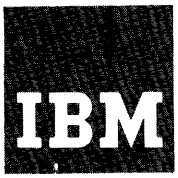
File No. S360-21
Form C28-6503-3



Systems Reference Library

IBM System/360 Basic Programming Support

Basic Assembler Language

This publication contains information required for writing programs in the Basic Assembler language, a symbolic programming language for the IBM System/360. The Basic Assembler language provides programmers with a convenient way to make full use of the operating capabilities of the IBM System/360. Source programs written in the Basic Assembler language are translated into object programs by a program called the Basic Assembler.

The Basic Assembler and its language are both described in this publication. The description of the language includes the rules for writing source programs, a list of the machine instructions that can be represented symbolically, and explanations of the instructions used to control the Basic Assembler. The description of the Basic Assembler consists primarily of discussions of those features that affect the planning and writing of source programs.

PREFACE

This publication describes a symbolic programming language for the IBM System/360. All the information required for writing IBM System/360 programs is provided. This includes the rules for writing source statements, a description of the assembler "instructions," and a list of the machine instructions that can be represented in the language. There is also a section describing the Basic Assembler, the program that translates source programs into machine-language programs. The information in this section will be helpful in planning for the IBM System/360 and in writing programs to permit the most efficient operation of the Basic Assembler. This section describes the input to the Basic Assembler, the type of output that will be generated, and those operations of the Basic Assembler that have direct programming significance.

Completion of a basic course in computer systems and programming concepts, or the equivalent, is a prerequisite to using this publication. Readers should also be familiar with the IBM System/360 and have an understanding of the storage-addressing scheme, data formats, and machine instruction formats and functions. This information can be found in the publication IBM System/360 Principles of Operation, Form A22-6821.

Reference is made in this manual to the relocating loader and the absolute loader. A detailed description of these programs is contained in the publication IBM System/360 Basic Programming Support Basic Utilities, Form C28-6505.

MAJOR REVISION (February 1965)

This publication is a major revision of the previous edition, Form C28-6503-2, which is now obsolete. Significant changes have been made throughout this publication, and the present edition should be reviewed in its entirety.

This publication was prepared for production using an IBM computer to update the text and to control the page and line format. Page impressions for photo-offset printing were obtained from an IBM 1403 Printer using a special print chain.

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

A form for readers' comments appears at the back of this publication. It may be mailed directly to IBM. Address any additional comments concerning this publication to the IBM Corporation, Programming Systems Publications, Department D58, PO Box 390, Poughkeepsie, N. Y. 12602

FIGURES

The Basic Assembler language is a symbolic programming language for use with the IBM System/360. This language provides programmers with a convenient means of writing machine instructions, designating registers and input/output devices, and specifying the format and addresses of storage areas, data, and constants. All the operational capabilities of the IBM System/360 can be expressed in Basic Assembler language programs.

The language features are designed to greatly simplify the writing of programs for the IBM System/360. By avoiding unnecessary complexity, the language features reduce program errors and, consequently, the time required to produce a program that is suitable for execution. They also make it easier to learn the language.

Source programs written in this language are translated into IBM System/360 machine language object programs by the Basic Assembler (that is, "the assembler"). In the process of translating programs, the assembler performs certain auxiliary functions. Some of these functions are automatically performed; others must be requested by special assembler instructions that the programmer writes in his source program.

The assembler is a two-phase program stored on cards. It has a special operating procedure for use with the IBM 1442 Card Read-Punch. When this procedure is used, the assembler punches information into the source-program deck during the first phase. Using this information in the second phase, the assembler produces an object program. For systems with tape or a 1402-2 Card Read-Punch, this intermediate information is stored in a tape or card file rather than in the source-program deck. The temporary file then serves as input for the second phase.

FEATURES OF THE IBM SYSTEM/360 BASIC ASSEMBLER

The most significant features provided by the assembler and its language are summarized below. This summary does not include all the features nor does it contain complete explanations of the features listed. For more detailed descriptions, the reader is referred to subsequent sections.

Mnemonic Operation Codes: Mnemonic operation codes are provided for all machine instructions. These codes are used instead of the more cumbersome internal operation codes of the machine. For example, the Branch-on-Condition instruction can be represented by the mnemonic BC instead of the machine operation code, 01000111. The various machine mnemonic operation codes are presented under the topic "Machine Instruction Mnemonics."

Symbolic Referencing of Storage Addresses: Instructions, data area, register numbers, and other program elements can be referred to by symbolic names, instead of actual machine addresses and designations. See the topic "Symbols."

Automatic Storage Assignment: The assembler assigns consecutive addresses to program elements as it encounters them. After processing each element, the assembler increments a counter by the number of bytes assigned to that element. This counter indicates the storage location available to the next element. See the topic "Location Counter."

Convenient Data Representation: Constants can be specified as decimal digits, alphabetic characters, hexadecimal digits, and storage addresses. Conversion of the data into the appropriate machine format of the IBM System/360 is performed by the assembler. Data can be in a form suitable for use in fixed-point and floating-point arithmetic operations. See the topic "DC - Define Constant."

Renaming Symbols: A symbolic name can be equated to another symbol so that both refer to the same storage location, general register, etc. This makes it possible for the same program item to be referred to by different names in different parts of the program. See the topic "EQU - Equate Symbol."

Program Linking: Independently assembled programs that will be loaded and executed together may make symbolic references to instructions and data in one another. See the discussion of "Program Link Instructions."

Relocatable Programs: The assembler produces object programs in a relocatable format; that is, a format that enables programs to be loaded and executed at storage locations different from those assigned when the programs were assembled.

Assembler Instructions: A set of special instructions to the assembler is included in the language. Some of the features described in this section are implemented by these instructions. See the topic "Assembler Instructions."

Base Register and Displacement Assignment: The programmer can instruct the assembler to assign base registers and to compute displacements for symbolic machine addresses. See the discussion of "Base Register Instructions."

Program Listings: For every assembly, the assembler can provide a listing of both the source program and the resulting object program. A description of the listing format can be found under the topic "Program Listing."

Error Checking: Source programs are examined by the assembler for possible errors arising from incorrect usage of the language. Wherever an error is detected, a coded warning message (called a flag) will be printed in the program listing. For card systems without printers, limited error notification is provided. See the topic "Error Notification."

Program Reassembly: A special reassembly procedure is provided for programs assembled by the IBM 1442-2 Card Read-Punch card-operating procedure. This will permit partially or completely assembled source programs, that have been modified, to be reassembled in less time than required for a new assembly. See the topic "Reassembly Procedure."

COMPATIBILITY WITH OTHER SYSTEM/360 ASSEMBLERS

Programs written in the Basic Assembler Language as described in this publication are acceptable to the other Basic Programming Support, Basic Operating System, and Operating System Assemblers, and the 7090/7094 Support Package Assembler. Similarly, source programs written in these other assembly languages are acceptable to the Basic Assembler if they do not embody any of the features of these assemblers which are unacceptable to the Basic Assembler. Appendix D contains a list of features supported by the System/360 Assemblers and may be used as a guide for the interchangeability of source programs.

The Basic Assembler will also accept programs written for the IBM System/360 Model 20 Basic Assembler, except where differences in machine design have made it necessary to include some instructions in the Model 20 Basic Assembler Language that are not contained in the Basic Assembler Language. These instructions are:

BAS BASR CIO HPR SPSW TIOB XIO
Y-type Expression Constants

Note also that the pseudo-registers zero through three on the Model 20 are handled differently from the corresponding actual registers on other models of the System/360.

MACHINE REQUIREMENTS

The assembler will operate on an IBM System/360 with the following minimum configuration:

8,192 bytes of storage
Standard Instruction Set
One IBM 1442 Model 2 or 1402 Card Read-Punch

The above configuration is for the card-operating procedure for the assembler, hereinafter called the card option.

If, in addition to the equipment required for the card option, IBM 2400-Series Magnetic Tape Units are available, the tape-operating procedure may be used. This procedure will be henceforth termed the tape option.

If an IBM 1443 Model 2 or 1403 Printer, or an IBM 1052 Printer-Keyboard is provided, the assembler will provide a program listing, complete with error messages, for each assembly. An option is available to list only those statements containing errors. For information concerning this option, refer to the topic "Program Listing."

