler language can be assembled by the appropriate assembler, OS/360, DOS/360, or TOS/360, provided that (1) any source statements involving subroutine linkages or supervisory functions are modified to the format specified for the applicable system, (2) there are no statements using instructions peculiar to the Model 44 or the Model 44 assembler program, and (3) all SETA variable symbols are defined using the LCLA or GBLA statements as specified in the appropriate language. Appendix G describes more specifically the relationship between the Model 44 Programming System Assembler Language and the other System/360 programming support system assembler languages.

THE ASSEMBLER LANGUAGE

The basis of the assembler language is a collection of mnemonic symbols that represent:

1. System/360 machine-language operation codes.
2. Operations (auxiliary functions) to be performed by the assembler program.

The language is augmented by other symbols, supplied by the programmer, and used to represent storage addresses or data. Symbols are easier to remember and to code than are their machine-language equivalents. Use of symbols greatly reduces programming effort and error.

Machine Operation Codes

The assembler language provides mnemonic machine-instruction operation codes for all machine instructions that can be processed by the Model 44 source programs and extended mnemonic operation codes for the conditional branch instruction. Appendix D lists the acceptable machine operation codes for Model 44 source programs.

Assembler Operation Codes

The assembler language also contains mnemonic assembler-instruction operation codes, used to specify auxiliary functions to be performed by the assembler. These are instructions to the assembler program itself and, with a few exceptions, do not result in the generation of machine-language code by the assembler program.

THE ASSEMBLER PROGRAM

The assembler program, also referred to as the "assembler," processes the source statements written in the assembler language.

Basic Functions

Processing a source program involves the translation of source statements into machine language, the assignment of storage locations to instructions and other elements of the program, and the performance of the auxiliary assembler functions designated by the programmer. The output of the assembler program is the relocatable module, a machine-language translation of the source program. The assembler furnishes a printed listing of the source statements and object program statements and additional information useful to the programmer in analyzing his program, such as error indications. The object program is in the format required by the linkage editor component of the System/360 Model 44 Programming System.

PROGRAMMER AIDS

The assembler provides auxiliary functions that assist the programmer in checking and documenting programs, in controlling address assignments, in segmenting a program, in data and symbol definition, and in controlling the assembler itself. Mnemonic operation codes for these functions are provided in the language.

Variety in Data Representation: Decimal, 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 publication *IBM System/360: Principles of Operation*, Form A22-6821, 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 there.

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 linkage editing 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 assembler allows 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 transfer of control between programs. A detailed discussion of program sectioning and linking is contained in Section 3.

Program Listings: A listing of the source program statements and the resulting object program statements may be produced by the assembler for each source program it assembles. The programmer can control the form and content of the listing to some degree.

An alphabetical listing of all the symbols used in the program, together with cross references to the statements that use each symbol, can also be produced.

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, as described in the publication *IBM System/360 Model 44 Programming System: Guide to System Use*, Form C28-6812.

PROGRAMMING SYSTEM RELATIONSHIPS

The assembler is a component of the IBM System/360 Model 44 Programming System and, as such, functions under control of the programming system. The programming system provides the assembler with input/output and other services needed in assembling a source program. In a like manner, the object program produced by the assembler will normally operate under control of the programming system and depend on it for input/output and other services. In writing the source program, the programmer uses the Supervisor Call (SVC) instruction to invoke the facilities of the programming system supervisor. The programming system supervisor is discussed in the publication *IBM System/360 Model 44 Programming System: Concepts and Facilities*, Form C28-6810.

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

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 (shown in 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 on the form for program identification and instructions to keypunch operators. None of this information is punched into a card.

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

The entries (i.e., coding), composing a statement, occupy columns 1-71 of a line. Therefore, column 1 is referred to as the "begin" column and column 71 is referred to as the "end" column. The "begin" statement boundary may be altered by use of the Input Format Control (ICTL) assembler instruction

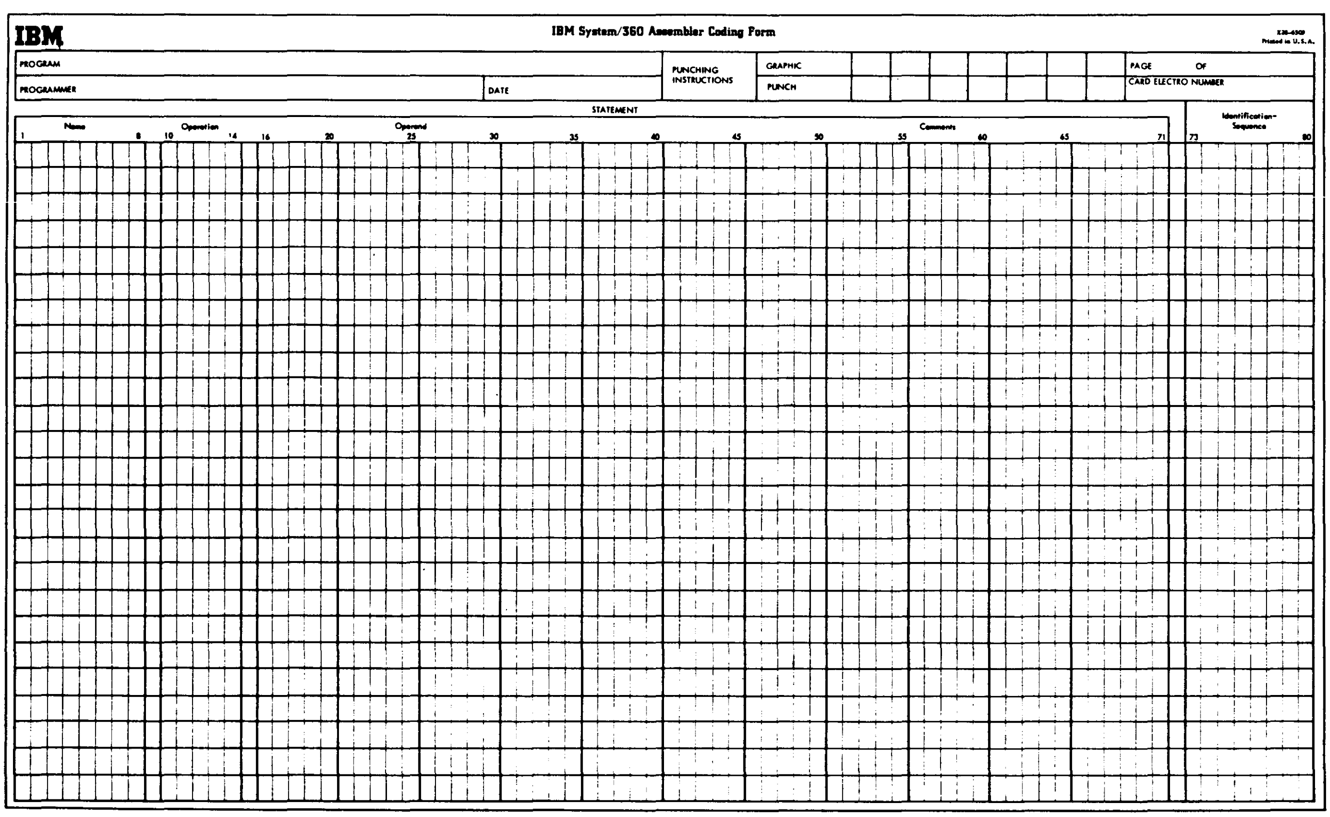

Figure 1. Coding Form

(discussed later) to designate an alternate begin column.

Statement Format

A statement can be either a comment or an instruction.

A statement 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.

Instructions 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.

Only one statement is allowed per line; a statement cannot be continued on additional lines. Column 72 must be blank. Columns 73 through 80 may contain a serial number, as discussed in Section 7, "Update Feature."

The coding form (Figure 1) is vertically ruled to provide an 8-character name field, a 5-character operation field, and a 56-character operand and/or comments field.

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

1. The entries must not extend beyond the "begin" and "end" statement boundaries (either the conventional boundaries, or the altered boundaries).
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 start in the begin column.

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 usually 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 in the name entry.

Operation Entries: The operation entry is the mnemonic operation code specifying the machine operation or assembler operation desired. An operation entry is mandatory and must start at least one position to the right of the begin column. Valid mnemonic operation codes for machine and assembler operations are contained in Appendixes D and E of this publication. Valid operation codes consist of five characters or fewer for machine or assembler-instruction operation codes. No blanks may appear within the operation entry.

Operand Entries: The operand entry is 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 particular instruction, an operand entry may consist of one or more operands. Operands are required for all machine instructions but not for all assembler instructions.

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

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 embedded blanks, e.g., C'A D'.

Comments Entries: Comments are descriptive items of information about the program that are to be inserted in the program listing. All 256 valid characters (see "Character Set" in this section), including blanks, may be used in writing a comment. The entry must be separated from the operand entry by a blank. The comments entry cannot extend beyond column 71.

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

Summary of Statement Format

The entries in an instruction must always be in the following order: name, operation, operand(s), comment.

Every instruction requires an operation entry. Comments entries are optional. Name entries are required for certain instructions and are optional in all other cases. Operand entries are required for all machine instructions and most assembler instructions.

The name and operation entries must not contain embedded blanks. Operands must not have blanks preceding or following the commas that separate them.

All entries must be contained within the designated statement boundaries.

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 in the source cards and reproduced in the program listing.

Character Set

Source statements are written using the following characters:

Letters	A through Z, and $, #, @
Digits	0 through 9
Special Characters	+ - , = . * ( ) ' / & blank

These characters are represented by the card-punch combinations and internal bit configurations listed in Appendix A. In addition, any of the 256 punch combinations may be designated anywhere that characters are used in comments and between paired single quotes.

ASSEMBLER LANGUAGE STRUCTURE

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

A source statement is composed of:

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

A name entry is:

- A symbol.

An operation entry is:

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

An operand entry is:

- One or more operands, each composed of one or more expressions. An expression is 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 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, depending on 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. A term is absolute if its value does not change upon relocation. A term is relocatable if its value changes upon relocation.

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

TERMS AND EXPRESSIONS

TERMS

A term is a character or combination of characters that represents a value. This value may be assigned by the assembler (symbols, location counter reference) or may be inherent in the term itself (self-defining term, literal).

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

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 refer to a program element. A symbol, created by the programmer for use as a name entry and/or an operand, must conform to these rules:

1. The symbol must not consist of more than eight characters.
2. The first character must be a letter. The other characters may be letters, digits, or a combination of the two.
3. A symbol may not contain special characters, including blanks.

The following are valid symbols:

READER	LOOP2	@B4
A23456	N	$A1
X4F2	S4	#56

The following symbols are invalid, for the reasons noted:

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

DEFINING SYMBOLS: The assembler assigns a value to each symbol appearing as a name entry in a source statement. The value assigned to a symbol naming a storage area, an instruction, a constant, or a control section is the address of the leftmost byte of the storage field containing the named item. Since the address of such an item may change upon program relocation, the symbol naming it is considered a relocatable term.

A symbol used as a name entry in the Equate Symbol (EQU) assembler instruction is 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 involving external references is discussed in Section 3, under "Program Sectioning and Linking.")

A symbol may be defined only once in an assembly. That is, each symbol used as the name of a statement must be unique within that assembly.

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

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. For example, the decimal self-defining term 15 represents a value of 15.

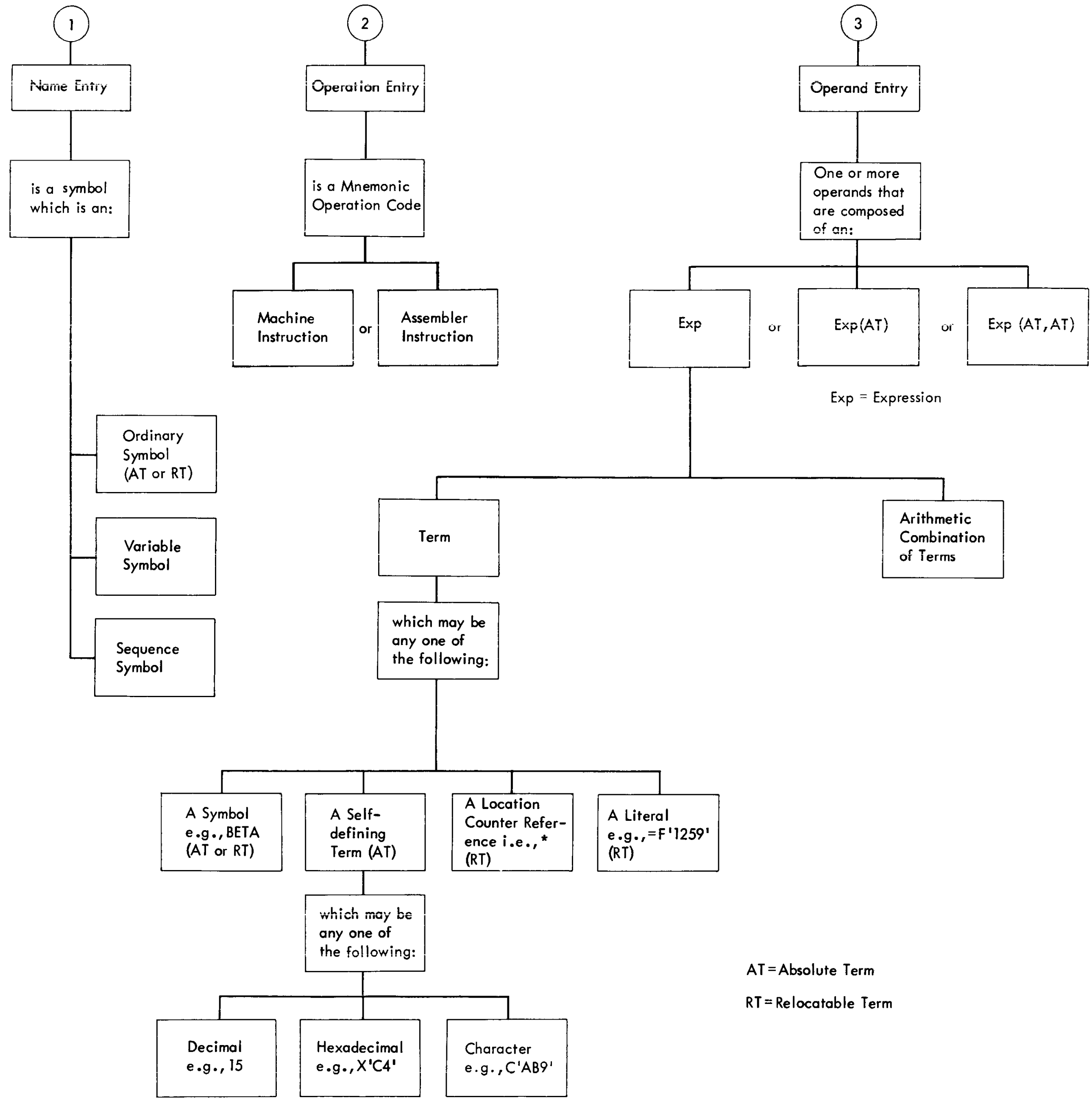

Figure 2. Assembler Language Structure -- Machine and Assembler Instructions

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

Self-defining terms are classed as absolute terms, since the values they represent do not change upon program relocation.