CARD AND TAPE OPTIONS

The Basic Assembler is a two-phase program. The first phase produces data for use by the second phase. The intermediate data produced by phase 1 must be passed on to the second phase via some external storage medium. The storage mediums used are punched cards or magnetic tape. If punched cards are used for the intermediate data, the system is known as a "Card Option System." If tape is used, the system is termed a "Tape Option System." The machine configuration determines which option applies at a particular installation.

An assembler language source program consists of a sequence of source statements punched into cards, one statement per card. The card columns available for punching source statements vary with the machine configuration (that is, input device, card or tape option) and the programmer's discretion. See the following list.

Input Unit	Option	Columns Available
1402	tape	1-71 or 25-71
1402	card	1-47 (see note) or 25-71
1442	tape	1-71 or 25-71
1442	card	25-71

Note: Columns 1-71 may be used for the 1402 card option rather than only columns 1-47. The assembler scans all 71 columns of the statement field when obtaining the information required to generate the appropriate object code; however, only the contents of columns 1-47 are included in the program listing produced by the assembler.

In addition to a source statement, each card may contain an identification sequence number in columns 73-80.

In this section, the discussion of card formats assumes that all statements begin in column 1. When card column assignments differ because of statements beginning in column 25, the column numbers associated with the statements beginning in column 25 are placed in parentheses. Example: 1(25).

The statements may be written on the standard coding forms that IBM provides. Two forms are available: a "long" form, Form X28-6507 (Figure 1), and a "short" form, Form X28-6506, for IBM Card Read-Punch card-option assemblies (Figure 2).

Each line of the coding form is used to write a single statement and/or comments. The information on each line is punched into one card. If a card is completely blank, it will be ignored by the assembler. The position numbers shown in the forms correspond to the card columns.

Space is provided at the top of both coding forms to identify the program and give instructions to the keypunch operator. None of this information is punched into the statement cards.

STATEMENT FIELDS

An assembler statement is composed of one to four fields, from left to right: name field, operation field, operand field, and comments field. The identification-sequence field is not part of the statement. The statement fields can be written on the coding form in what basically is a free form. As a convenience, however, the name and operation fields are marked on the coding forms by heavy lines that indicate the maximum length of these fields. Programmers may wish to align the fields at these lines to create a neat and orderly appearance in the program listing.

Some general rules that must be observed when writing statements are:

1. The only required field in a statement is the operation field. The other fields are optional, depending on the operation and the programmer's wishes.
2. The fields in a statement must be in order and separated from one another by at least one blank.
3. The name, operation, and operand fields must not contain embedded blanks. A blank may, however, occur in the operand field as a character self-defining value or character constant.
4. Only one statement is allowed to a line; a statement cannot be continued on additional lines.
5. Column 72 must be blank.

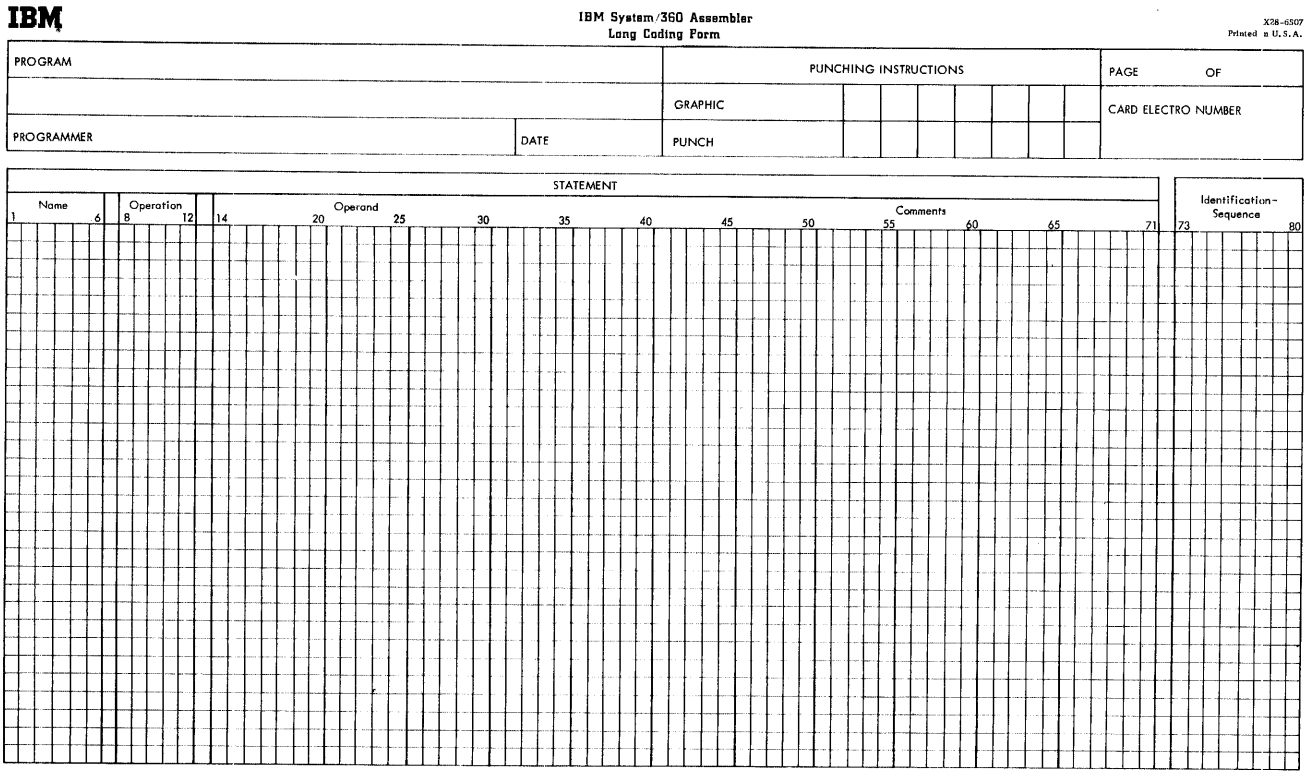


Figure 1. IBM System/360 Long Coding Form

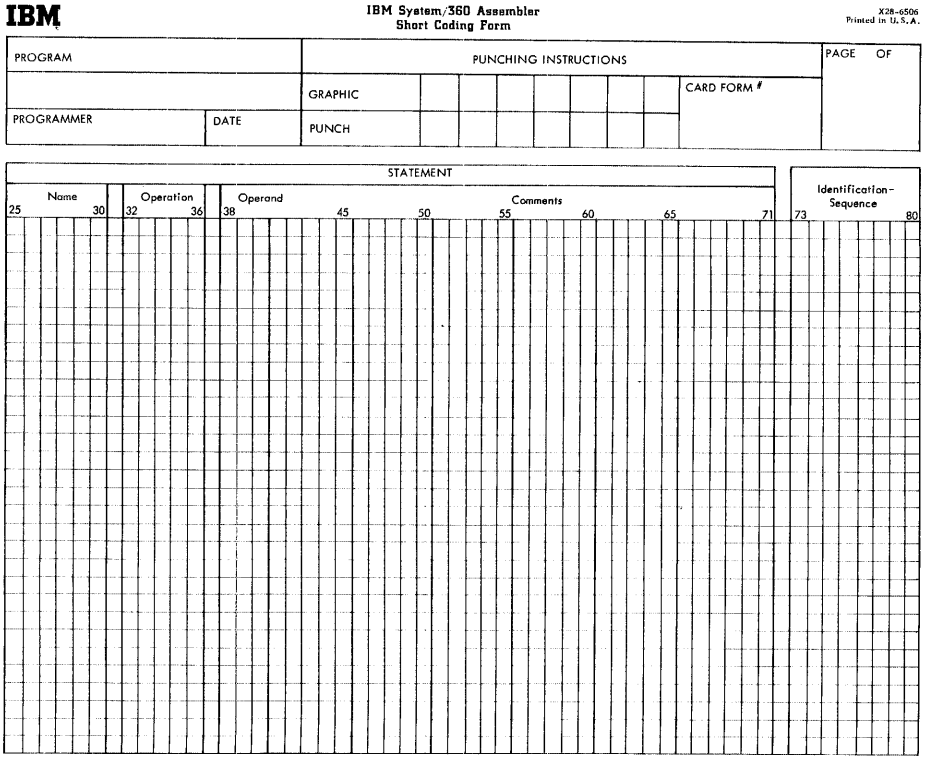


Figure 2. IBM System/360 Short Coding Form

Name Field

The name field is used to assign a symbolic name to a statement. A name enables other statements to refer to the statement by that name. If a name is given, it must begin in column 1 (25) and must not extend beyond column 6 (30). A name is always a symbol and must conform to the rules for symbols (see section, "Symbols"). Figure 3 shows the symbol FIELD2 used as a name.