<u>USING SELF-DEFINING TERMS:</u> Self-defining terms are the means of specifying machine values or bit configurations without equating the values to symbols and using the symbols.

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, 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 <u>value</u> is assembled into the instruction. When a data constant is referred to or a literal is specified in the operand of an instruction, its <u>address</u> is assembled into the instruction. Self-defining terms are always right-justified; truncation or padding with zeros, if necessary, occurs on the left.

<u>Decimal Self-Defining Term:</u> A decimal self-defining term is simply an unsigned decimal number written as a sequence of decimal digits. High-order zeros may be used (e.g., 009). 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; 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; to be exact, it may not exceed 16777215 (2^{24}-1). A decimal self-defining term is assembled as its binary equivalent. Some examples of decimal self-defining terms are: 8, 147, 4092, and 00021.

<u>Hexadecimal Self-Defining Term:</u> A hexadecimal self-defining term is an unsigned hexadecimal number (written as a sequence of one to six hexadecimal digits) enclosed in single quotes and preceded by the letter X: X'C49'.

Each hexadecimal digit is assembled as its 4-bit binary equivalent. Thus, a hexadecimal term used to represent an 8-bit mask would include two hexadecimal digits. The maximum value of a hexadecimal term is 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 B.

<u>Character Self-Defining Term:</u> A character self-defining term consists of one to three characters enclosed by single quotes and preceded by the letter C. All letters, decimal digits, and special characters may be used in a character term. In addition, any of the remainder of the 256 punch combinations may be designated in a character self-defining term. Examples of character self-defining terms are as follows:

```
C'/'          C' ' (blank)
C'ABC'        C'13'
```

Because of the use of single quotes and ampersands as syntactic characters (ampersands are used as syntactic characters in variable symbols, which are discussed in Section 6), the following rule must be observed when using these characters in a character term:

For each single quote or ampersand desired in a character self-defining term, two single quotes or ampersands must be written. For example, the character values to the left are specified as indicated to the right:

```
A'#           C'A''#'
B&B           C'B&&B'
' '           C''' '''
'&'           C'''&&'''
```

Each character in the character sequence is assembled as its 8-bit code equivalent (see Appendix A). The two single quotes or ampersands that must be used to represent a single quote or ampersand within the character sequence are assembled as one single quote or ampersand.

Location Counter Reference

The location counter reference enables the programmer to refer to the current value of the location counter. The location counter is used to assign storage addresses to program statements. It is the assembler'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 after the statement has been processed, incremented by the length of the assembled item. Thus, after a statement has been processed, it 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.

For each successively declared control section, the location counter assigns locations in consecutively higher areas of storage. The first location of each control section is aligned to a double-word boundary. (Control sections are discussed further in Section 3, "Program Sectioning and Linking.")

The location counter setting can be controlled by using the START and ORG assembler instructions, which are described in Sections 3 and 5. The maximum value for the location counter is $2^{24}-1$.

The programmer may refer to the current value of the location counter at any place in a program by using an asterisk as a term 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 as the operand 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.

A reference to the location counter may be made in an address constant literal (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 that requires the use of a predefined symbol, with the exception of the EQU and ORG assembler instructions.

Literals

A literal may be used to introduce data into a program. It is simply a DC operand preceded by an equal sign (=).

A literal represents data rather than a reference to data. The appearance of a literal in a statement causes the assembler program to assemble the data specified by the literal, store this data in a "literal pool," and place the address of the storage field containing the data in the operand field of the assembled 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. Specifying a literal is in contrast to using the DC assembler instruction to enter the data into the program and then specifying the name of the DC instruction in the operand. Only one literal is allowed in a machine-instruction statement.

A literal may not be combined with any other terms.

A literal may not be used as the receiving field of an instruction that modifies storage.

A literal may not be specified in an address constant (see Section 5, "DC--Define Constant").

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 value F'274' is stored.

A literal may be used as an operand wherever a storage address is specified in a machine instruction or in a CCW assembler instruction operand. 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 data itself. The descriptive portion of the literal must indicate the format of the constant.

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 (Section 5) for the means of specifying a literal. The type of literal designated in an instruction is not checked for correspondence with the operation code of the instruction.

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
```

<u>The Literal Pool:</u> 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 must be assigned by the programmer within the control section in which the literal is used.

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 in Section 5 under "LTORG--Begin Literal Pool."

EXPRESSIONS

This subsection discusses the expressions used in coding operand entries for source statements. Two types of expressions, absolute and relocatable, are presented along with the rules for determining these attributes of an expression.

As shown in Figure 2, an expression is composed of a single term or an arithmetic combination of terms. The arithmetic operators that may be used to combine the terms of an expression are + (addition), - (subtraction), * (multiplication), and / (division).

The following are examples of valid expressions (provided that BETA, LAMBDA, GAMMA, TEN, and TWO are absolute):

```
AREA1+X'2D'    BETA*10
*+32           C'ABC'
N-25           29
FIELD          LAMBDA+GAMMA
=F'1234'       TEN/TWO
```

The rules for coding expressions are:

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 three terms.
4. An expression may not have more than one level of parentheses (i.e., a parenthetical expression may not appear within a parenthetical expression).
5. A multiterm expression may not contain a literal.
6. A parenthesized expression may not contain a literal.

Evaluation of Expressions

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

A multiterm expression, e.g., BETA+10, ENTRY-EXIT, 25*10+A, is reduced to a single value, as follows:

1. Each term is given its value.
2. Every expression is computed to 32 bits.
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 always yields an integer result; any fractional portion of the result is dropped. For example, 1/2*10 yields a zero result, whereas 10*1/2 yields 5.
5. Division by zero is valid and yields a zero result.

A parenthesized multiterm expression used in an expression is processed before the rest of the terms in the expression, e.g., in the expression BETA*(CON-10), the term CON-10 is evaluated first and the resulting value is used in computing the final value of the expression.

Negative values are carried in two's complement form. Final values of expressions are the truncated rightmost 24 bits of the results. The value of an expression before truncation must be in the range -2^{24} through $2^{24}-1$. A negative result is considered to be a 3-byte positive value. Intermediate results have a range of -2^{31} through $2^{31}-1$.

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(s) composing them.

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, or any of the self-defining terms. All arithmetic operations are permitted between absolute terms.

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

1. There must be an even number of relocatable terms in the expression.

2. The relocatable terms must be paired. Each pair of terms must have the same relocatability attribute, i.e., they appear in the same control section in this assembly (see Section 3, "Program Sectioning and Linking"). Each pair must consist of terms with opposite signs. The paired terms do not have to be contiguous, e.g., RT+AT-RT.

3. 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 (with opposite signs and the same relocatability attribute) 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 with the same relocatability attribute. 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 value of the expression would still be 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 with the same relocatability attribute.

```
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, i.e., with the same relocata-
    bility attribute)
```

RELOCATABLE EXPRESSION: 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. All relocatable expressions must have a positive value.

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. All relocatable terms but one must be paired. Pairing is described in the preceding discussion of absolute expressions.

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. The relocatability attribute is that of the odd relocatable term.

For example, in the expression W-X+W, the terms W and X are relocatable terms with the same relocatability attribute. If, initially, W equals 10 and X equals 15, the value of the expression is 5. However, upon relocation, this value will change. If a relocation factor of 100 is applied, the value of the expression is 105. Note that the value of the paired terms, W-X, remains constant at -5 regardless of relocation. Thus, the new value of the expression, 105, 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 with the same relocatability attribute, Y is a relocatable term with a different relocatability attribute.

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

ADDRESSING

The IBM 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 composed 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 in the machine.)

The programmer writes an explicit address by specifying the displacement and the base register number. In designating explicit addresses, a base register may not be combined with a relocatable symbol.

The programmer writes an implied address by specifying an absolute or relocatable address. The assembler has the facility to select a base register and compute a displacement, thereby generating an explicit address from an implied address, provided that it has been informed as to (1) what base registers are available to it and (2) what each contains. The programmer conveys this information to the assembler through the USING and DROP assembler 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 USING statements, which 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 that contains implicit addressing.

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.

USING -- Use Base Address Register

The USING instruction specifies a general register that is available for use as a base register. This instruction also states the base address value that the assembler may assume will be in the register at object time. Note that a USING instruction does not load the register specified. It is the programmer's responsibility to make sure that the specified base address value is placed into the register. 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	Two expressions of the form v,r

Operand v must be an absolute or relocatable expression. Literals are not permitted. Operand v specifies a value that the assembler can use as a base address. The operand r must be an absolute term. It specifies the general register that the assembler assumes will contain the base address represented by operand v. The value of r must be in the range from 0 to 15.

For example, the following USING statement tells the assembler it may assume that the current value of the location counter will be in general register 12 at execution time.

Name	Operation	Operand
	USING	*,12

If the programmer changes the value in a base register currently being used, and wishes the assembler to compute displacement 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 from any control section in the program or has an absolute value of zero. If general register 0 is specified, the assembler assumes that register 0 contains the value zero.

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 with an alternate register in the USING statement.
2. Inserting an instruction that loads the alternate 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	One absolute term

The absolute term indicates a general register previously named in a USING statement that is now unavailable for base addressing. The following statement, for example, prevents the assembler from using register 7:

Name	Operation	Operand
	DROP	7

It is not necessary to use a DROP statement when the base address being used 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 that 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 specified in a USING instruction to have a relocatable value. 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 calculating the base register to be used, the assembler will always use the available register giving the smallest displacement. If there are two registers with the same value, the highest numbered register will be used.

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

In the preceding sequence, the BALR instruction loads register 2 with the address of the immediately following storage location. In this case, it is the address of 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.

In the following sequence, the BALR and L instructions load registers 2 through 5. The USING instructions indicate 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 changing the number of registers designated by the USING and L instructions and the number of address constants specified in the DC instruction.

Name	Operation	Operand
BEGIN	BALR	2,0
	USING	HERE,2
	USING	HERE+4096,3
	USING	HERE+8192,4
	USING	HERE+12288,5
HERE	L	3,BASEAD
	L	4,BASEAD+4
	L	5,BASEAD+8
	B	FIRST
BASEAD	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 program in sections. The sections may be assembled separately, then combined via the linkage editor into one or more executable phases. 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 plus the number of unique symbols in EXTRN statements may not exceed 255.

Sectioning a program is optional, and many programs can best be written without sectioning. 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 should be taken into 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 (i.e., without affecting the location 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 assembler output consists of the assembled control sections and a control dictionary. The control dictionary contains information the linkage editor needs to complete cross-referencing between control sections as they are combined into an object program. The linkage editor can combine control sections from various assemblies 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 multisectioned program, or part of a multisectioned 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

Locations are assigned to control sections as if 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.

START -- Start Assembly

The START instruction may be used to give a name to the first (or only) control section of a program. It may also be used to specify an initial location counter value 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. Otherwise, the control section is considered to be unnamed. All subsequent statements are assembled as part of that control section. This continues until an instruction identifying a different control section (CSECT, DSECT, or COM) is encountered.

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

The assembler uses the self-defining term specified by the operand as the initial location counter value of the program. This value should 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 initial location counter value of the program at zero. The location counter is set at the next double-word boundary when the value of the START operand is not divisible by eight.

Note: The START instruction may not be preceded by any type of assembler language statement that may either affect or depend upon the setting of the location counter.

CSECT -- Identify Control Section

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

Name	Operation	Operand
A symbol or blank	CSECT	Must be blank

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. Multiple CSECT instructions must have unique names. The name of a CSECT may be blank, provided that no other CSECT or START instruction has a blank name.

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

The occurrence of a CSECT instruction terminates the previous control section.

Unnamed Control Section

If it is desired to write a program that is unsectioned, the program does not need to contain a CSECT or START instruction. In this case, the assembler will generate an unnamed START statement for the first assembler language statement that may either affect or depend upon the setting of the location counter.

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 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	Must be blank

The symbol in the name field is a valid relocatable symbol whose value represents the first byte of the section.

All statements following the DSECT instruction are assembled as part of that control section until a statement identifying a different control section is encountered (i.e., another DSECT, CSECT, or COM instruction). All assembler language instructions may occur within dummy sections.

Symbols that name 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."

The occurrence of a DSECT instruction terminates the previous control section. A DSECT cannot be resumed.

Note: 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.

Name	Operation	Operand
ASMBL2	CSECT	
BEGIN	BALR	2,0
	USING	*,2
	.	
	.	
	.	
	USING	INAREA,3
	CLI	INCODE,C'A'
	BE	ATYPE
	.	
	.	
	.	
ATYPE	LA	5,0
	LA	6,5
	L	7,INPUTA(5)
	ST	7,WORKA(5)
	LA	5,4(5)
	BCT	6,*-12
	LA	5,0
	LA	6,9
	LH	7,INPUTB(5)
	STH	7,WORKB(5)
	LA	5,2(5)
	BCT	6,*-12
	.	
	.	
	.	
WORKA	DS	5F
WORKB	DS	9H
	.	
	.	
	.	
INAREA	DSECT	
INCODE	DS	CL1
INPUTA	DS	5F
INPUTB	DS	9H
	.	
	.	
	.	
	END	

An example of addressing dummy sections is shown in the foregoing 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. The coding shown in the example is from Assembly 2.

The input area is described in Assembly 2 by the DSECT control section 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 it is assumed that general register 3 contains 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.

COM -- DEFINE COMMON CONTROL SECTION

The COM assembler instruction identifies and reserves a common area of storage that may be referred to by independent assemblies that have been linked and loaded for execution as one overall program. The format is:

Name	Operation	Operand
Symbol or blank	COM	Must be blank

Only one common control section can be designated in an assembly.

When several assemblies, each designating a common control section of the same name, are linkage edited, the amount of storage reserved for this name is equal to the longest of these common control sections. (In this context, a blank common control section is considered to be uniquely named.)

The common area may be broken up into subfields through use of the DS and DC assembler instructions. Names of subfields are defined relative to the beginning of the common section, as in the DSECT control section.

Instructions or constants that appear in a common control section are not assembled, i.e., no machine language code is generated for them. As much storage is reserved as would be required for the instructions or constants if they were assembled. Data can be placed in a common control section only through execution of the program.

If the assignment of common storage is done in the same manner by each independent assembly, reference to a location in common by any assembly results in the same location being referenced. When assembled, common location assignment starts at zero.

The occurrence of a COM instruction terminates the previous control section.

Name	Operation	Operand
	COM	
AREA1	DC	5F'0'
AREA2	DS	9H
MASK	LA	6,AREA2

The above statements reserve a common area of storage that is 42 bytes long. The common area contains three subfields: AREA1 occupies five fullwords (20 bytes), AREA2 occupies nine halfwords (18 bytes), and MASK occupies 4 bytes. No machine language code is generated.

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 during a subsequent phase of processing. 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 may 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.

ENTRY -- IDENTIFY ENTRY-POINT SYMBOL

The ENTRY instruction identifies linkage symbols that are 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	One symbol that also appears as a statement name

A program may contain a maximum of 100 ENTRY symbols. ENTRY symbols that are not defined (i.e., that do not appear as statement names), although invalid, will also count toward this maximum of 100 ENTRY symbols.

The symbol in the ENTRY operand field may be used as an operand by another program. An ENTRY statement operand may not contain a symbol defined in a dummy section or common. The following example identifies the statement named SINE as an entry point to the program.

Name	Operation	Operand
	ENTRY	SINE

Note: 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.

EXTRN -- IDENTIFY EXTERNAL SYMBOL

The EXTRN instruction identifies linkage symbols that are 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	One symbol

The symbol in the operand field may not appear as a name of a statement in this program. It may not be used in expressions requiring that all symbols be previously defined. Thus, an EXTRN symbol may not be used in the operand of an EQU assembler instruction.

The following example identifies two external symbols that have been used as operands in this program but are defined in some other program.

Name	Operation	Operand
	EXTRN	RATEBL
	EXTRN	WITHCA

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

Note: When external symbols are used in an expression, they may not be paired. Each external symbol must be considered as having a unique relocatability attribute.

Addressing External Control Sections

One way in which a program is linked to an external control section is to have the program:

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

2. Load the address constant into a general register and branch to the control section via the register.

Name	Operation	Operand
	EXTRN	SINE
MAINPR	CSECT	
BEGIN	BALR	2,0
	USING	*,2
	.	
	.	
	.	
	L	3,ACON
	BALR	1,3
	.	
	.	
	.	
ACON	DC	A(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 RATETB, which is in another control section, the following coding might be used:

Name	Operation	Operand
	EXTRN	RATETB
MAINPR	CSECT	
BEGIN	BALR	2,0
	USING	*,2
	.	
	.	
	.	
	L	4,RATEAD
	USING	RATETB,4
	A	3,RATETB
	.	
	.	
	.	
RATEAD	DC	A(RATETB)
	END	BEGIN

The combined number of control sections and dummy sections plus the number of unique symbols in EXTRN statements may not exceed 255.

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 publication IBM System/360: Principles of Operation, Form A22-6821.

This section should be used in conjunction with Appendix C, which describes assembler operand field formats for the various machine instructions.

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. Four formats are acceptable to the assembler: RR, RX, RS, and SI. Within each basic format, further variations are possible.

The symbolic format of a machine instruction is similiar to, but does not duplicate, its actual format. Appendix C illustrates machine format for the four 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 Section 1.

Any machine instruction statement may be named by a symbol, which other assembler statements can use as an operand. The value of the symbol is the address of the leftmost byte assigned to the assembled instruction.

Instruction Alignment and Checking

All machine instructions are aligned automatically by the assembler on halfword 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 ensure 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, fullword multiply, and 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, an operand that specifies both an index register and a base register is written as follows: 40(3,5).

Appendix C shows two types of addressing formats for RX, RS, and SI 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, an Add instruction (RX format) may have either of the following symbolic operands:

```
R1,D2(X2,B2)    --    explicit address
R1,S2(X2)       --    implied address
```

Whereas D2 must be represented by an absolute expression, S2 may be represented by either a relocatable or an absolute expression. Both X2 and B2 must be absolute terms.

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 that has been assigned a value of 7400. Assume also that the assembler has been notified (by a USING instruction) that general register 12 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 hexadecimal:

Assembler statement:

```
     ST      4,FIELD
```

Assembled instruction:

```
     Op.Code  R1  X2  B2   D2
     50       4   0   C   CE8
```

An address may be specified explicitly as a base register and displacement (and index register for RX instructions) by the formats shown in the second column of Table 1. The address may be specified as an implied address by the formats shown in the third column.

Table 1. Details of Address Specification

Type	Explicit Address	Implied Address
RX	D2(X2,B2)	S2(X2)
	D2(,B2)	S2
RS	D2(B2)	S2
SI	D1(B1)	S1

A comma must separate operands. Parentheses must enclose a subfield or subfields, and a comma must separate two subfields within parentheses. When parentheses are used to enclose one subfield, and the subfield is omitted, the parentheses must be omitted. In the case of two subfields that are separated by a comma and enclosed by parentheses, 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       (no indexing,
                        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.

   ```
   L      2,48(4,5)
   L      2,48(,5)      (no indexing)
   ```

3. 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.

   ```
   L      2,48(4,5)
   L      2,FIELD(4)   (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 C 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.

MACHINE-INSTRUCTION MNEMONIC CODES

The mnemonic operation codes (shown in Appendix D) 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 ST represents Store. 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 Section 5). Furthermore, letters U and W have been added to indicate, respectively, short and long, unnormalized floating-point operations, and letters D and E have been added to indicate, respectively, long and short, normalized floating-point operations. For example, AE indicates Add Normalized Short, whereas AW indicates Add Unnormalized Long. Where applicable, fullword 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.

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 Section 3, "Program Sectioning and Linking" and "Base Register Instructions."

RR Format

Name	Operation	Operand
ALPHA1	LR	1,2
ALPHA2	LR	REG1,REG2
BETA	SPM	15
GAMMA1	SVC	250
GAMMA2	SVC	TEN

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

RS Format

Name	Operation	Operand
ALPHA1	SLL	REG2,15
ALPHA2	SLL	REG2,0(15)

ALPHA1 is a shift instruction shifting the contents of REG2 left 15 bit positions. ALPHA2 is a shift instruction shifting the contents of REG2 left by the value contained in general register 15.

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'
LAMBDA	L	3,20(,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. LAMBDA specifies no indexing.

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 through GAMMA3 instructions specify explicit addresses, whereas the BETA instructions and GAMMA4 instruction specify implied addresses. GAMMA2 specifies a displacement of zero. GAMMA3 specifies no base register.

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

Extended Code		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
	<u>Used After Compare Instructions</u>		
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)
	<u>Used After Arithmetic Instructions</u>		
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)
BNP	D2(X2,B2)	Branch on Not Plus	BC 13,D2(X2,B2)
BNM	D2(X2,B2)	Branch on Not Minus	BC 11,D2(X2,B2)
BNZ	D2(X2,B2)	Branch on Not Zero	BC 7,D2(X2,B2)
	<u>Used After Test Under Mask Instructions</u>		
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)
BNO	D2(X2,B2)	Branch if Not Ones	BC 14,D2(X2,B2)

Figure 3. Extended Mnemonic Codes

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.

In the following examples, which illustrate the use of extended mnemonics, it is to be assumed that the symbol GO is defined elsewhere in the program.

Name	Operation	Operand
	B	40(3,6)
	B	40(,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 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
COM -- Identify Common Control Section

* 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
ORG -- Set Location Counter
LTORG -- Begin Literal Pool
CNOP -- Conditional No Operation
END -- End Assembly
REPRO -- Reproduce Following Card

----
* Discussed in Section 3.

SYMBOL DEFINITION INSTRUCTION

EQU -- EQUATE SYMBOL

The EQU instruction is used to define a symbol by assigning to it the value and the relocatability attribute 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 may be absolute or relocatable. Any symbols appearing in the expression must be previously defined.

The symbol in the name field is given the value and the relocatability attribute of the expression in the operand field.

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. For example:

Name	Operation	Operand
AREA	EQU	ALPHA-BETA+GAMMA

The name, AREA, 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, or 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.

DC -- DEFINE CONSTANT

The DC instruction is used to provide constant data in storage. A variety of constants may be specified: fixed-point, floating-point, 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 format described below

The operand consists of four subfields: the first three describe the constant; the fourth provides the constant or constants. The first and third subfields may be omitted, but the second and fourth must be specified. Note that more than one constant may be specified in the fourth subfield for most types of constants. Each constant so specified must be of the same type; the descriptive subfields that precede the constants apply to all of them.

No blanks may occur within any of the subfields (unless provided as characters in a character constant or a character self-defining term), nor may they occur between the subfields of an operand.

The subfields of the DC operand are written in the following sequence:

Subfield			
1	2	3	4
Duplication Factor	Type	Length	Constant(s)

The symbol that names the DC instruction is the name of the constant (or first constant if the instruction specifies more than one). Relative addressing (e.g., SYMBOL+2) may be used to address the various constants if more than one has been specified, because the number of bytes allocated to each constant can be determined.

The value of the symbol naming the DC instruction is the address of the leftmost byte (after alignment) of the first, or only, constant.

Boundary alignment varies according to the type of constant being specified and the presence of a length specification. Some constant types are aligned only 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 (halfword, fullword, or doubleword) in the absence of a length specification. If length is specified, no boundary alignment occurs for such constants.

Bytes that must be skipped in order to align the field at the proper boundary are not considered to be part of the constant. Thus, the location counter is incremented to reflect the proper boundary (if any incrementing is necessary) before the address value is established. Therefore, the symbol naming the constant will not receive a value that is the location of a skipped byte.

Bytes skipped to align a DC statement are set to zero; bytes skipped to align a DS statement are not set to zero.

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

LITERAL DEFINITIONS: Note that the description of literals in Section 2 referred to the following discussion of the DC operand in reference to the writing of a literal operand. All subsequent operand specifications are applicable to writing literals; the only differences are listed below.

1. The literal is preceded by an equal sign.

2. Multiple constants may not be specified.

3. Unsigned decimal self-defining terms must be used to express the duplication factor and length values.

4. The duplication factor may not be zero.

5. If a reference to the location counter occurs in an address constant that specifies a duplication factor greater than one, the value of the location counter used in each duplication is incremented by the length of the constant; if, however, the reference occurs in a literal address constant, the value remains unchanged throughout duplication.

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

Operand Subfield 1: Duplication Factor

The duplication factor may be omitted. If specified, it causes the constant(s) to be generated the number of times indicated by the factor. The factor may be specified either by an unsigned decimal self-defining term or by a positive absolute expression that is enclosed by parentheses. All symbols in the expression must be previously defined. A location counter reference may not appear in such an expression. The maximum value permitted for the duplication factor is 65,535.

The duplication factor is applied after the constant is assembled. When more than one constant is specified in a DC operand having a duplication factor, the duplication factor is applied to the constants as a unit, rather than individually. Thus, if a duplication factor of 2 is specified for the constants 1, 2, and 3, the constants are generated in the order -- 1 2 3 1 2 3 -- not in the order -- 1 1 2 2 3 3.

Note that a duplication factor of zero is permitted and achieves the same result as it would in a DS instruction. A DC instruction with a zero duplication factor will not produce control dictionary entries. See "Forcing Alignment" under "DS -- Define Storage."

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 operand. (If the reference occurs in a literal address constant, however, the value remains unchanged.)

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 single-letter code as shown in Figure 4.

Further information about these constants is provided in "Operand Subfield 4: Constant" below.

Operand Subfield 3: Length

The length subfield may be omitted. If used, it indicates the length of the specified constant. This is written as Ln, where n is either an unsigned decimal self-defining term or a positive absolute expression enclosed by parentheses. Any symbols in the expression must be previously defined. A location counter reference may not appear in such an expression.

The value of n represents the number of bytes of storage that are assembled for the constant. The maximum values permitted for the length of the various types of constants are summarized in Appendix F. This table also indicates the implied length for each type of constant; the implied length is used unless a length subfield is present. A length may be specified for any type of constant. However, no boundary alignment will be provided when a length is given.