If column 1 (25) is blank, the assembler will assume that the statement has no name. Column 1 (25) is also used to indicate that the card is a comments card (see discussion of the "Comments Field").

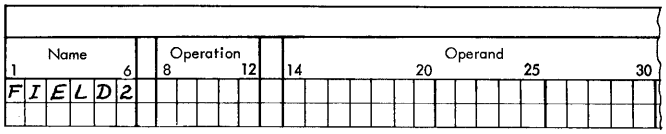


Figure 3. Example of the Name Field

Operation Field

The operation field is used to specify the mnemonic operation code of a machine or assembler instruction. This field may begin in any column to the right of column 1 (25) if the name field is blank. If the name field is not blank, at least one blank must separate the name and operation fields. The operation field may contain any valid mnemonic operation code. The valid machine-instruction mnemonics are listed in the section "Machine Instruction Statements;" the valid assembler-instruction mnemonics are listed in the section "Assembler Instructions." A valid mnemonic will never exceed five characters. If an invalid mnemonic is specified, the assembler will treat the statement as a comments statement and flag an error.

Figure 4 shows the mnemonic for the compare instruction (RR format) used in a statement named TEST. Note that this mnemonic could have been placed in columns 6-7, since this would have satisfied the requirement that at least one blank space separate the fields.

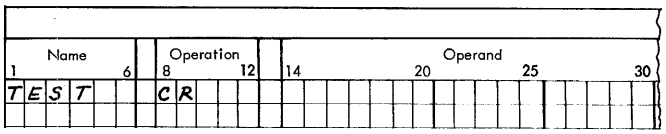


Figure 4. Example of the Operation Field

Operand Field

The contents of the operand field provide the assembler with additional information about the instruction specified in the operation field. If a machine instruction has been specified, the operand field contains information required by the assembler to generate the machine instruction. That is, the operand field specifies registers, storage addresses, input/output devices, immediate data, masks, and storage-area lengths. For an assembler instruction, the operand field conveys whatever information the assembler requires for the particular instruction.

The operand field may begin in any column to the right of the operation field, provided at least one blank space separates it from the last character of the mnemonic.

Certain assembler instructions do not require the operand field to be specified. If there is no operand field but there is a comments field, the absence of the operand field must be indicated by a comma, preceded and followed by one or more blanks. Figure 5 illustrates this rule.

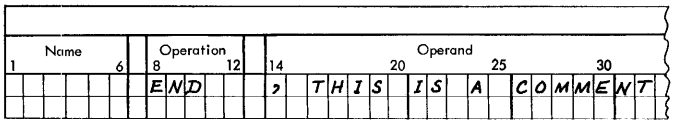


Figure 5. Example of No Operand Field with Comments

Depending on the instruction, the operand field may be composed of one or more subfields, called operands. Operands must be separated by commas. It must be remembered that a blank delimits the field; thus, a blank must not intervene between operands and commas. Figure 6 is an example of the same compare instruction shown in Figure 4, with its two operands specifying general registers 5 and 6. In Figure 6, as in Figure 4, the fields are separated by more than the minimum number of blank spaces.

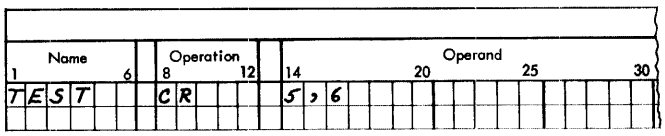


Figure 6. Example of the Operand Field

Comments Field

Comments are strictly for the convenience of the programmer. They permit lines or paragraphs of descriptive information about the program to be inserted into the program listing. Comments appear only in the program listing; they have no effect on the assembled object program. Any valid characters (including blanks) may be used as comments.

The comments field must (1) appear to the right of the operand field and (2) be preceded by at least one blank. If there is no operand field but there is a comments field, the absence of the operand field must be indicated by a comma, preceded and followed by one or more blanks. The entire statement field can be used for comments by placing an asterisk in column 1 (25); the entire statement will be treated as comments. Column 72, however, must remain blank.

If it is necessary to continue full-card comments on additional lines, each such line must have an asterisk in column 1 (25), as illustrated in Figure 7.

IDENTIFICATION-SEQUENCE FIELD

The identification-sequence field may be used for program identification and statement sequence numbers. This field can occupy columns 73-80 only. The information in this field normally is punched in every statement card. The assembler, however, will not check this field. It will merely reproduce the information in the field on the output listing of the program.

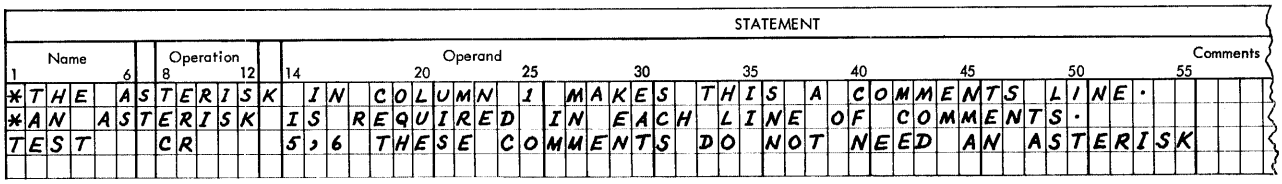


Figure 7. Example of the Comments Field

Language statements will be accepted by the assembler only if they conform to the established grammatical rules and vocabulary restrictions that are presented in this section. The reader can expect that many of the points not fully explained when they are first mentioned in this section will be described in detail subsequently.

CHARACTER SET

Basically, statements may be written using the following characters:

A through Z
0 through 9
* + - , ( ) ' . blank

The card column punch-combinations that the assembler will accept for these characters are listed below. This list also contains the punches assumed for additional printer graphics, which may be used in comments.

Character	Punch Combination
A - I	12 punch and a 1 - 9 punch, respectively
J - R	11 punch and a 1 - 9 punch, respectively
S - Z	0 (zero) punch and a 2 - 9 punch, respectively
0 - 9	0 (zero) - 9, respectively
blank	No punches
&	12
/	0-1
-	11
. (period)	12-3-8
$	11-3-8
,	0-3-8
#	3-8
<	12-4-8
*	11-4-8
%	0-4-8
@	4-8
(	12-5-8
)	11-5-8
' (single quotation)	5-8
+	12-6-8
=	6-8

SYMBOLS

Symbols are created by the programmer and used by him for symbolic referencing of storage areas, instructions, input/output units, and registers.

A symbol may contain from one to six characters; the characters may be any combination of alphabetic (A through Z) and numerical (0 through 9) characters. The first character must be alphabetic. Special characters and embedded blanks must not be used in symbols. Any violation of these rules will be noted with an error flag in the program listing and the symbol will not be used.

The following are valid symbols:

READER
A23456
LOOP2
N
S4

These symbols are invalid:

256B	First character is not alphabetic
AREATWO	More than six characters
RCD*34	Contains a special character

Defining Symbols

Symbols are meaningful in statements when used as operands and names. When a symbol is used as an operand, the assembler will normally assign certain attributes to it. These "attributes" are assigned to the symbol by the assembler when the symbol is defined. In order for a symbol to be used as an operand, it must be defined somewhere in the program.

A symbol is defined when the programmer uses it as the name of a statement. When the assembler finds a symbol in the name field, it will assign an address-value attribute and a length attribute to the symbol. The address value is the storage address of the leftmost byte of the field allotted to the statement; the length is the number of bytes in the storage field named by the symbol. This length is called the implied length associated with the symbol. The convenience of having implied

lengths will become apparent in the discussion of the symbolic format of machine instructions in the SS format.

A symbol defined in this manner is normally called a relocatable symbol. That is, the address value of the symbol will change if the program is loaded at a location other than its assembled location.

Symbols can be assigned arbitrary absolute values by use of the EQU assembler instruction. These values may designate registers, input/output units, immediate data, etc. They can also specify actual storage addresses such as permanently allocated interrupt locations. Symbols so defined are termed absolute symbols since their values are fixed and will not change because of program location.

Previously Defined Symbols

Sometimes the programmer will desire to give an alternate name to a previously defined symbol. "Previously defined" means that the symbol has appeared as the name of some statement prior to being used in the operand field of another statement. Figure 8 shows how the symbol TEST, defined in the first statement, is given an alternate name.

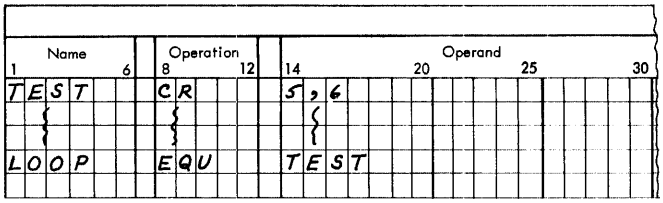


Figure 8. Example of Coding with Previously Defined Symbols

External and Entry-Point Symbols

Symbols are normally defined in the same program in which they are used as operands. It is possible, however, to define a symbol in one program, use it in another program assembled independently of the first, and then execute both programs together. Such a symbol is called an "external symbol" when it is used as an operand. The symbol is termed an "entry-point symbol" in the program in which it is defined. The address value of the entry-point symbol will be assigned to the external symbol when both programs are loaded by the relocating loader.

Before using an external symbol or defining an entry-point symbol, the programmer must indicate to the assembler which of the symbols are external and which are entry points. The ENTRY and EXTRN assembler instructions are provided for this purpose. Both instructions are described in the section "Assembler Instructions."

External symbols are always relocatable. They are subject to certain usage restrictions that are discussed at pertinent places elsewhere in this publication.

General Restrictions on Symbols

The following restrictions are in addition to those imposed elsewhere in the discussion of symbols:

1. A symbol may appear only once in a program as the name of a statement. If a symbol is used as a name more than once, only the first usage will be recognized. Each subsequent usage of the symbol as a name will be ignored and noted with an error flag in the program listing.
2. The number of symbols that may be defined in a program is restricted, depending on the machine's storage size. These restrictions are explained in detail in the section "The Symbol Table."
3. A symbol must always be defined as having a positive value not exceeding 65,535. Any symbol whose definition is contrary to this rule will not be used and the statement in which it appears will be flagged as an error.

THE LOCATION COUNTER

The assembler maintains a counter that it uses to assign consecutive storage addresses to program statements. This counter is called the Location Counter. It always points to the current address. After each machine instruction is processed, the Location Counter is incremented by the number of bytes assigned to that instruction. Certain assembler instructions also cause the Location Counter to be incremented, whereas others do not affect it.

The programmer can set and change the Location Counter by using the START and ORG assembler instructions described in the section "Assembler Instructions."

Location Counter Overflow: The maximum value of the Location Counter is 65,535, a 16-bit value. If a program being assembled causes the Location Counter to be incremented beyond 65,535, the assembler will retain only the rightmost 16 bits in the counter and continue the assembly, checking for any other source program errors. No object program will be produced. The assembler can, however, provide a listing of the entire source program. The statement causing the overflow will be flagged in the listing.

Program References: The programmer may refer to the current value of the Location Counter at any place in a program by using an asterisk as an operand. The asterisk represents the location of the first byte currently available. The use of an asterisk in a machine-instruction statement is the same as giving the statement a name and then using that name as an operand in the same statement. Note that the asterisk will have a different address value each time it is used. The asterisk will have the same length attribute that a symbol placed in the name field would have. An asterisk used as an operand is considered a relocatable symbol.

SELF-DEFINING VALUES

The ability to represent an absolute value symbolically is an advantage in cases where the value will be referred to repeatedly. However, it is equally necessary to have a convenient means of specifying an actual machine value or a bit configuration without having to go through the procedure of equating it to a symbol and using the symbol. The assembler language provides this facility through the self-defining value, which can be a decimal, hexadecimal, or character representation.

Self-defining values may be used to specify such program elements as immediate data, masks, registers, addresses, and address increments. The type of representation selected (decimal, hexadecimal, or character) will depend on what is being specified. The use of a self-defining value is quite distinct from the use of data constants specified by the DC assembler instruction and by literal operands. When a self-defining value is used in a machine-instruction statement, its value is assembled into the instruction. When a data constant is specified in a machine instruction, its address is assembled into the instruction.

Decimal

A decimal self-defining value is an unsigned number of from one to six decimal digits. A decimal self-defining value of more than six digits is not valid. The acceptable decimal digits are 0 through 9. Some examples are:

```
    7      4092      0007
  147       128    199860
```

The assembler imposes additional restrictions on decimal self-defining values, depending on their use. For example, a decimal self-defining value designating a general register should be from 0 through 15; one designating a core storage address should not exceed the size of available storage.

Hexadecimal

A hexadecimal self-defining value is an unsigned number of from one to six hexadecimal digits, enclosed in single quotation marks and preceded by the letter X. Hexadecimal self-defining values of more than six digits are not valid.

Each hexadecimal digit converts to a four-bit value. The hexadecimal digits, and their bit patterns are:

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

The following are examples of hexadecimal self-defining values:

```
X'25'     X'B'      X'12FA1E'
X'F4F'    X'00CD'   X'00E0'
```

A table for converting decimal to hexadecimal is provided in Appendix B.

Character

A character self-defining value is a single character, enclosed in single quotation marks and preceded by the letter C. A character self-defining value may be a blank or any combination of punches in a single card column that translates into the 8-bit IBM Extended BCD Interchange Code. There are 256 such combinations. Appendix A is a table of these combinations, their interchange codes, and, where applicable, their printer graphics. A single quotation

mark used as a character self-defining value, or an ampersand, is represented as two single quotation marks, or two ampersands, enclosed in single quotation marks, thus: C'''' or C'&&'

Examples of character self-defining values are:

```
C'/'        C'#'        C'.'
C'B'        C'2'        C' ' (blank)
```

The same value can frequently be represented by any one of the three types of self-defining values. Thus, the decimal self-defining value 196 can be expressed in hexadecimal as X'C4' and as a character, C'D'. The selection of a particular type of value is left to the programmer. Decimal self-defining values, for example, might be used for actual addresses or register and input/output unit numbers; hexadecimal self-defining values for masks; and character self-defining values for immediate data.

EXPRESSIONS

The term "expression" refers to symbols or self-defining values used as operands, either singly or in some arithmetic combination. Expressions are used to specify the various fields of machine instructions. They also are used as the operands of assembler instruction statements.

Expressions are classified as either simple or compound and either relocatable or absolute. Unless otherwise qualified, the term "expression" as used hereinafter implies any expression, simple or compound, absolute or relocatable.

A simple expression is a single unsigned symbol (including the asterisk used as the Location Counter value) or a single unsigned self-defining value used as an operand. The following are simple expressions:

```
FIELD2        2        C'R'
X'BF'         *        ALPHA
```

A compound expression is a combination of two or, at most, three simple expressions, connected to each other by arithmetic operators. The recognized operators are + (plus), - (minus), and * (asterisk), denoting, respectively, addition, subtraction, and multiplication. The following are compound expressions:

```
N+14*256        ENTRY-OVER
FIELD+X'2D'     *+GAMMA-200
```

Note that an asterisk is used above both for the Location Counter (*+GAMMA-200) and as an operator (N+14*256), but cannot be used in succession to denote the two in the same expression. The following example is invalid:

```
**5
```

A compound expression must not contain either two simple expressions or two operators in succession, nor may it begin with an operator. The following examples violate these rules and, therefore, are invalid:

```
AREAX'C'        -DELTA+256
FIELD+-10       +FIELD-10
```

Relative Addressing

Using compound expressions, the programmer can address instructions and data areas relative to the Location Counter or to some symbolic storage location. This is called relative addressing. In the sequence of instructions shown in Figure 9, the location of the CR instruction can be expressed as ALPHA+2 or BETA-4. Note that relative addressing is always in bytes, never in words or instructions. All of the mnemonics in Figure 9 are for two-byte instructions in the RR format.

Name (1–6)	Operation (8–12)	Operand (14–30)
ALPHA	LR	3,4
	CR	4,6
	BCR	1,14
BETA	AR	2,3
	BC	15,ALPHA+2

Figure 9. Example of Relative Addressing

Attributes of Expressions

The assembler separately evaluates each expression in the operand field. An expression is considered terminated by a comma, a left or right parenthesis, or a blank, depending on what the expression specifies (see section "Machine Instruction Statements"). The evaluation procedure is as follows:

1. Each simple expression is given its numerical value.

2. Moving from left to right, the arithmetic operations are performed, multiplication before addition and subtraction. Thus, A+B*C is evaluated as A+(B*C) and not (A+B)*C.
3. The arithmetic result becomes the value attribute of the expression.