Code	Type of Constant	Machine Format
C	Character	8-bit code for each character
X	Hexadecimal	4-bit code for each hexadecimal digit
F	Fixed-point	Signed, fixed-point binary format; normally a fullword
H	Fixed-point	Signed, fixed-point binary format; normally a halfword
E	Floating-point	Short floating-point format; normally a fullword
D	Floating-point	Long floating-point format; normally a double word
A	Address	Value of address; normally a fullword

Figure 4. Type Codes for Constants

Operand Subfield 4: Constant

This subfield supplies the constant(s) described by the subfields that precede it. A data constant (all types except A) is enclosed by single quotes. An address constant (type A) is enclosed by parentheses. To specify two or more constants in the subfield, the constants must be separated by commas and the entire sequence of constants must be enclosed by the appropriate delimiters (i.e., single quotes or parentheses). Thus, the format for specifying the constant(s) is one of the following:

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

Fixed-point (F and H), floating-point (E and D), and address (A) constants are aligned on the proper boundary, as shown in Appendix F, unless a length modifier is specifed. 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 fullword constants, the first would be aligned on a fullword boundary, and the rest would automatically fall on fullword boundaries.

----

* Not permitted for character and hexadecimal constants.

The total storage requirement of an operand is the product of the length multiplied by the number of constants in the operand, which in turn is multiplied by 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. Thus, if several address constants in the same instruction refer to the location counter, the value of the location counter varies from constant to constant. Similarly, if a single 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 following 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.

Special consideration must be given to representing single quotes and ampersands as characters. Each single quote or ampersand desired as a character in the constant must be represented by a pair of single quotes or ampersands. Only one single quote or ampersand appears in storage.

The maximum length of a character constant is 256 bytes. No boundary alignment is performed. Each character is translated into one byte. Paired single quotes or paired ampersands count as one character. 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 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 are necessary are dropped.

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

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

Name	Operation	Operand
FIELD	DC	C'TOTAL TS 110'

However, in this next example, the length 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 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

However, if the length had been specified as six instead of four, the generated constant would have been as shown below (with the spaces between and following the grouped constants being the sixth character):

ABCDE ABCDE ABCDE

Hexadecimal Constant -- X: A hexadecimal constant consists of one or more of the hexadecimal digits, which are 0 through 9 and A through F. Only one hexadecimal constant may be specified per operand. The maximum length of a hexadecimal constant is 32 bytes (64 hexadecimal digits). No 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, the leftmost byte has the leftmost four bits filled with a hexadecimal zero, while the other four bits contain the odd (first) digit.

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 exceeds the specified length, as many hexadecimal digits as necessary are dropped from the left.

2. If the number of hexadecimal digit pairs is less than the specified length, as many hexadecimal zeros as are necessary are added on the left.

An 8-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 ones (the DS instruction sets the location counter to a fullword boundary):

Name	Operation	Operand
	DS	0F
TEST	DC	X'FF00FF00'

The next example uses a hexadecimal constant as a literal and inserts ones 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
ALPHA	DC	3XL2'A6F4E'

The resulting constant would be 6F4E, which would occupy the specified 2 bytes. It would then be duplicated 3 times, as requested by the duplication factor. If it had merely been specified as X'A6F4E', the resulting constant would have had a hexadecimal zero in the leftmost position, as follows:

0A6F4E

Fixed-Point Constants -- F and H: A fixed-point constant is written as a signed or unsigned decimal self-defining term. It is assumed that the sign is positive if an unsigned term is specified.

The decimal value is converted to a binary number. If the value of the number exceeds the length specified or implied, the sign is lost, the necessary leftmost bits are truncated to the length of the field, and the value is then assembled into the whole field. Any duplication factor that is present is applied after the constant is assembled. A negative number is carried in two's complement form.

An implied length of 4 bytes is assumed for a fullword (F) and 2 bytes for a halfword (H), and the constant is aligned to the proper fullword or halfword if a length is not specified. However, any length up to and including 8 bytes may be specified for either type of constant by a length modifier, in which case no boundary alignment occurs.

Maximum and minimum values for fixed-point constants are:

Length	Maximum	Minimum
8	$2^{63}-1$	$-(2^{63}-1)$
4	$2^{31}-1$	-2^{31}
2	$2^{15}-1$	-2^{15}
1	$2^{7}-1$	-2^{7}

A field of 3 fullwords is generated from the statement shown below. The value of CONWRD is the address of the leftmost byte of the first word, and the length of the constant is 4, the implied length for a fullword, 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'

A constant could be specified as a literal:

Name	Operation	Operand
	AH	7,=H'350'

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, it is assumed that the number is an integer. It is also assumed that the sign is positive 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 a signed or unsigned decimal self-defining term specifying the exponent of the factor 10. The exponent may be in the range from -78 to +75. If an unsigned exponent is specified, it is assumed that the sign is a plus.

The floating-point constant is converted to a normalized hexadecimal floating-point constant in machine format. Truncation of the fraction is performed according to the specified or implied length, and the number is stored in the proper field. The resulting number will not differ from the exact value by more than one in the rightmost place.

The implied length for a fullword (type E) constant is 4 bytes; the implied length for a double word (type D) constant is 8 bytes. The constant is aligned at the proper word or double word boundary if a length is not specified. However, any length up to and including 8 bytes may be specified for either type of constant by a length modifier, in which case no boundary alignment occurs.

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.

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

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

Name	Operation	Operand
FLOAT	DC	D'+46,-3.729,+473'

<u>Address Constant -- A:</u> An address constant is specified as an absolute or relocatable expression. (Note that an expression may be single term or multiterm.) The value of the expression is calculated as explained in Section 2 with one exception: the maximum value of an absolute expression may be $2^{31}-1$. The value is then truncated on the left, if necessary, to the specified or implied length of the field and assembled into the rightmost bits of the field. The implied length of an address constant is 4 bytes, and alignment is to a fullword boundary unless a length is specified, 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 to 4 bytes may be used for an absolute expression, while a length of only 3 or 4 may be used for a relocatable expression.

Address constants are 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 the registers. Coding examples that illustrate these considerations are provided in Section 3 under the heading "Programming with the USING Instruction."

In the following examples, the field generated from the statement named CONST contains 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., an address constant literal).

Name	Operation	Operand
CONST	DC	A(*+4096)
	L	4,=A(*+4096)

When the location counter reference occurs in a literal, the value of the location counter is the address of the first byte of the instruction in which the literal is used.

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.

Name	Operation	Operand
A symbol or blank	DS	One operand in the format described below

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. Although the formats are identical, there are two differences in the specification of subfields, as follows:

1. The specification of data (subfield 4) is optional in a DS operand, but it is mandatory in a DC operand. If the constant is specified, it must be valid.

2. The maximum length that may be specified for character (C) and hexadecimal (X) field types is 65,535 bytes rather than 256 bytes.

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 the programmer need do is show it as subfield 4 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 is the location of the leftmost byte of the reserved area. The length of the field 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. Bytes skipped for alignment are not set to zero.

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 4 bytes. The leftmost byte is aligned to a fullword boundary. Thus, either code could be specified if it were desired to reserve 4 bytes of storage aligned to a fullword boundary. To obtain a length of 8 bytes, one could specify either the E or F field type with a length modifier of 8. However, a duplication factor would have to be used to reserve a larger area, because the maximum length specification for either type is 8 bytes. Note also that specifying length would cancel any special boundary alignment.

In contrast, character (C) and hexadecimal (X) fields have an implied length of 1 byte. Either of these codes, if used, would have to be accompanied by a length modifier, unless just 1 byte is to be reserved. Although no alignment occurs, the use of C and X field types permits greater latitude in length specifications, the maximum for either type being 65,535 bytes. (Note that this differs from the maximum for these types in a DC instruction.) Unless a field of 1 byte is desired, either the length must be specified for the C or X field types, or else the data must be specified (as subfield 4), 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

Additional examples of DS statements are shown below:

Name	Operation	Operand
ONE	DS	CL80 (80 bytes)
TWO	DS	80C (80 bytes)
THREE	DS	6F (six full words)
FOUR	DS	D (one double word)
FIVE	DS	4H (four half words)

Note: A DS statement causes the storage area to be reserved but not set to zeros. Assumptions should not be made as to the contents of the reserved area.

Special Uses of the Duplication Factor

FORCING ALIGNMENT: The location counter can be forced to a double word, fullword, or halfword 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
TABLE	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:

Position	Meaning
5-10	Payroll Number
11-30	Employee Name
31-36	Date
47-54	Gross Wages
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.

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

Note that the first statement names the entire area by defining the symbol RDAREA; 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.

CCW -- DEFINE CHANNEL COMMAND WORD

The CCW instruction provides a convenient way to define and generate an 8 byte channel command word aligned at a doubleword boundary. The internal machine format of a channel command word is shown in Table 2. 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 term that specifies the command code. The value of this term 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 through 4.

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

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

The following is an example of a CCW statement:

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

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

<u>32-35</u>	<u>36-39</u>
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.

Table 2. Channel Command Word

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 the program listing and assembly output cards, to provide blank lines in the program listing, and to designate how much detail is to be included in the program listing. In no case are instructions or constants generated in the object program. Listing control statements with the exception of PRINT are not printed in the listing.

TITLE -- IDENTIFY ASSEMBLY OUTPUT

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

Name	Operation	Operand
An ID or blank	TITLE	A sequence of characters, enclosed in single quotes

The name field may contain an ID field of from one to four alphabetic or numeric characters in any combination. The contents of the ID field are punched into columns 73 through 76 of all the output cards for the program except those produced by the REPRO assembler instruction. Only the first TITLE statement in a program may make use of the ID field. The ID field of all subsequent TITLE statements must be blank.

The operand field may contain up to 62 characters enclosed in single quotes. Special consideration must be given to representing single quotes and ampersands as characters. Each single quote or ampersand desired as a character in the constant must be represented by a pair of single quotes or ampersands. Only one single quote or ampersand appears in storage. The contents of the operand field are printed at the top of each page of the program listing.

A program may contain more than one TITLE statement. Each TITLE statement provides the heading for subsequent pages in the program listing, until another TITLE statement is encountered. Each TITLE 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:

Name	Operand	Operand
PGM1	TITLE	'FIRST HEADING'

then PGM1 is punched into all of the output cards (columns 73 through 76) and FIRST HEADING appears at the top of each subsequent page.

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 A NEW HEADING appears at the top of each subsequent page.

<u>Note:</u> A title card with a non-blank name field must be used if the output deck will at any time be processed by the update feature of the assembler. In conjunction with the sequence numbers punched automatically in columns 77 through 80, a 4-character ID provides 8-digit serialization in columns 73 through 80, as required for an update operation.

EJECT -- START NEW PAGE

The EJECT instruction causes the next line of the listing to appear at the top of a new page. This instruction provides a convenient way to separate routines in the program listing. The format of the EJECT instruction statement is as follows:

Name	Operation	Operand
Blank	EJECT	Must be blank

If the line before the EJECT statement would have been the last line on a page anyway, the EJECT statement has no effect. Two EJECT statements may be used in succession to obtain a blank page. A TITLE instruction followed immediately by an EJECT instruction will produce a page with nothing but the operand entry of the TITLE instruction. Text following the EJECT instruction will begin at the top of the next page.

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 program listing. A blank operand causes one blank line to be inserted. If the operand value exceeds the number of lines remaining on the listing page, the statement will have the same effect as an EJECT statement.

PRINT -- PRINT OPTIONAL DATA

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

Name	Operation	Operand
Blank	PRINT	One or two operands

One or both of the following operands are used:

1. ON - A listing is printed.

   OFF - No listing is printed.

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

   NODATA - Only the leftmost 8 bytes are printed in the listing.

A program may contain any number of PRINT statements. The PRINT statement controls the printing of the program 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

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 program listing. However, if the following statement is the last PRINT statement to appear before the DC statement, only 8 bytes of zeros are printed in the program listing.

Name	Operation	Operand
	PRINT	NODATA

Whenever an operand is omitted, it is assumed to be unchanged and continues according to its last specification.

If the OFF operand is used, no data will be printed even though a DATA operand is specified. Thus, with the following statement nothing would be printed.

Name	Operation	Operand
	PRINT	OFF,DATA

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 indicate statement format, and to punch a card. Except for the CNOP instruction, none of these assembler instructions generate instructions or constants in the object program.

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	The decimal value 1 or 25

The operand specifies the begin column of the source statement.

If no ICTL statement is used in the source program, the assembler assumes that column 1 is the begin column.

REPRO -- REPRODUCE FOLLOWING CARD

The REPRO assembler-instruction causes data on the following statement line to be punched into a card; the data is not processed. Neither a sequence number nor the identification is punched on the card. One REPRO instruction produces one punched card.

A REPRO statement that occurs before all program sectioning and linking instructions and before any assembler language instruction that may either affect or depend upon the setting of the location counter causes the assembler to punch a card that precedes all other cards of the object deck. A REPRO statement that occurs after any of the program sectioning and linking instructions has been encountered causes the assembler to punch a card that follows the object cards produced for all of the program sectioning and linking instructions. (The program sectioning and linking instructions are: START, CSECT, DSECT, COM, ENTRY, and EXTRN.)

The format of the REPRO instruction statement is:

Name	Operation	Operand
Blank	REPRO	Must be blank

The line to be reproduced may contain any combination of valid characters, starting in column 1 and continuing through column 72 of the line. Column 1 of the line corresponds to column 1 of the card to be punched.

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. An 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 the next available (unused) location for that control section.