In addition to computing the value attribute of an expression, the assembler also determines its length attribute. For a compound expression, the length attribute is the same as the implied length attribute of its leftmost simple expression. If the leftmost simple expression in an expression is a self-defining value or an asterisk, the implied length attribute of that expression is one byte.

Absolute and Relocatable Expressions

An expression is absolute if it (1) contains only absolute symbols or self-defining values, or (2) is of the following forms (where R is a relocatable symbol and A is an absolute symbol or self-defining value):

```
R-R       R-R+A     R-R-A     A+R-R
R-A-R     A-R+R     R+A-R
```

Although the address values of both relocatable symbols are subject to change when the program is loaded, the difference between their values will be constant; that is, absolute.

A relocatable expression is one whose value would change by N if the program was loaded N bytes away from its assembled location. Relocatable expressions must therefore conform to these rules:

1. A relocatable expression must contain either one or three relocatable symbols. If there are three relocatable symbols, one (and only one) must be preceded by the minus (-) operator. If only one relocatable symbol is present, it must not be preceded by the minus operator.
2. A relocatable symbol may not be multiplied. That is, it must not be preceded or followed by the asterisk (*) operator.

The following examples illustrate absolute and relocatable expressions. R represents relocatable symbols; A, absolute symbols.

Absolute Expressions:

```
R-R+5
A+14*C'H'
2048
A*A
```

Relocatable Expressions:

```
R+2
R-8*A
R-R+R
*-X'FB2'
R-A
```

The following expressions are invalid for the reasons listed:

R+R R+R-A	Contain two relocatable symbols.
R*A	Relocatable symbol is multiplied.
R+R+R	No minus operator.
A-R	Single relocatable symbol is preceded by a minus operator.
R-R-R	Two minus operators.

Restrictions

The following restrictions apply to all expressions. Additional limitations are imposed where pertinent in this publication.

1. An expression can have a negative value only when it is an absolute expression specifying an address constant using the DC assembler instruction.
2. An expression containing an external symbol may not contain any other relocatable symbols. For the purpose of evaluating such an expression, the value of the external symbol at assembly time will be zero; the symbol will be revalued when the program is loaded.
3. If an expression is used as the operand of a machine instruction statement, any self-defining values within it must not exceed 4,095. Instructions containing self-defining values exceeding 4,095 will be set to zero. The operation code will remain unchanged.
4. The maximum value of an expression is 65,535. If an expression exceeding this maximum is used in a machine instruction statement, the entire instruction except for the operation code will be set to zero; if used in

instruction except for the operation code will be set to zero; if used in an assembler instruction statement, the action taken depends on the instruction.

Note: The maximum value of each individual term in the operand field of USING, ORG, END, EQU, CCW (second operand), and DC (A) assembler instructions must not exceed 16,777,215. The maximum value of an entire expression in an operand field of a USING, ORG, END, or EQU instruction is, however, 65,535. The maximum value of an entire expression in the operand field of a DC (A) or CCW (second operand) instruction is 16,777,215.

The assembler language provides for the symbolic representation of all machine instructions. The symbolic format of these instructions varies with the machine format. There are five basic machine formats: RR, RX, RS, SI, and SS. Within each basic format, further variations are possible.

Machine instructions are automatically aligned by the assembler on half-word boundaries. Any byte skipped because of alignment will be set to zero. Such situations arise when data is inserted into the instruction string, as in a calling sequence.

Any machine instruction statement may be given a name, which other language statements can use. The value attribute of such a name is the address of the leftmost byte assigned to the assembled instruction. The length attribute of the name depends on the basic machine format:

Basic Machine Format	Implied Length (in Bytes)
RR	2
RX	4
RS	4
SI	4
SS	6

INSTRUCTION FORMAT

Figure 10 shows each basic machine format, followed by its corresponding symbolic operand field formats and mnemonic operation codes. The numbers in the basic machine formats are the bit sizes of the field.

Figure 11 identifies the field codes used in Figure 10. Figure 11 also contains other pertinent information for specifying the fields in machine instruction statements. The following are additional points that must be considered:

1. If no indexing is used in an RX instruction and the base register (B2) is present, the X2 field must be written as a zero. If indexing is used, and if the base register is implied, the base register field may be omitted.
2. If the field or fields enclosed in parentheses are omitted, the parentheses (and the comma between them) may also be omitted.
3. If the value of an absolute expression exceeds the maximum (stated in Figure 11) for a field, the entire instruction will be set to zero except for the operation code; the statement will be flagged in the program listing. The preceding does not apply to the displacement field.
4. If the value of a displacement field exceeds 4,095, only the rightmost 12 bits will be used; the listing will be flagged.
5. If the programmer writes an absolute expression specifying a displacement and does not specify a base register, the assembler will place zero in the base-register field. The same applies to the index register.
6. If any invalidity in the operand field (other than those listed above) prevents correct evaluation of an expression, the entire instruction, except for the operation code, will be set to zero and the statement will be flagged. Such invalidities would include undefined symbols, use of relocatable expressions where absolute expressions are called for, etc.

IMPLIED BASE REGISTERS AND DISPLACEMENTS

The assembler has the facility for assigning base registers and computing displacements for symbolic storage addresses. If this facility is used, the programmer simply specifies a symbolic address, by using a relocatable symbol, thus implying that the assembler is to select the base register and displacement. Before this can be done, however, the programmer must indicate to the assembler the contents and number of the general registers available for base registers. This information is conveyed with the USING and DROP instructions described in the section "Base Register Instructions."

Basic Machine Format								Assembler Operand Field Format	Applicable Instructions
Bits	8	4	4	4	12	4	12		
RR	Op Code	R1	R2	N/A	N/A	N/A	N/A	R1,R2	All RR instructions except BCR,SPM,SVC
	Op Code	M1	R2	N/A	N/A	N/A	N/A	M1,R2	BCR
	Op Code	R1	0	N/A	N/A	N/A	N/A	R1	SPM
	Op Code	I		N/A	N/A	N/A	N/A	I	SVC
RX	Op Code	R1	X2	B2	D2	N/A	N/A	R1,D2(X2,B2)	All RX instructions except BC
	Op Code	M1	X2	B2	D2	N/A	N/A	M1,D2(X2,B2)	BC
RS	Op Code	R1	R3	B2	D2	N/A	N/A	R1,R3,D2(B2)	BXH,BXLE,LM,STM
	Op Code	R1	R3	B2	D2	N/A	N/A	R1,D2(B2)	All shift instructions
SI	Op Code	I2		B1	D1	N/A	N/A	D1(B1),I2	All SI instructions except LPSW,SSM,HIO, SIO,TIO,TCH
	Op Code	0	0	B1	D1	N/A	N/A	D1(B1)	LPSW,SSM,HIO, SIO,TIO,TCH,TS
SS	Op Code	L1	L2	B1	D1	B2	D2	D1(L1,B1),D2(L2,B2)	PACK,UNPK,MVO, AP,CP,DP,MP,SP, ZAP
	Op Code	L		B1	D1	B2	D2	D1(L,B1),D2(B2)	NC,OC,XC,CLC, MVC,MVN,MVZ,TR, TRT,ED,EDMK

Figure 10. Machine Instruction Statement Formats

Base registers and displacements can be implied for RX, RS, SI, and SS instructions. For example, the operands of an RS instruction can be specified as

```
R1, R3, S2
```

where S2 represents a symbolic address (i.e., a relocatable symbol) that the assembler will separate into a displacement (D2) and base register (B2).

To specify addresses in this manner, the programmer must observe these rules:

1. The base register instructions (USING and DROP) must be used as described subsequently in this publication.
2. The symbolic address must be represented by a simple or compound relocatable expression.

Reference Summary for Operand Fields				
Field Code	Code Represents	Field Bit Size	Expression	
			Allowable Types	Maximum Values
R1,R2,R3	General or floating-point register	4	Simple absolute	15
M1	Mask	4	Simple absolute	15
D1,D2	Displacement	12	Simple or compound absolute	4095
B1,B2	Base register	4	Simple absolute	15
X2	Index register	4	Simple absolute	15
L1,L2	Length	4	Simple absolute	16*
L	Length	8	Simple absolute	256*
I2,I	Immediate	8	Simple absolute	255
* These are maximum values for length fields allowed in assembler statements; the values assembled for the instruction length fields are one less than these values.				

Figure 11. Operand Field Summary

3. A base register must not be written. An explicit base register will cause the assembler to treat the storage address as a displacement and an error will result because a displacement must always be an absolute expression. An explicit index register may be used, however, in the usual manner.

In the following example, the relocatable expression FIELD, with an address value of 7400, is used in a machine instruction; assume that the assembler has been told that general register 12 contains 4096 and is available as a base register.

```
ST    4,FIELD
```

The assembled machine instruction (in decimal) would be as follows, the value of D2 being the difference between 7400 and 4096.

Operation Code	R1	X2	B2	D2
50	4	0	12	3304

Had the instruction been ST 4,FIELD(2), the assembled machine instruction would differ from the previous example only in that the content of the X2 field would be 2 rather than zero.

IMPLIED AND EXPLICIT LENGTHS

The length field in *SS* instructions can be implied or explicit. An implied length is the length attribute of either the absolute expression specifying the displacement or the relocatable expression specifying the symbolic address, whichever

Reference Summary for Implied Operands				
Basic Machine Format	Explicit Base Registers and Displacement		Implied Base Registers and Displacement	
	Explicit Length (1)	Implied Length (2)	Explicit Length (3)	Implied Length (4)
RX	D2 (2,B2)	N/A	S2 (X2)	N/A
RS	D2 (B2)	N/A	S2	N/A
SI	D1 (B1)	N/A	S1	N/A
SS	D1 (L1,B1)	D1 (,B1)	S1 (L1)	S1
SS	D2 (L2,B2)	D2 (,B2)	S2 (L2)	S2
SS	D1 (L,B1)	D1 (,B1)	S1 (L)	S1
SS	D2 (B2)	N/A	S2	N/A
The S1 and S2 fields are relocatable expressions or absolute expressions representing values up to 4095; all other fields are absolute expressions. Where the S1 and S2 fields are absolute expressions, base register zero is implied.				

Figure 12. Implied Operand Field Summary

is written in the statement. The length attribute of a compound expression is the implied length of its leftmost simple expression.

An explicit length, by contrast, is written by the programmer in the statement as a simple absolute expression. If a length is explicit, it overrides the implied length associated with the displacement or symbolic address.

Regardless of how the length is specified (implied or explicit), if it exceeds the values indicated in Figure 11 for the L, L1, and L2 fields, the entire assembled instruction, except the operation code, will be set to zero.

Note that the length, whether implied or explicit, is always an effective length. That is, it is one more than the value inserted into the length field of the assembled machine instruction. In the case where an explicit length of zero is specified, the assembler assumes an effective length of one. Thus, a zero is inserted in the length field of the assembled instruction.

The reference summary in Figure 12 is for use with the figure showing the machine instruction formats (Figure 10). For each explicit operand format in column 1, any of the corresponding implied operand formats in columns 2, 3, or 4 can be substituted in order to specify an implied length or an implied base register and displacement, or both.

MACHINE INSTRUCTION MNEMONICS

This section contains an alphabetical listing of the mnemonics of all the machine instructions and their operand field formats. The column headings in the list are:

1. Mnemonic Code: This column contains the mnemonic operation code for the machine instruction.
2. Instruction: This column contains the name of the instruction associated with the mnemonic.
3. Operation Code: This column contains the hexadecimal equivalent of the actual machine operation code.
4. Basic Machine Format: This column contains the basic machine format of the instruction:
   RR, RS, RX, SI, or SS.
5. Operand Field Format: This column shows the explicit symbolic format of the operand and field for the particular mnemonic.

Appendix E provides a table for easy conversion of hexadecimal operation codes to their associated mnemonic codes.

Mnemonic Code	Instruction	Operation 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 Half-Word	4A	RX	R1,D2(X2,B2)
AL	Add Logical	5E	RX	R1,D2(X2,B2)
ALR	Add Logical	1E	RR	R1,R2
AP	Add Decimal	FA	SS	D1(L1,B1),D2(L2,B2)
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 Algebraic	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 Half-Word	49	RX	R1,D2(X2,B2)
CL	Compare Logical	55	RX	R1,D2(X2,B2)
CLC	Compare Logical	D5	SS	D1(L,B1),D2(B2)
CLI	Compare Logical Immediate	95	SI	D1(B1),I2
CLR	Compare Logical	15	RR	R1,R2
CP	Compare Decimal	F9	SS	D1(L1,B1),D2(L2,B2)
CR	Compare Algebraic	19	RR	R1,R2
CVB	Convert to Binary	4F	RX	R1,D2(X2,B2)
CVD	Convert to Decimal	4E	RX	R1,D2(X2,B2)
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
DP	Divide Decimal	FD	SS	D1(L1,B1),D2(L2,B2)
DR	Divide	1D	RR	R1,R2
ED	Edit	DE	SS	D1(L,B1),D2(B2)
EDMK	Edit and Mark	DF	SS	D1(L,B1),D2(B2)
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)

Mnemonic Code	Instruction	Operation Code	Basic Machine Format	Operand Field Format
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 Half-Word	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)
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 Half-Word	4C	RX	R1,D2(X2,B2)
MP	Multiply Decimal	FC	SS	D1(L1,B1),D2(L2,B2)
MR	Multiply	1C	RR	R1,R2
MVC	Move Characters	D2	SS	D1(L,B1),D2(B2)
MVI	Move Immediate	92	SI	D1(B1),I2
MVN	Move Numerics	D1	SS	D1(L,B1),D2(B2)
MVO	Move with Offset	F1	SS	D1(L1,B1),D2(L2,B2)
MVZ	Move Zones	D3	SS	D1(L,B1),D2(B2)
N	AND Logical	54	RX	R1,D2(X2,B2)
NC	AND Logical	D4	SS	D1(L,B1),D2(B2)
NI	AND Logical Immediate	94	SI	D1(B1),I2
NR	AND Logical	14	RR	R1,R2
O	OR Logical	56	RX	R1,D2(X2,B2)
OC	OR Logical	D6	SS	D1(L,B1),D2(B2)
OI	OR Logical Immediate	96	SI	D1(B1),I2
OR	OR Logical	16	RR	R1,R2
PACK	Pack	F2	SS	D1(L1,B1),D2(L2,B2)
RDD	Read Direct	85	SI	D1(B1),I2

Mnemonic Code	Instruction	Operation Code	Basic Machine Format	Operand Field Format
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 Half-Word	4B	RX	R1,D2(X2,B2)
SIO	Start I/O	9C	SI	D1(B1)
SL	Subtract Logical	5F	RX	R1,D2(X2,B2)
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
SP	Subtract Decimal	FB	SS	D1(L1,B1),D2(L2,B2)
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 Single 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 Half-Word	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
TR	Translate	DC	SS	D1(L,B1),D2(B2)
TRT	Translate and Test	DD	SS	D1(L,B1),D2(B2)
TS	Test and Set	93	SI	D1(B1)
UNPK	Unpack	F3	SS	D1(L1,B1),D2(L2,B2)
WRD	Write Direct	84	SI	D1(B1),I2
X	Exclusive OR	57	RX	R1,D2(X2,B2)
XC	Exclusive OR	D7	SS	D1(L,B1),D2(B2)
XI	Exclusive OR, Immediate	97	SI	D1(B1),I2
XR	Exclusive Logical OR	17	RR	R1,R2
ZAP	Zero and Add Decimal	F8	SS	D1(L1,B1),D2(L2,B2)

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 (shown in the following examples) requires the use of the USING assembler instruction described later in the publication.

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, which are categorized as absolute expressions. The operands of ALPHA2 and GAMMA2 are symbols that are equated elsewhere to absolute values.

RX Format

Name	Operation	Operand
ALPHA1	L	1,39(4,10)
ALPHA2	L	REG1,39(4,TEN)
BETA1	L	2,ZETA(4)
BETA2	L	REG2,ZETA(REG4)
GAMMA1	L	2,ZETA
GAMMA2	L	REG2,ZETA

Both ALPHA instructions specify explicit addresses; REG1 and TEN are absolute symbols. Both BETA instructions specify implicit addresses, and both use index registers. Indexing is omitted from the GAMMA instructions. GAMMA1 and GAMMA2 specify implicit addresses.

RS Format

Name	Operation	Operand
ALPHA1	BXH	1,2,20(14)
ALPHA2	BXH	REG1,REG2,20(REGE)
ALPHA3	BXH	REG1,REG2,ZETA
BETA1	SLL	1,20(9)
BETA2	SLL	REG1,20(9)
BETA3	SLL	REG1,ZETA

Whereas ALPHA1 and ALPHA2 specify explicit addresses, ALPHA3 specifies an implicit address. Similarly, the BETA instructions illustrate both explicit and implicit addresses.