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

Name	Operation	Operand
	ORG	*-500

If it is desired to reset the location counter to 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 since the previous LTORG (or start of the current control section) to be assembled at appropriate boundaries starting at the first double-word boundary following the LTORG statement. If no such literals exist, alignment of the next instruction (which is not a LTORG instruction) will occur. Bytes skipped are not set to zero. The format of the LTORG instruction statement is:

Name	Operation	Operand
A symbol or blank	LTORG	Must be blank

The symbol represents the address of the first byte of the literal pool. The LTORG statement forces all literals in a control section to be generated as a part of that control section. A LTORG statement must appear after the last reference to any literal in a control section.

Duplicate Literals

If duplicate literals occur within the range controlled by one LTORG statement, only one literal is stored. Literals are considered duplicates only if their specifications are identical. A literal will be stored, even if it appears to duplicate another literal, if it is an A-type address constant containing any reference to the location counter.

The following examples illustrate how the assembler stores pairs of literals, if the placement of each pair is controlled by the same LTORG statement.

```
X'F0'
          Both are stored
C'0'

XL3'0'
          Both are stored
HL3'0'

A(*+4)
          Both are stored
A(*+4)

X'FFFF'
          Identical; the first is stored
X'FFFF'

X'FF'
          Both are stored
XL1'FF'
```

CNOP -- CONDITIONAL NO OPERATION

The CNOP instruction allows the programmer to align an instruction at a specific halfword boundary. If any bytes must be skipped in order to align the instruction properly, the assembler ensures an unbroken instruction flow by generating no-operation instructions. This facility is useful in creating calling sequences that consist of a linkage to a subroutine followed by parameters.

The CNOP instruction ensures the alignment of the location counter setting to a halfword, fullword, 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 2 bytes.

<table>
<tr><td colspan="8" align="center">Double Word</td></tr>
<tr><td colspan="4" align="center">Fullword</td><td colspan="4" align="center">Fullword</td></tr>
<tr><td colspan="2" align="center">Halfword</td><td colspan="2" align="center">Halfword</td><td colspan="2" align="center">Halfword</td><td colspan="2" align="center">Halfword</td></tr>
<tr><td>Byte</td><td>Byte</td><td>Byte</td><td>Byte</td><td>Byte</td><td>Byte</td><td>Byte</td><td>Byte</td></tr>
<tr><td colspan="2">0,4
0,8</td><td colspan="2">2,4
2,8</td><td colspan="2">0,4
4,8</td><td colspan="2">2,4
6,8</td></tr>
</table>

Figure 5. CNOP Alignment

The format of the CNOP instruction statement is as follows:

Name	Operation	Operand
Blank	CNOP	Two absolute expressions of the form b,w

Any symbols used in the expressions in the operand field must previously have been defined. A location counter reference may not appear in such an expression.

Operand b specifies at which byte in a fullword 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 fullword (w = 4) or double word (w = 8). The following pairs of b and w are valid:

b,w	Specifies
0,4	Beginning of a fullword
2,4	Middle of a fullword
0,8	Beginning of a double word
2,8	Second halfword of a double word
4,8	Middle (third halfword) of a double word
6,8	Fourth halfword 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.

Assume that the location counter is currently aligned at a double word boundary. Then the CNOP instruction in the following sequence has no effect; it is merely printed in the assembly listing:

Name	Operation	Operand
	CNOP	0,8
	BALR	2,14

However, the following 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 halfword in a doubleword, 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 ensuring 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. A literal may not be used.

The typical form 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 may be transferred when loading is complete. This point is usually the first machine instruction in the program, as shown in the following sequence. If the operand field is blank, control is automatically transferred to the first byte of the first control section in the assembly.

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

The conditional assembly instructions allow the programmer to bypass source statements during an assembly, depending on the values assigned to variable symbols.

There are 4 conditional assembly instructions:

SETA AIF AGO ANOP

Note: Other System/360 programming support system assembler languages employ an additional conditional assembly instruction (the LCLA instruction) for the definition of variable symbols. Since the Model 44 Programming System Assembler Language does not require explicit definition of variable symbols beyond their appearance in the name field of a SETA instruction, LCLA statements are not required in the language. To enable an additional degree of compatibility, however, LCLA statements are treated as comments by this assembler.

The SETA instruction is used to define a variable symbol and assign an arithmetic value to it.

The AIF, AGO, and ANOP instructions are used (in conjunction with sequence symbols) to indicate which statements are to be processed by the assembler. The programmer can test the values assigned to variable symbols, thereby determining which statements are to be processed.

An example illustrating the use of conditional assembly instructions is included at the end of this section.

VARIABLE SYMBOLS

A variable symbol is a type of symbol that is assigned different values by the programmer. A variable symbol is written as an ampersand followed by from one to seven letters and/or digits, the first of which must be a letter. A variable symbol may be used in any operand where a self-defining term is allowed.

SETA -- SET ARITHMETIC

The SETA instruction is used to assign an arithmetic value to a variable symbol. A variable symbol is defined when it appears in the name field of a SETA instruction. The format of this instruction is:

Name	Operation	Operand
A variable symbol	SETA	An arithmetic expression

The expression in the operand field is evaluated as a signed 24-bit arithmetic value which is assigned to the variable symbol in the name field. The minimum and maximum allowable values of the expression are -2^{23} and $+2^{23}-1$, respectively.

The expression may consist of one term or an arithmetic combination of up to three terms. The terms may be either self-defining terms or variable symbols. (Self-defining terms are described in Section 1 of this publication.)

The arithmetic operators that may be used to combine the terms of an expression are + (addition), - (subtraction), * (multiplication), and / (division).

An expression may not contain two terms or two operators in succession, nor may it begin with an operator.

The following are valid operand fields of SETA instructions:

```
&AREA+X'2D'   &HERE-&EXIT
&BETA*10      29
```

The following are invalid operand fields of SETA instructions:

```
&AREAX'C'     (two terms in succession)
&FIELD+-3     (two operators in
                succession)
-&DELTA*2     (begins with an operator)
*+32          (begins with an operator;
                two operators in
                succession)
NAME/25       (NAME is not a valid term)
```

Evaluation of Arithmetic Expressions

The procedure used to evaluate the arithmetic expression in the operand field of a SETA instruction is the same as that used to evaluate arithmetic expressions in assembler language statements. The only difference between the two types of arithmetic expressions is the terms that are allowed in each expression.

The following evaluation procedure is used:

1. Each term is given its numerical value.

2. If a variable symbol is used, the arithmetic value assigned to it is substituted. If no arithmetic value has been assigned to the variable symbol, it is assumed the value is zero.

3. The arithmetic operations are performed moving from left to right. However, multiplication and/or division are performed before addition and subtraction.

4. The computed result is the value assigned to the variable symbol in the name field.

One level of parentheses may be used in a SETA operand. Each term enclosed by parentheses counts toward the maximum of three terms.

The following are examples of SETA instruction operand fields that contain parenthesized terms:

```
(&HERE+32)*29
&EXIT/(&ENTRY-4)
&BETA+(&ENTRY*2)
```

The parenthesized portion of an arithmetic expression is evaluated before the other term in the expression is evaluated.

LOGICAL EXPRESSIONS

Logical expressions enable the programmer to test the values assigned to variable symbols. A logical expression is used in the operand field of an AIF instruction and is evaluated to determine if it is true or false.

A logical expression consists of an arithmetic relation enclosed by parentheses. No blanks should appear between the enclosing parentheses and the first or last character of the arithmetic relation.

An arithmetic relation consists of two arithmetic expressions connected by a relational operator. The relational operator must be immediately preceded and followed by at least one blank. The relational operators are EQ (equal), NE (not equal), LT (less than), GT (greater than), LE (less than or equal), and GE (greater than or equal).

The arithmetic expressions that may be used in an arithmetic relation are limited to those expressions that are valid in the operand field of a SETA instruction.

The following are valid logical expressions:

```
(7*(&ALPHA+6) EQ &GAMMA)
(&AREA+2 GT 29)
(&XYZ NE &P12*(&A+6))
```

The following are invalid logical expressions:

&B EQ &A	(not enclosed in parentheses)
(&P EQ &B 6)	(two terms in succession)
(&A EQ 5)	(blank following left parenthesis and preceding right parenthesis)

SEQUENCE SYMBOLS

Sequence symbols provide the programmer with the ability to vary the sequence in which statements are processed by the assembler.

A sequence symbol may be used in the name field of any statement (except an ICTL statement) that does not require an ordinary symbol.

A sequence symbol is used in the operand field of an AIF or AGO statement to refer to the statement named by the sequence symbol.

A sequence symbol consists of a period followed by one to seven letters and/or digits, the first of which must be a letter.

The following are valid sequence symbols:

```
.READER   .A23456
.LOOP2    .X4F2
.N        .S4
```

The following are invalid sequence symbols:

CARDAREA	(first character is not a period)
.456B	(first character after period is not a letter)
.AREA2456	(more than seven characters after period)

.BCD%84 (contains a special character other than initial period)

.IN AREA (contains a special character, i.e., blank, other than initial period)

AIF -- CONDITIONAL BRANCH

The AIF instruction is used to conditionally alter the sequence in which source program statements are processed by the assembler. The assembler assigns a maximum count of 4096 AIF and AGO branches that may be executed in the source program. The format of this instruction is:

Name	Operation	Operand
A sequence symbol or blank	AIF	A logical expression immediately followed by a sequence symbol

Any logical expression may be used in the operand field of an AIF instruction. The sequence symbol in the operand field must immediately follow the closing parenthesis of the logical expression.

The logical expression in the operand field is evaluated to determine if it is true or false. If the expression is false, the next sequential statement is processed by the assembler. If the expression is true, the statement named by the sequence symbol in the operand field is the next statement processed by the assembler. This statement must not precede the AIF instruction.

The following are valid operand fields of AIF instructions:

```
(&AREA+X'2D' GT 29).READER
((32-&HERE)*4 GT 48).THERE
```

The following are invalid operand fields of AIF instructions:

```
(&HERE NE 6)          (no sequence symbol)
.X4F2                 (no logical expression)
(&N+4 GT &L)   .PASS  (blanks between logical
                       expression and sequence
                       symbol)
```

AGO -- UNCONDITIONAL BRANCH

The AGO instruction is used to unconditionally alter the sequence in which source program statements are processed by the assembler. The assembler assigns a maximum count of 4096 AIF and AGO branches that may be executed in the source program. The format of this instruction is:

Name	Operation	Operand
A sequence symbol or blank	AGO	A sequence symbol

The statement named by the sequence symbol in the operand field is the next statement processed by the assembler. This statement must not precede the AGO instruction.

ANOP -- ASSEMBLY NO OPERATION

The ANOP instruction facilitates conditional and unconditional branching to statements named by symbols or variable symbols. The format of this instruction is:

Name	Operation	Operand
A sequence symbol	ANOP	Blank

If the programmer wants to use an AIF or AGO instruction to branch to another statement, he must place a sequence symbol in the name field of the statement to which he wants to branch. However, if the programmer has already entered a symbol or variable symbol in the name field of that statement, he cannot place a sequence symbol in the name field. Instead, the programmer must place an ANOP instruction before the statement and then branch to the ANOP instruction. This has the same effect as branching to the statement immediately after the ANOP instruction.

USING CONDITIONAL ASSEMBLY INSTRUCTIONS

The following coding is an example of the use of conditional assembly instructions within a program.

	Name	Operation	Operand
	&ALPHA	SETA	3
	&BETA	SETA	4
		.	
		.	
		.	
		AIF	(&ALPHA EQ &BETA).SKIP
1		LA	5,4
2		AR	5,3
3		AGO	.OTHER
4	.SKIP	ANOP	
5	HERE	LA	5,7
6		AR	5,4
7	.OTHER	SR	6,5
		.	
		.	
		.	

The SETA instructions define the variable symbols &ALPHA and &BETA and assign to them the arithmetic values 3 and 4, respectively. The AIF instruction tests these values to determine the next statement to be processed by the assembler. The logical expression (&ALPHA EQ &BETA) is evaluated and found to be false: &ALPHA is not equal in value to &BETA. Therefore, the assembler continues with the next sequential statement. Statements 1 and 2 are processed. The AGO instruction then causes the assembler unconditionally to bypass statements until it encounters a statement with the sequence symbol .OTHER in the name field. Thus, statements 4, 5, and 6 are bypassed; statement 7 is the next statement processed.

For a subsequent assembly, the SETA instructions can be replaced so that the values of &ALPHA and &BETA are equal. Under these circumstances the logical expression in the AIF instruction is true and the next statement to be processed is the statement with the sequence symbol .SKIP in the name field. Thus, the assembler bypasses statements 1, 2, and 3 and processes statement 4. This is an ANOP instruction, required only because the statement following it has an ordinary symbol in the name field. Statements 5, 6, and 7 are then processed by the assembler.

SECTION 7: UPDATE FEATURE

The assembler has an update feature which allows the user to update a serialized, card-image data set by inserting, replacing, or deleting one or more records. The update feature can also be used to serialize a data set for use in subsequent update operations, or to serialize while updating. If the data set consists of source language statements written in assembler language, the assembler may be instructed to assemble the updated source data set concurrently with the update process.

INPUT/OUTPUT CONSIDERATIONS

The update feature may be used to modify any EBCDIC data set that consists of cards or card images, either blocked or unblocked, provided columns 73 through 80 of each card image are available for or contain a valid serial number. A valid serial number consists of 8 alphameric, non-blank characters. It usually contains a low-order, numeric field sufficient to accommodate numerical sequencing (in increments of 10) of the entire data set.

An object deck produced by the assembler can be updated only if produced during an assembly that used a TITLE instruction with a non-blank name field. The 4-character name field of the TITLE instruction is reproduced in columns 73 through 76 of the object program output and, in conjunction with sequence numbers punched automatically in columns 77 through 80, provides a valid serial number for a subsequent update operation.

The update input generally consists of two data sets: an old data set, and an edit data set. Updating of the old data set proceeds under the control of the edit data set.