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 specify explicit addresses, whereas the BETA instructions and GAMMA4 specify implicit addresses. GAMMA2 specifies a displacement of zero. GAMMA3 does not specify a base register.

SS Format

Name	Operation	Operand
ALPHA1	AP	40(9,8),30(6,7)
ALPHA2	AP	40(NINE,REG8),30(REG6,7)
ALPHA3	AP	FIELD2,FIELD1
ALPHA4	AP	FIELD2(9),FIELD1(6)
BETA	AP	FIELD2(9),FIELD1
GAMMA1	MVC	40(9,8),30(7)
GAMMA2	MVC	40(NINE,REG8),DEC(7)
GAMMA3	MVC	FIELD2,FIELD1
GAMMA4	MVC	FIELD2(9),FIELD1

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

Just as machine instructions are used to request the machine to perform a sequence of operations, so assembler instructions are requests to the assembler to perform certain operations. There are 15 such assembler instructions. Some already have been briefly mentioned in the preceding sections. All the assembler instructions are listed below by mnemonic operation code and name. They are fully described in the subsequent text. Figure 19 at the end of this section contains a summary description of all assembler instructions.

Assembler Control Instructions	
ICTL	Input Control
START	Start Program
ORG	Reset Location Counter
CNOP	Conditional No Operation
END	End Program
EJECT	Start New Page
SPACE	Space Listing

Definition Instructions	
EQU	Equate Symbol
DS	Define Storage
CCW	Define Channel Command Word
DC	Define Constant

Base Register Instructions	
USING	Use Base Address Register
DROP	Drop Register

Program Linking Instructions	
ENTRY	Identify Entry-Point Symbol
EXTRN	Identify External Symbol

Assembler instruction statements, in contrast to machine instruction statements, do not always cause actual machine instructions to be included in the object program. Some (for example, DS, DC) generate no instructions but cause storage areas to be set aside for constants and other data. Others (for example, EQU, SPACE) are effective only at assembly time; they generate nothing in the object program and have no effect on the Location Counter.

ASSEMBLER CONTROL INSTRUCTIONS

The assembler control instructions are used to specify the beginning and end of an assembly, set the Location Counter to a value or word boundary, control the program listing, and indicate the statement format. Except for the CNOP instruction, none of these assembler instructions generate instructions or constants in the object program.

ICTL - Input Control

The ICTL instruction tells the assembler in which card column the statement portion of the source-program cards begin. The mnemonic operation code of the ICTL statement must start in column 26 or higher. The format of the ICTL instruction statement is:

Name	Operation	Operand
Not used	ICTL	The decimal value 1 or 25

If the statements are to begin in column 25, the format is:

```
        ICTL    25
```

If the statements begin in column 1, the format is:

```
        ICTL    1
```

If (1) the ICTL statement is not used or (2) the operand field does not contain a 1 or 25, column 1 will be assumed for the tape option and column 25 will be assumed for the card option. When the ICTL statement is used, it must be the first statement in the source program. If it appears anywhere else, it will not be used. If a name is present, the name will not be used.

START - Start Program

The START instruction may be used to indicate the beginning of an assembly, to give a name to the program, and to set the Location Counter to an initial value. The format of the START instruction statement is:

Name	Operation	Operand
A symbol (optional)	START	A self-defining value or blank

The symbol in the name field becomes the name of the program. The symbol is assigned the address corresponding to the self-defining value in the operand field. This symbol can be specified as an external symbol (using the EXTRN instruction) in other programs, without using the ENTRY instruction to identify it as an entry point in this program. If there is no symbol in the name field, the assembler will assign a name consisting of six blanks.

A self-defining value that specifies the initial setting of the Location Counter is written in the operand field. If the value of the operand is not a multiple of eight, the Location Counter will be set at the next double-word boundary. The self-defining value must not exceed the maximum allowable setting of the Location Counter. If the operand field is invalid or blank, the Location Counter will be set to zero.

The initial setting of the Location Counter becomes the starting location of the program. This location is the initial loading location if the program is loaded by the absolute loader. It can also be used as the temporary starting location for loading the program while it is being tested. This enables the programmer to match the locations shown in the listing produced by the assembler with the locations in storage print listings. When the program has been checked out, it can then be relocated elsewhere by the relocating loader.

If both the START and ICTL instructions are used, the START instruction must immediately follow the ICTL instruction. If it appears anywhere else or if it is not used, the assembler will set the Location Counter to zero and give the program a name of six blanks. Any invalid occurrences of a START instruction will not be used. It should be noted that if the ICTL instruction is not used, the START instruction should be the first in the program.

Either of the START statements below could be used to assign the name PROG2 to the program and to set the Location Counter to a value of 2040:

```
PROG2    START   2040
PROG2    START   X'7F8'
```

ORG - Reset Location Counter

The ORG instruction resets the Location Counter to a relative value. This instruction may be used anywhere in the program, as often as desired. The format of the ORG instruction statement is:

Name	Operation	Operand
Not used	ORG	A relocatable expression

The Location Counter is reset to the value of the relocatable expression. An ORG instruction that resets the Location Counter below its initial value as specified in the START instruction will not be used; it will, however, be printed in the listing with an error flag. Any symbol(s) in the expression must be previously defined. If the operand field is blank or invalid, the ORG instruction will not be used. If a name is specified, the name will not be used.

The statement:

```
         ORG     *+500
```

increases the Location Counter by 500 above its current setting. Nothing is assembled for the 500 bytes skipped. That is, these bytes are not cleared by the assembler.

The ORG instruction provides an alternate way of reserving storage areas; the preferred way usually is with the DS (Define Storage) assembler instruction. However, where a storage area cannot be conveniently defined with the DS instruction, the ORG instruction can be used. For example, to reserve two storage areas of equal size, the following coding might be used:

```
TABLE1   DS      50F
         DS      100H
         .
         .
         .
TABLE2   EQU     *
         ORG     *+TABLE2-TABLE1
```

Note that the EQU assembler instruction permits TABLE2 to be used in the ORG statement as a previously defined symbol.

CNOP - Conditional No Operation

The CNOP instruction allows the programmer to align an instruction at a specific word boundary without breaking the instruction flow should any bytes be skipped for alignment. This facility is useful in creating calling sequences consisting of a linkage to a subroutine followed by parameters such as Channel Command Words (CCW) which require proper word boundaries.

The CNOP instruction aligns the Location Counter setting to a half-word, word, or double-word boundary. If the Location Counter is already aligned, the CNOP instruction has no effect. If the alignment specified requires the Location Counter to be incremented, a no-operation instruction (an RR branch-on-condition instruction with a zero R1 and R2 field) will be generated for each pair of bytes (half-words) skipped. If an odd number of bytes are skipped, the first byte will be set to zero.

The format of the CNOP instruction statement is:

Name	Operation	Operand
Not used	CNOP	Two decimal values of the form: b, w

Operand b specifies at which byte in a word or double-word the Location Counter is to be set; b can be 0, 2, 4, or 6. Operand w specifies whether the byte b is in a word (4) or double-word (8).

The following pairs of b and w values are valid:

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

Figure 13 shows the position in a double-word that each of these pairs specifies. Note that 0,4 and 2,4 specify two locations in a double-word.

If the operand field is blank or invalid, the CNOP instruction will not be used. A name, if present, will not be used.

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

```
CNOP    0,8
BALR    2,14
```

will have no effect; it will be printed in the program listing. This sequence, however:

```
CNOP    6,8
BALR    2,14
```

will cause three branch-on-condition instructions (no operations) to be generated, thus aligning the BALR instruction at the last half-word in a double-word:

```
BCR     0,0
BCR     0,0
BCR     0,0
BALR    2,14
```

After the BALR instruction is generated, the Location Counter will be at a double-word boundary so that a Channel Command Word (CCW) can be correctly positioned.

END - End Program

The END instruction terminates the assembly of a program. It may also supply a point in the program to which control is transferred after the program is loaded.

The END instruction must always be the last statement in the source program. When the assembler detects this statement, it produces a Load End card in the user's object program for use by the load program.