The old data set consists of card images in sequence by serial number. The edit data set consists of additional card-image data and control statements, and must also be in sequence by serial number, except as noted below. Certain update modes of operation do not require an old data set.

The update output consists of an updated data set and an update listing. (Production of the updated data set is optional for an update-and-assemble run.) The update listing is separate from the program listing produced by the assembler. It describes the results of the update operation, and includes diagnostic error messages for any errors encountered while updating.

The user specifies a mode of operation appropriate to his input/output requirements in a control card supplied at assembler execution time. The various options and associated input/output assignments are discussed in the publication IBM System/360 Model 44 Programming System: Guide to System Use, Form C28-6812.

UPDATE OPERATION

PROCEDURE

The update feature uses the EBCDIC collating sequence to compare the serial numbers of the current records of the old data set and the edit data set. As long as the edit serial is greater, card images are passed from the old data set to the output data set. As soon as the edit serial is equal to or less than the old serial, the edit record is acted upon, as follows:

1. If the edit record is an update instruction, the instruction is performed. The specific actions relating to each update instruction are described under "Update Instructions," below. Generally, their performance involves a repositioning of the old data set, subsequent to which the edit data set is advanced to the next record, and a new comparison is initiated as above.

2. If the edit record is not an update instruction, the edit card image is inserted into the output data set and the edit data set is advanced to the next record. If the comparison yielded an equality, the old data set is also advanced to the next record, thus effecting replacement of the old record with the corresponding edit record. A new comparison is then initiated, and processing continues as above.

Unserialized card images (columns 73 through 80 are all blank) may appear in either data set at any time. They are considered to have the lowest value, and are processed immediately upon being encountered. If, however, blank serialization occurs simultaneously in both data sets, the edit record is acted upon.

Note: The following considerations apply during an update-and-assemble operation in which both an old data set and an edit data set are employed. Because the assembler immediately processes an end-of-data (/*) statement with blank serialization encountered in the edit data set, it will not process records remaining in the old data set. To enable processing of the old data set in its entirety, it is necessary to either: (1) serialize the end-of-data statement in such a way that its serial number is greater than the serial number of the last record in the old data set, or (2) ensure that the record immediately preceding the end-of-data statement in the edit data set acts, as required, upon or through the last record in the old set. During an update-only operation, in which job definition statements are not recognized as such, the only requirement is that the end-of-data statement follow the special control statement (the ENDUP instruction) required to terminate an update-only operation. (For a complete discussion of the end-of-data statement, see the publication, IBM System/360 Model 44 Programming System: Guide to System Use, Form C28-6812.)

INSERTION AND REPLACEMENT

Card images may be inserted or replaced without using update instructions. New card images are placed in the edit data set and processed as follows:

- A serialized card image in the edit data set replaces a card image with matching serialization in the old data set.
- A card image with unmatching serialization is inserted in sequence.
- Unserialized card images are inserted immediately upon being encountered.

If only the first of a group of cards in the edit data set contains a serial number, the entire group is inserted.

Card images inserted or replaced are noted as such in the update listing.

UPDATE INSTRUCTIONS

Functions other than insertion and replacement are specified by update instructions inserted by the user in the edit data set. The OMIT and CPYTO instructions allow a specified segment of the old data set to be omitted from or copied to the new data set. The SKPTO and REWND instructions enable a specified repositioning of the old data set, either forward to a specified serial number, or backward to the first record.

Two additional instructions may be used. The NUM instruction causes reserialization of the new data set. The ENDUP instruction terminates an update-only operation.

A detailed description of each update instruction follows. The following considerations apply:

1. Update instructions in the edit data set function as control statements to the update feature. They are not inserted into the new data set, which may, subsequently or concurrently, be processed by the language translation facility of the assembler. (If, however, an update instruction appears in the old data set, it will not be recognized as such, will not affect the update process, and will be inserted into the new data set unless replaced or deleted by the edit data set.) Update instructions are invalid as input during an assemble-only run.
2. The OMIT, SKPTO, CPYTO, and REWND instructions refer to an old data set. If encountered during a run in which an old data set is not used, they are flagged as invalid in the update listing and ignored.
3. The NUM and ENDUP instructions may use the serial field to refer to an old data set. Hence, the serial field should be blank when these instructions are used during a run in which an old data set is not used. If, during such a run, a NUM or an ENDUP instruction with a nonblank serial field is encountered, the serial number is ignored, and the instruction is acted upon at once.

NUM Instruction

The NUM instruction is used to reserialize columns 73 through 80 of the records in the new data set. The format of the instruction is:

Name	Operation	Operand	Serial
Ignored	NUM	A serial number or blank	A serial number or blank

The operand field contains 8 characters that specify the starting value of the new serial number. The number in the serial field identifies the record in the old data set at which reserialization is to begin.

When a NUM instruction is encountered, records are read from the old data set and written into the new data set until a serial number is encountered that matches the serial number of the instruction. The operation is then initiated, and continues until a new NUM instruction is encountered or until the last record has been numbered. A NUM instruction with a blank operand field may be used to terminate reserialization of the new data set.

If the serial field of a NUM instruction is blank, reserialization is initiated or terminated immediately.

Reserialization from record to record is done in increments of ten. The rightmost portion of the serial number specified in the operand field must contain enough numeric positions to prevent incrementation from affecting a non-numeric character.

As an example, consider the following statement:

Name	Operation	Operand	Serial
	NUM	IJK00000	ABC24710

Starting with the record in the old data that has serial number ABC24710, records in the new data set will be serialized IJK00000, IJK00010, IJK00020, etc.

OMIT Instruction

The OMIT instruction causes deletion of one or more records appearing in the old data set. Deleted records are noted as such in the update listing. The format of the OMIT instruction is:

Name	Operation	Operand	Serial
Ignored	OMIT	A serial number or blank	A serial number

The serial field contains the serial number at which deletion begins. The operand field contains the serial number at which deletion ends. Those records that have serial numbers that are equal to or between these two serial numbers will be deleted.

When an OMIT instruction is encountered, records are read from the old data set into the new data set until a serial number is encountered that is greater than or equal to the serial number of the instruction. The operation is then initiated, and continues until a serial number is encountered that is greater than or equal to the operand field of the instruction.

If the operand field is blank, the operation is terminated subsequent to the deletion of the single record specified.

If the serial field is blank, the statement is flagged as erroneous in the update listing and ignored.

SKPTO Instruction

The SKPTO instruction causes the bypassing of one or more records that appear in the old data set. Bypassed records do not appear in the new data set, are not assembled, and do not appear in the update listing.

The format of the SKPTO instruction is:

Name	Operation	Operand	Serial
Ignored	SKPTO	A serial number	A serial number or blank

The serial field contains the serial number of the record immediately preceding the first record to be bypassed. The operand field contains the serial number of the record immediately following the last record to be bypassed. Note that the records whose serial numbers are specified are not themselves bypassed.

When a SKPTO instruction is encountered, records are read from the old data set and written into the new data set until a serial number is encountered that matches the serial number of the instruction. The operation is then initiated, and continues until a serial number is encountered that matches the operand field of the instruction.

If the serial field is blank, the operation is initiated immediately.

CPYTO Instruction

The CPYTO instruction causes one or more records to be copied from the old data set into the new data set. Copied records are not assembled and do not appear in the update listing. The format of the CPYTO instruction is:

Name	Operation	Operand	Serial
Ignored	CPYTO	A serial number	A serial number or blank

The serial field contains the serial number of the record immediately preceding the first record to be copied. The operand field contains the serial number of the record immediately following the last record to be copied. Note that the records whose serial numbers are specified are not themselves copied.

When a CPYTO instruction is encountered, records are read from the old data set and written into the new data set until a serial number is encountered that matches the serial number of the instruction. The operation is then initiated, and continues until a serial number is encountered that matches the operand field of the instruction.

If the serial field is blank, the operation is initiated immediately.

REWND Instruction

The REWND instruction causes the unit holding the old data set to be repositioned to the first record in the data set. In conjunction with the SKPTO instruction, it allows the user to rearrange major segments of the old data set.

The format of the REWND instruction is:

Name	Operation	Operand	Serial
Ignored	REWND	Blank	A serial number or blank

When a REWND instruction is encountered, records are read from the old data set and written into the new data set until a serial number is encountered that matches the serial number of the instruction. The corresponding record is then inserted into the new data set, subsequent to which the repositioning of the old data set is performed.

If the serial field of the instruction is blank, repositioning takes place immediately.

ENDUP Instruction

The ENDUP instruction is required to terminate an update-only operation. ENDUP should not be issued when the data set is being both updated and assembled. The format of the ENDUP instruction is:

Name	Operation	Operand	Serial
Ignored	ENDUP	Blank	A serial number or blank

When an ENDUP instruction is encountered, records are read from the old data set and written into the new data set until a serial number is encountered that matches the serial number of the instruction. The corresponding record is then inserted into the new data set, subsequent to which the update operation is terminated.

If the serial field of the instruction is blank, updating is terminated immediately.

Note: An ENDUP instruction must appear in the edit data set during an update-only run. It is the only input that delimits the edit data set. If the assembler does not encounter an ENDUP instruction, it will continue to process the edit data set until it encounters end-of-file.

SEQUENCE CHECKING

During an update operation that employs both an old data set and an edit data set, the serialization of the edit data set is checked for proper sequencing. In general, a nonblank serial number must be higher than the preceding nonblank serial number in the edit data set. Blank serialization may, however, occur at any time, and does not constitute a sequence error.

Two exceptions apply. The serial number of a NUM instruction may be equal to (but not greater than) the serial number of the next record in the edit data set. The serial number of the edit record following a REWND instruction may assume any value.

An additional requirement applies to those update instructions (viz., OMIT, SKPTO, and CPYTO) that specify a range of operation by using the serial field and the operand field, respectively, to identify the initial and terminal extents of the range. For such instructions, a nonblank operand field participates in the sequence check, and must conform to one of the following two requirements:

- A nonblank operand of an OMIT instruction must be greater than the serial number of the instruction and smaller than the serial number of the next record in the edit data set.
- The operand field of a SKPTO or CPYTO instruction must be greater than the serial number of the instruction and smaller than or equal to the serial number of the next record in the edit data set.

If a serialization error is encountered, an error message is written in the update listing. Further action depends on the type of statement being processed, as follows:

- If the record is not an update instruction, it is inserted immediately into the new data set.
- If the record is an OMIT instruction, the instruction is ignored.
- If the record is any other update instruction, records are read from the old data set and written into the new data set until a matching serial number is found.

Note that update does not recognize sequence errors in the old data set. Their occurrence may cause unpredictable results.

EXAMPLES OF UPDATE OPERATION

Each of the following examples desribes an update operation by specifying two input data sets and the resulting output data set.

Example 1: The old data set is as follows:

Columns 1-72	Columns 73-80
Old Data 1	EX100010
Old Data 2	EX100020
Old Data 3	EX100030
Old Data 4	EX100040
Old Data 5	EX100050
Old Data 6	EX100060
Old Data 7	EX100070
Old Data 8	EX100080
Old Data 9	EX100090
Old Data 10	EX100100
Old Data 11	EX100110

The user wishes to insert Edit Data 1 between Old Data 2 and Old Data 3; to delete Old Data 5 and replace it with Edit Data 2, 3, and 4; to omit Old Data 7, 8, and 9; and to insert Edit Data 5 between Old Data 10 and Old Data 11. The following edit data set will perform these modifications, during an update-only run:

Columns 1-72	Columns 73-80
Edit Data 1	EX100025
Edit Data 2	EX100050
Edit Data 3	(Blank)
Edit Data 4	(Blank)
OMIT EX100090	EX100070
Edit Data 5	EX100105
ENDUP	EX100110

The new data set resulting from this operation is as follows:

Columns 1-72	Columns 73-80
Old Data 1	EX100010
Old Data 2	EX100020
Edit Data 1	EX100025
Old Data 3	EX100030
Old Data 4	EX100040
Edit Data 2	EX100050
Edit Data 3	(Blank)
Edit Data 4	(Blank)
Old Data 6	EX100060
Old Data 10	EX100100
Edit Data 5	EX100105
Old Data 11	EX100110

Example 2: The user wishes to rearrange and reserialize the following old data set:

Columns 1-72	Columns 73-80
Old Data 1	EX2A0010
Old Data 2	EX2A0020
Old Data 3	EX2A0030
Old Data 8	EX2A0040
Old Data 9	EX2A0050
Old Data 10	EX2A0060
Old Data 7	EX2A0070
Old Data 4	EX2A0080
Old Data 5	EX2A0090
Old Data 6	EX2A0100
Old Data 11	EX2A0110

In order to accomplish this, the following edit data set is employed, during an update-only run.

Columns 1-72	Columns 73-80
NUM EX2B0010	(Blank)
SKPTO EX2A0080	EX2A0030
REWND	EX2A0100
SKPTO EX2A0070	(Blank)
REWND	EX2A0070
SKPTO EX2A0040	(Blank)
SKPTO EX2A0110	EX2A0060
ENDUP	EX2A0110

The resulting new data set is as follows:

Columns 1-72	Columns 73-80
Old Data 1	EX2B0010
Old Data 2	EX2B0020
Old Data 3	EX2B0030
Old Data 4	EX2B0040
Old Data 5	EX2B0050
Old Data 6	EX2B0060
Old Data 7	EX2B0070
Old Data 8	EX2B0080
Old Data 9	EX2B0090
Old Data 10	EX2B0100
Old Data 11	EX2B0110

Note the significance of the serial numbers on the second REWND instruction and the ENDUP instruction. These serial numbers are required to inhibit the effects of an immediate action that would result in the loss of Old Data 7 and Old Data 11 in the new data set. This distinction is illustrated by use of the following edit data set, in which the above-mentioned serialization does not appear:

Columns 1-72	Columns 73-80
NUM EX2B0010	(Blank)
SKPTO EX2A0080	EX2A0030
REWND	EX2A0100
SKPTO EX2A0070	(Blank)
REWND	(Blank)
SKPTO EX2A0040	(Blank)
SKPTO EX2A0110	EX2A0060
ENDUP	(Blank)

The following new data set is produced:

Columns 1-72	Columns 73-80
Old Data 1	EX2B0010
Old Data 2	EX2B0020
Old Data 3	EX2B0030
Old Data 4	EX2B0040
Old Data 5	EX2B0050
Old Data 6	EX2B0060
Old Data 8	EX2B0070
Old Data 9	EX2B0080
Old Data 10	EX2B0090

Example 3: The user wishes to update and assemble one segment of an old data set, and to create a new data set which contains the updated version of this segment. The user wishes to assemble another segment of the old data set without modification. The new data set must include this segment, as well as those segments of the old data set which the user does not wish to assemble, with the exception of one such segment, which he wishes to delete. The old data set is as follows:

Columns 1-72	Columns 73-80
Old Data 1	CPY10010
Old Data 2	CPY10020
Old Data 3	CPY10030
Old Data 4	DEL00010
Old Data 5	DEL00020
Old Data 6	DEL00030
Old Data 7	ASM10010
Old Data 8	ASM10020
Old Data 9	ASM10030
Old Data 10	CPY20010
Old Data 11	CPY20020
Old Data 12	CPY20030
Old Data 13	ASM20010
Old Data 14	ASM20020
Old Data 15	ASM20030
Old Data 16	ASM20040
Old Data 17	ASM20050
Old Data 18	ASM20060
Old Data 19	ASM20070
Old Data 20	CPY30010
Old Data 21	CPY30020
Old Data 22	CPY30030

The user specifies an update-and-assemble operation under the control of the the following edit data set:

Columns 1-72	Columns 73-80
CPYTO DEL00010	(Blank)
SKPTO ASM10010	(Blank)
CPYTO ASM20010	ASM10030
NUM ASM20010	(Blank)
Edit Data 1	ASM20025
Edit Data 2	(Blank)
Edit Data 3	ASM20050
NUM	CPY30010
CPYTO 99999999	(Blank)

A new data set is produced as shown below; records marked with an asterisk have been assembled.

Columns 1-72	Columns 73-80
Old Data 1	CPY10010
Old Data 2	CPY10020
Old Data 3	CPY10030
Old Data 7*	ASM10010
Old Data 8*	ASM10020
Old Data 9*	ASM10030
Old Data 10	CPY20010
Old Data 11	CPY20020
Old Data 12	CPY20030
Old Data 13*	ASM20010
Old Data 14*	ASM20020
Edit Data 1*	ASM20030
Edit Data 2*	ASM20040
Old Data 15*	ASM20050
Old Data 16*	ASM20060
Edit Data 3*	ASM20070
Old Data 18*	ASM20080
Old Data 19*	ASM20090
Old Data 20	CPY30010
Old Data 21	CPY30020
Old Data 22	CPY30030

Note that Example 3 will result in the production of a warning message. The operand of the last instruction in the edit data set does not correspond to a serial number in the old data set. It is specified, however, so that the CPYTO operation will act through the last record in the old data set. A warning message will appear in the update listing.

8-Bit BCD Code	Character Set Punch Combination	Decimal	Hexa-decimal	Printer Graphics
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	

8-Bit BCD Code	Character Set Punch Combination	Decimal	Hexa-decimal	Printer Graphics
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		64	40	blank
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	75	4B	. (period)
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	, (comma)

8-Bit BCD Code	Character Set Punch Combination	Decimal	Hexa-decimal	Printer Graphics
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	' (single quote)
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	

8-Bit BCD Code	Character Set Punch Combination	Decimal	Hexa-decimal	Printer Graphics
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	193	C1	A
11000010	12,2	194	C2	B
11000011	12,3	195	C3	C
11000100	12,4	196	C4	D
11000101	12,5	197	C5	E
11000110	12,6	198	C6	F
11000111	12,7	199	C7	G
11001000	12,8	200	C8	H
11001001	12,9	201	C9	I
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	209	D1	J
11010010	11,2	210	D2	K
11010011	11,3	211	D3	L
11010100	11,4	212	D4	M
11010101	11,5	213	D5	N
11010110	11,6	214	D6	O
11010111	11,7	215	D7	P
11011000	11,8	216	D8	Q
11011001	11,9	217	D9	R
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	

8-Bit BCD Code	Character Set Punch Combination	Decimal	Hexa-decimal	Printer Graphics
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	226	E2	S
11100011	0,3	227	E3	T
11100100	0,4	228	E4	U
11100101	0,5	229	E5	V
11100110	0,6	230	E6	W
11100111	0,7	231	E7	X
11101000	0,8	232	E8	Y
11101001	0,9	233	E9	Z
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	240	F0	0
11110001	1	241	F1	1
11110010	2	242	F2	2
11110011	3	243	F3	3
11110100	4	244	F4	4
11110101	5	245	F5	5
11110110	6	246	F6	6
11110111	7	247	F7	7
11111000	8	248	F8	8
11111001	9	249	F9	9
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	

APPENDIX B: HEXADECIMAL-DECIMAL NUMBER CONVERSION TABLE

The table provides direct conversion of decimal and hexadecimal numbers in the following ranges:

Hexadecimal	Decimal
000 to FFF	0000 to 4095

Decimal numbers (0000-4095) are given within the table. The first two characters (high-order) of hexadecimal numbers (000-FFF) are given in the left-hand column of the table; the third character (x) is arranged across the top of each part of the table.

To find the decimal equivalent of the hexadecimal number 0C9, look for 0C in the left-hand column, and across that row under the column for x = 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 hexadecimal characters in the left-hand column, and the value for x at the top of the column containing the decimal number. For example, the decimal number 123 has the hexadecimal equivalent of 07B; the decimal number 1478 has the hexadecimal equivalent of 5C6.

For numbers outside the range of the table, add the following values to the table.

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

	x = 0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00x	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
01x	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
02x	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
03x	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
04x	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
05x	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
06x	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
07x	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
08x	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
09x	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0Ax	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0Bx	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0Cx	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0Dx	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0Ex	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0Fx	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255
10x	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
11x	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
12x	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
13x	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
14x	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
15x	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
16x	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
17x	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
18x	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
19x	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1Ax	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1Bx	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1Cx	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1Dx	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1Ex	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1Fx	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511
20x	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
21x	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
22x	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
23x	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
24x	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
25x	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
26x	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
27x	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
28x	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
29x	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2Ax	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2Bx	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2Cx	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2Dx	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2Ex	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2Fx	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767
30x	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
31x	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
32x	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
33x	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
34x	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
35x	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
36x	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
37x	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
38x	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
39x	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3Ax	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3Bx	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3Cx	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3Dx	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3Ex	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3Fx	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023

x =	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
40x	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
41x	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
42x	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
43x	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
44x	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
45x	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
46x	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
47x	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
48x	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
49x	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4Ax	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4Bx	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4Cx	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4Dx	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4Ex	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4Fx	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
50x	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
51x	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
52x	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
53x	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
54x	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
55x	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
56x	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
57x	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
58x	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
59x	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5Ax	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5Bx	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5Cx	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5Dx	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5Ex	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5Fx	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535
60x	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
61x	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
62x	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
63x	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
64x	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
65x	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
66x	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
67x	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663
68x	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
69x	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6Ax	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6Bx	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6Cx	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6Dx	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6Ex	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6Fx	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791
70x	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
71x	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
72x	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
73x	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
74x	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
75x	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
76x	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
77x	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
78x	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
79x	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7Ax	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7Bx	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7Cx	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7Dx	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7Ex	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7Fx	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047

x =	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
80x	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
81x	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
82x	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
83x	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
84x	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
85x	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
86x	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
87x	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
88x	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
89x	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8Ax	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8Bx	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8Cx	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8Dx	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8Ex	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8Fx	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303
90x	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
91x	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
92x	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
93x	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
94x	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
95x	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
96x	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
97x	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
98x	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
99x	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9Ax	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9Bx	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9Cx	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9Dx	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9Ex	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9Fx	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559
A0x	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A1x	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A2x	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A3x	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A4x	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A5x	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A6x	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A7x	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A8x	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A9x	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AAx	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
ABx	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
ACx	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
ADx	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AEx	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AFx	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
B0x	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B1x	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B2x	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B3x	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B4x	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B5x	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B6x	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B7x	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B8x	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B9x	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BAx	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BBx	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BCx	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BDx	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BEx	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BFx	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071

x =	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C0x	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C1x	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C2x	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C3x	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C4x	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C5x	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C6x	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C7x	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C8x	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C9x	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CAx	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CBx	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CCx	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CDx	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CEx	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CFx	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327
D0x	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D1x	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D2x	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D3x	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D4x	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D5x	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D6x	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D7x	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D8x	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D9x	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DAx	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DBx	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DCx	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DDx	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DEx	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DFx	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583
E0x	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E1x	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E2x	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E3x	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E4x	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E5x	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E6x	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E7x	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E8x	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E9x	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EAx	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EBx	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
ECx	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
EDx	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EEx	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EFx	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839
F0x	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F1x	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F2x	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F3x	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F4x	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F5x	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F6x	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F7x	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F8x	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F9x	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FAx	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FBx	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FCx	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FDx	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FEx	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FFx	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

	Basic Machine Format	Assembler Operand Field Format	Applicable Instructions
RR	8 Operation Code; 4 R1; 4 R2	R1,R2	All RR instructions except SPM and SVC
	8 Operation Code; 4 R1; ~~4 R2~~	R1	SPM
	8 Operation Code; 8 I	I	SVC
		(See notes 1, 5, and 7)	
RX	8 Operation Code; 4 R1; 4 X2; 4 B2; 12 D2	R1,D2(X2,B2) R1,S2(X2)	All RX instructions
		(See notes 1 through 4, and 6)	
RS	8 Operation Code; 4 R1; 4 R3; 4 B2; 12 D2	R1,R3,D2(B2) R1,R3,S2	BXH,BXLE,LM,STM
	8 Operation Code; 4 R1; ~~4 R3~~; 4 B2; 12 D2	R1,D2(B2) R1,S2	All Shift instructions
		(See notes 1 through 3, 6, and 7)	
SI	8 Operation Code; 8 I2; 4 B1; 12 D1	D1(B1),I2 S1,I2	All SI instructions except CHPM,LPSW,LPSX, SSM,HIO,SIO,TIO,TCH,TS
	8 Operation Code; ~~8 I2~~; 4 B1; 12 D1	D1(B1) S1	CHPM,LPSW,LPSX,SSM,HIO, SIO,TIO,TCH,TS
		(See notes 2, 3, and 5 through 7)	

Notes for Appendix C:

1. R1, R2, and R3 are absolute terms 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. In BC and BCR machine instructions, R1 specifies a 4-bit mask.
2. D1 and D2 are absolute expressions that specify displacements. A value of 0 through 4095 may be specified.
3. B1 and B2 are absolute terms that specify base registers. Register numbers are 0 through 15.
4. X2 is an absolute term that specifies an index register. Register numbers are 0 through 15.
5. I and I2 are absolute expressions that provide immediate data. The value of the expression may be 0 through 255.
6. S1 and S2 are absolute or relocatable expressions that specify an address.
7. RR, RS, and SI instruction fields that are crossed out in the machine formats are not examined during instruction execution. The fields are not written in the symbolic operand, but are assembled as binary zeros.

APPENDIX D: MACHINE-INSTRUCTION OPERATION CODES

This appendix contains an alphabetical listing of the mnemonic operation codes of all the machine instructions that can be represented in assembler language. The column headings in the list and the information each column provides are as follows:

Mnemonic Code: This column gives the mnemonic operation code for the machine instruction.

Instruction: This column contains the name of the instruction associated with the mnemonic.

Machine Code: This column contains the hexadecimal equivalent of the actual machine operation code.

Basic Machine Format: This column gives the basic machine format of the instruction: RR, RX, RS, or SI.

Operand Field Format: This column shows the symbolic format of the operand field for the particular mnemonic.

Mnemonic Code	Instruction	Machine Code	Basic Machine Format	Operand Field Format
A	Add	5A	RX	R1,D2(X2,B2)
AD	Add Normalized, Long	6A	RX	R1,D2(X2,B2)
ADR	Add Normalized, Long	2A	RR	R1,R2
AE	Add Normalized, Short	7A	RX	R1,D2(X2,B2)
AER	Add Normalized, Short	3A	RR	R1,R2
AH	Add Halfword	4A	RX	R1,D2(X2,B2)
AL	Add Logical	5E	RX	R1,D2(X2,B2)
ALR	Add Logical	1E	RR	R1,R2
AR	Add	1A	RR	R1,R2
AU	Add Unnormalized, Short	7E	RX	R1,D2(X2,B2)
AUR	Add Unnormalized, Short	3E	RR	R1,R2
AW	Add Unnormalized, Long	6E	RX	R1,D2(X2,B2)
AWR	Add Unnormalized, Long	2E	RR	R1,R2
BAL	Branch and Link	45	RX	R1,D2(X2,B2)
BALR	Branch and Link	05	RR	R1,R2
BC	Branch on Condition	47	RX	M1,D2(X2,B2)
BCR	Branch on Condition	07	RR	M1,R2
BCT	Branch on Count	46	RX	R1,D2(X2,B2)
BCTR	Branch on Count	06	RR	R1,R2
BXH	Branch on Index High	86	RS	R1,R3,D2(B2)
BXLE	Branch on Index Low or Equal	87	RS	R1,R3,D2(B2)
C	Compare	59	RX	R1,D2(X2,B2)
CD	Compare, Long	69	RX	R1,D2(X2,B2)
CDR	Compare, Long	29	RR	R1,R2
CE	Compare, Short	79	RX	R1,D2(X2,B2)
CER	Compare, Short	39	RR	R1,R2
CH	Compare Halfword	49	RX	R1,D2(X2,B2)
CHPM	Change Priority Mask	B3	SI	D1(B1),I2
CL	Compare Logical	55	RX	R1,D2(X2,B2)
CLI	Compare Logical Immediate	95	SI	D1(B1),I2
CLR	Compare Logical	15	RR	R1,R2
CR	Compare Algebraic	19	RR	R1,R2
D	Divide	5D	RX	R1,D2(X2,B2)
DD	Divide, Long	6D	RX	R1,D2(X2,B2)
DDR	Divide, Long	2D	RR	R1,R2
DE	Divide, Short	7D	RX	R1,D2(X2,B2)
DER	Divide, Short	3D	RR	R1,R2
DR	Divide	1D	RR	R1,R2

Mnemonic Code	Instruction	Machine Code	Basic Machine Format	Operand Field Format
EX	Execute	44	RX	R1,D2(X2,B2)
HDR	Halve, Long	24	RR	R1,R2
HER	Halve, Short	34	RR	R1,R2
HIO	Halt I/O	9E	SI	D1(B1)
IC	Insert Character	43	RX	R1,D2(X2,B2)
ISK	Insert Storage Key	09	RR	R1,R2
L	Load	58	RX	R1,D2(X2,B2)
LA	Load Address	41	RX	R1,D2(X2,B2)
LCDR	Load Complement, Long	23	RR	R1,R2
LCER	Load Complement, Short	33	RR	R1,R2
LCR	Load Complement	13	RR	R1,R2
LD	Load, Long	68	RX	R1,D2(X2,B2)
LDR	Load, Long	28	RR	R1,R2
LE	Load, Short	78	RX	R1,D2(X2,B2)
LER	Load, Short	38	RR	R1,R2
LH	Load Halfword	48	RX	R1,D2(X2,B2)
LM	Load Multiple	98	RS	R1,R3,D2(B2)
LNDR	Load Negative, Long	21	RR	R1,R2
LNER	Load Negative, Short	31	RR	R1,R2
LNR	Load Negative	11	RR	R1,R2
LPDR	Load Positive, Long	20	RR	R1,R2
LPER	Load Positive, Short	30	RR	R1,R2
LPR	Load Positive	10	RR	R1,R2
LPSW	Load PSW	82	SI	D1(B1)
LPSX	Load PSW Special	B2	SI	D1(B1)
LR	Load	18	RR	R1,R2
LTDR	Load and Test, Long	22	RR	R1,R2
LTER	Load and Test, Short	32	RR	R1,R2
LTR	Load and Test	12	RR	R1,R2
M	Multiply	5C	RX	R1,D2(X2,B2)
MD	Multiply, Long	6C	RX	R1,D2(X2,B2)
MDR	Multiply, Long	2C	RR	R1,R2
ME	Multiply, Short	7C	RX	R1,D2(X2,B2)
MER	Multiply, Short	3C	RR	R1,R2
MH	Multiply Halfword	4C	RX	R1,D2(X2,B2)
MR	Multiply	1C	RR	R1,R2
MVI	Move Immediate	92	SI	D1(B1),I2
N	AND Logical	54	RX	R1,D2(X2,B2)
NI	AND Immediate	94	SI	D1(B1),I2
NR	AND Logical	14	RR	R1,R2
O	OR Logical	56	RX	R1,D2(X2,B2)
OI	OR Immediate	96	SI	D1(B1),I2
OR	OR Logical	16	RR	R1,R2
RDDW	Read Direct Word	B5	SI	D1(B1),I2
S	Subtract	5B	RX	R1,D2(X2,B2)
SD	Subtract Normalized, Long	6B	RX	R1,D2(X2,B2)
SDR	Subtract Normalized, Long	2B	RR	R1,R2
SE	Subtract Normalized, Short	7B	RX	R1,D2(X2,B2)
SER	Subtract Normalized, Short	3B	RR	R1,R2
SH	Subtract Halfword	4B	RX	R1,D2(X2,B2)
SIO	Start I/O	9C	SI	D1(B1)
SL	Subtract Logical	5F	RX	R1,D2(X2,B2)

Mnemonic Code	Instruction	Machine Code	Basic Machine Format	Operand Field Format
SLA	Shift Left Single Algebraic	8B	RS	R1,D2(B2)
SLDA	Shift Left Double Algebraic	8F	RS	R1,D2(B2)
SLDL	Shift Left Double Logical	8D	RS	R1,D2(B2)
SLL	Shift Left Single Logical	89	RS	R1,D2(B2)
SLR	Subtract Logical	1F	RR	R1,R2
SPM	Set Program Mask	04	RR	R1
SR	Subtract	1B	RR	R1,R2
SRA	Shift Right Single Algebraic	8A	RS	R1,D2(B2)
SRDA	Shift Right Double Algebraic	8E	RS	R1,D2(B2)
SRDL	Shift Right Double Logical	8C	RS	R1,D2(B2)
SRL	Shift Right Single Logical	88	RS	R1,D2(B2)
SSK	Set Storage Key	08	RR	R1,R2
SSM	Set System Mask	80	SI	D1(B1)
ST	Store	50	RX	R1,D2(X2,B2)
STC	Store Character	42	RX	R1,D2(X2,B2)
STD	Store, Long	60	RX	R1,D2(X2,B2)
STE	Store, Short	70	RX	R1,D2(X2,B2)
STH	Store Halfword	40	RX	R1,D2(X2,B2)
STM	Store Multiple	90	RS	R1,R3,D2(B2)
SU	Subtract Unnormalized, Short	7F	RX	R1,D2(X2,B2)
SUR	Subtract Unnormalized, Short	3F	RR	R1,R2
SVC	Supervisor Call	0A	RR	I
SW	Subtract Unnormalized, Long	6F	RX	R1,D2(X2,B2)
SWR	Subtract Unnormalized, Long	2F	RR	R1,R2
TCH	Test Channel	9F	SI	D1(B1)
TIO	Test I/O	9D	SI	D1(B1)
TM	Test Under Mask	91	SI	D1(B1),I2
TS	Test and Set	93	SI	D1(B1)
WRDW	Write Direct Word	B4	SI	D1(B1),I2
X	Exclusive OR	57	RX	R1,D2(X2,B2)
XI	Exclusive OR Immediate	97	SI	D1(B1),I2
XR	Exclusive OR	17	RR	R1,R2

APPENDIX E: ASSEMBLER INSTRUCTIONS

Operation Entry	Name Field	Operand Field
AGO	Sequence symbol optional	A sequence symbol
AIF	Sequence symbol optional	A logical expression immediately followed by a sequence symbol
ANOP	Sequence symbol required	Must be blank
CCW	Symbol optional	Four operands
CNOP	Must be blank	Two absolute expressions, separated by a comma
COM	Symbol optional	Must be blank
CSECT	Symbol optional[1]	Must be blank
DC	Symbol optional	One operand
DROP	Must be blank	One operand
DS	Symbol optional	One operand
DSECT	Symbol required	Must be blank
EJECT	Must be blank	Must be blank
END	Must be blank	A relocatable expression or blank
ENTRY	Must be blank	One operand
EQU	Symbol required	An absolute or relocatable expression
EXTRN	Must be blank	One operand
ICTL	Must be blank	The decimal value 1 or 25
LTORG	Symbol optional	Must be blank
ORG	Must be blank	A relocatable expression or blank
PRINT	Must be blank	One or two operands
REPRO	Must be blank	Must be blank
SETA	Variable symbol required	An arithmetic expression
SPACE	Must be blank	A decimal self-defining term or blank
START	Symbol optional	A self-defining term or blank
TITLE[2]	A special symbol (1 to 4 characters) or not present	One to 62 characters, enclosed in single quotes
USING	Must be blank	An absolute or relocatable expression followed by an absolute term

[1]A symbol is required if an unnamed START or CSECT instruction has already been used during this assembly.
[2]See Section 5 for the description of the name entry.

APPENDIX F: SUMMARY OF CONSTANTS

TYPE	IMPLIED LENGTH (BYTES)	ALIGNMENT	LENGTH SPECIFICATION RANGE	SPECIFIED BY	NUMBER OF CONSTANTS PER OPERAND	TRUNCATION/ PADDING SIDE
C	as needed	byte	1 to 256[1]	characters	one	right
X	as needed	byte	1 to 256[1]	hexadecimal digits	one	left
F	4	fullword	1 to 8	decimal digits	multiple	left
H	2	halfword	1 to 8	decimal digits	multiple	left
E	4	fullword	1 to 8	decimal digits	multiple	right
D	8	double word	1 to 8	decimal digits	multiple	right
A	4	fullword	1 to 4[2]	an expression	multiple	left

[1] In a DS assembler instruction C and X type constants may have length specification to 65,535.
[2] Absolute expressions may have 1 to 4 bytes. Relocatable expressions may have 3 or 4 bytes.

APPENDIX G: ASSEMBLER LANGUAGES--FEATURES COMPARISON CHECKLISTS

With certain exceptions, the IBM System/360 Model 44 Programming System Assembler Language is a selected subset of the languages available in the IBM System/360 programming support systems designed for the Models 30, 40, 50, 65, and 75 -- specifically, System/360 Operating System (OS/360), System/360 Disk Operating System (DOS/360), and System/360 Tape Operating System (TOS/360). This appendix uses three lists to describe this subset. The first of these lists comprises those features of the Model 44 Programming System Assembler Language that are not within the subset (i.e., those features peculiar to the Model 44 Assembler). The second and third lists comprise, respectively, those features of the other System/360 assembler languages that are not included in the Model 44 Programming System Assembler Language, and those that are included subject to the limitations noted.

1. Features of the IBM System/360 Model 44 Programming System Assembler Language that are not supported by the other System/360 assembler languages are, as follows:

   a. Update instructions

   b. Named common control sections

   c. Implicit definition of SETA symbols

   d. Mnemonic operation codes for the following machine instructions:

      (1) Change Priority Mask (CHPM)

      (2) Load PSW Special (LPSX)

      (3) Read Direct Word (RDDW)

      (4) Write Direct Word (WRDW)

2. Features of the IBM System/360 Operating System Assembler Language (and, in some cases, of some or all of the other System/360 programming support system assembler languages) that are not supported by the Model 44 Programming System Assembler Language are, as follows:

   a. Binary self-defining terms

   b. Continuation cards

   c. COPY instruction

   d. ISEQ instruction

   e. Macro instructions

   f. Mnemonic operation codes for machine instructions in the storage-to-storage (SS) format, or for any of the following machine instructions:

      (1) Convert to Binary (CVB)

      (2) Convert to Decimal (CVD)

      (3) Read Direct (RDD)

      (4) Write Direct (WRD)

   g. PUNCH instruction

3. Features of the IBM System/360 Operating System Assembler Language (and, in some cases, of some or all of the other System/360 programming support system assembler languages) only selected subsets of which are supported by the Model 44 Programming System Assembler Language are, as follows:

FEATURE	LIMITATIONS
CCW instruction	Operands 1, 3, and 4 must be specified as absolute terms.
Conditional assembly instructions	A limited subset is employed; Section 6 of this manual ("Conditional Assembly Instructions") describes this subset in detail.
CSECT instruction	Multiple CSECT instructions must have unique names (i.e., a control section, once terminated, cannot be resumed).
DC and DS instructions	One operand only; no bit-length specification; no scale or exponent modifier; only constants of types C, X, F, H, E, D, and A.
DROP instruction	One operand only, must be an absolute term.
DSECT instruction	Multiple DSECT instructions must have unique names (i.e., a dummy section, once terminated, cannot be resumed).
ENTRY instruction	One operand only.
EQU instruction	Operand may not contain an external symbol.
Expressions	Maximum of three terms; only one level of parentheses; no complex relocatability.
EXTRN instruction	One operand only.
ICTL instruction	One operand only (the begin column specification); operand must assume a value of 1 or 25.
Literal pool	The positioning of the literal pool must be assigned by the programmer, and a literal pool must be assigned following the last occurrence of a literal in any given control section.
Machine instruction operands	Operands R1, R2, B1, B2, and X2 (see Appendix C) must be written as absolute terms; multiterm absolute expressions are not permitted for these operands.
PRINT instruction	Two operands only (the GEN/NOGEN option is not supported).
REPRO instruction	Columns 1 through 72 only are reproduced.
TITLE instruction	Maximum of 62 characters in operand field (exclusive of delimiting single quotes).
USING instruction	Operand r must be an absolute term; only one operand r (i.e., only one general register) may be specified in a single USING instruction.

IBM Technical Newsletter

File No. S360-21
Re: Form No. C28-6811-1
This Newsletter No. N33-8543
Date: May 20, 1968
Previous Newsletter Nos. none

This Technical Newsletter amends the Systems Reference Library publication IBM System/360 Model 44, Programming System, Assembler Language
Form C28-6811-1

This Technical Newsletter reflects the addition to the Assembler Language of the BXH, BXLE, LM, STM, and EX instructions and makes certain minor corrections to the publication.
Substitute the following pages for the pages currently in your copy of the publication:

35 and 36
67 and 68
69 and 70
73 and 74

All changes and additions are indicated by a vertical line to the left of the affected portion of the text. Figures that were changed have a bullet (•) to the left of the figure caption.

File this cover letter at the back of the publication. It will then serve as a record of the changes received and incorporated.

IBM Laboratory, Publications Dept., Uithoorn Netherlands

PRINTED IN U.S.A.

fold fold

FIRST CLASS
PERMIT NO. 33504
NEW YORK, N.Y.

BUSINESS REPLY MAIL

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM CORPORATION

1271 AVENUE OF THE AMERICAS
NEW YORK, N.Y. 10020

Attention: PUBLICATIONS

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]