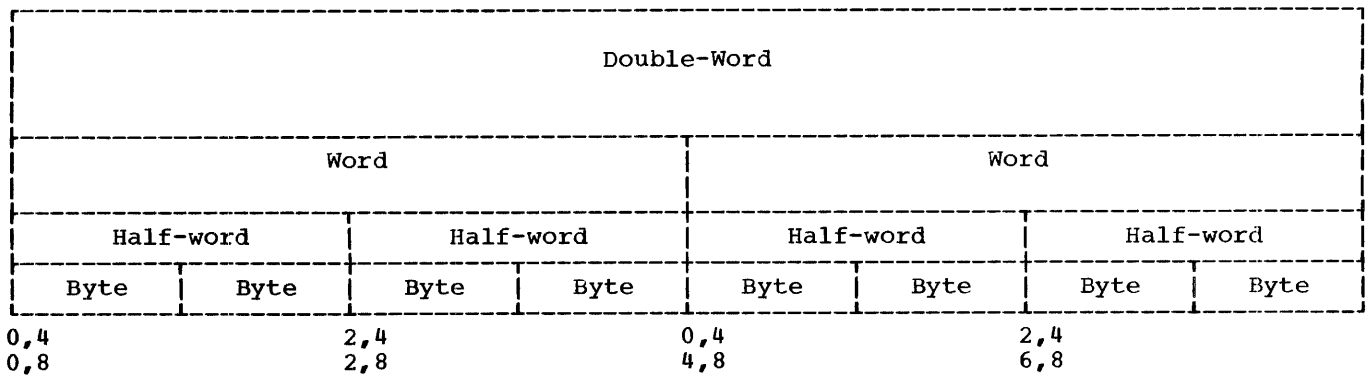


Figure 13. Boundary Alignment with a CNOP Instruction

The format of the END instruction statement is:

Name	Operation	Operand
Not used	END	A relocatable expression or blank

The expression in the operand field specifies the point to which control is transferred when loading is complete. The value of the expression will be punched in the Load End card. If the operand field is blank or invalid, nothing will be punched in the Load End card. In this case, control will be passed to the first storage location (above decimal location 128) occupied by the user's program when the program is loaded. If the operand field is invalid, the statement will be flagged as a possible error. If a name is present, it will not be used.

The point to which control usually is transferred is the first machine instruction in the program, as shown in this sequence:

```
          START   2000
AREA      DS      50F
BEGIN     SR      3,3
          .
          .
          .
          END     BEGIN
```

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:

Name	Operation	Operand
Not used	EJECT	Not used

Normally, the EJECT statement is not included in the program listing; however, anything appearing in the name or operand fields will result in including the statement in the listing. In this case, the EJECT statement is printed prior to skipping to the new 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:

Name	Operation	Operand
Not used	SPACE	A decimal value

A decimal value is used to specify the number of blank lines to be inserted in the program listing. If this value exceeds the number of lines remaining on the listing page, the statement will have the same effect as an EJECT statement. A blank operand field will cause one line to be

skipped. Normally, the SPACE statement is not included in the program listing. There are, however, some exceptions. Anything in the name field of a SPACE statement results in including the statement in the listing. In this case, the statement is printed prior to spacing. If the operand field is invalid (that is, not a decimal value or one greater than 4095), the statement is flagged and listed. No space operation occurs.

DEFINITION INSTRUCTIONS

The definition assembler instructions are used to define and enter constant data into a program, specify the contents of Channel Command Words, and reserve areas of core storage. The fields generated by these instructions can be referred to by symbolic names. The EQU instruction is included with the definition instructions because it is used for defining symbols.

EQU - Equate Symbol

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

Name	Operation	Operand
A symbol	EQU	An expression

The symbol in the name field is given the same value attribute as the expression. The length attribute of the symbol will be that of the leftmost term of the expression. If the term is an asterisk (the Location Counter) or a self-defining value, the implied length of the symbol is one. The expression in the operand field can be relocatable or absolute, and the symbol will be similarly defined. Any symbols in the expression must be previously defined and have a positive value. Symbols not conforming to these rules will not be used. The associated EQU statements will be flagged.

If the expression in the operand field or the symbol in the name field, or both, are invalid or not present, the EQU statement will be flagged in the listing and will not be used.

The EQU instruction is the usual way of equating symbols to register numbers, input/output unit numbers, immediate data, actual addresses, and other arbitrary values. The examples below illustrate how this might be done:

```
REG2    EQU     2       General register
IO125   EQU     125     Input/output unit
TEST    EQU     X'3F'   Immediate data
TIMER   EQU     80      Actual address
```

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

```
FIELD   EQU     ALPHA-BETA+GAMMA
```

FIELD will be 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.

DS - Define Storage

The DS instruction is used to reserve storage areas and to assign names to the areas. This instruction is the preferred way of symbolically defining storage for work areas, input/output areas, etc. The format of the DS instruction statement is:

Name	Operation	Operand
A symbol (optional)	DS	An operand describing the area to be reserved, in the form shown below

The single operand specifies the number, type, and, if desired, the length of the fields to be reserved. The general form of the operand is:

dtLn

Where:

d
: is a decimal self-defining value that specifies the number of fields (from zero to 65,535) to be reserved. It is called the duplication factor. If it is omitted, one field will be reserved.

t
: is the type code specifying the type of field to be reserved and can be one of the following letters:

Code	Field Type	Implied Length (in Bytes)
C	Character (byte)	1
H	Half-word	2
F	Full-word	4
D	Double-word	8

Ln

can be used only if the field code is C. Ln is the length code written as the letter L immediately followed by n, which is the length (in bytes) of each field. n can be a decimal value that is not zero or greater than 256.

Half-word, full-word, and double-word fields will be aligned to their proper boundaries. With a duplication factor (d) of zero, the DS instruction can be used to cause boundary alignment. Thus, the statement:

```
        DS      0D
```

will set the Location Counter at the next double-word boundary.

If there is a symbol in the name field, it will be assigned the current value of the Location Counter after any alignment. The length attribute of the symbol will be the implied length associated with the field code, unless a length code (Ln) is specified, in which case the length attribute will be the same as the length n.

For example, to define four 10-byte fields and one 100-byte field, the respective DS statements might be:

```
FIELD   DS      4CL10
AREA    DS      CL100
```

Then, to move the first 10 bytes at AREA into FIELD, the coding is as follows, assuming implied base registers and displacements:

```
        MVC     FIELD,AREA
```

Note that the length attribute of FIELD, which is 10, is implied. Explicit length specification can be used to move the first 20 bytes at AREA into FIELD. The following instruction illustrates this:

```
        MVC     FIELD(20),AREA
```

Additional examples of DS statements are shown below. The implied length attribute of each symbol appears in parentheses before the symbol:

```
(80)  DONE    DS    CL80    One 80-byte field
(1)   DTWO    DS    80C     80 one-byte fields
(4)   DTHREE  DS    6F      Six full-words
(8)   DFOUR   DS    D       One double-word
(2)   DFIVE   DS    4H      Four half-words
```

If the operand is incorrectly specified, the statement will not be used. An error flag will appear in the listing.

A DS statement causes the reserved area to be skipped but not cleared. Therefore, the programmer should not assume that the area contains all zeros when the program is loaded. Whenever the assembler processes a DS statement, it will terminate the current output card (called a Text card) in the object deck and start the next card at the location following the reserved areas, thus skipping them. To minimize the number of Text cards punched, DS statements should be kept together as much as possible. Note, however, that Text cards are not terminated if no bytes are skipped by DS statements used only for boundary alignment.

CCW - Define Channel Command Word

The CCW instruction provides a convenient way to define and generate an eight-byte Channel Command Word aligned at a double-word boundary. The internal machine format of a Channel Command Word is shown in Figure 14. The format of a CCW instruction statement is:

Name	Operation	Operand
A symbol (optional)	CCW	Four operands, separated by commas, specifying the contents of the Channel Command Word in the form described below

The four operands, from left to right, are:

1. A simple absolute expression specifying the command code. The value of this expression will be right-justified in byte 1.
2. A relocatable expression specifying the data address. The value of this expression will be right-justified in bytes 2-4.
3. A simple absolute expression specifying the flags in bits 32-36 and zeros in bits 37-39. The value of

this expression is right-justified in byte 5. Byte 6 is set automatically to all zeros.

4. A simple absolute expression specifying the count. The value of this expression is right-justified in bytes 7-8.

The following is an example of a CCW statement:

```
CCW      X'OF',READIN,X'A8',80
```

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

```
32      36       40       44
 1010    1000     0000     0000
```

No operand field may be omitted. Operands not used must be written as zeros. An error in the operand field will cause eight bytes of zeros, aligned at a double-word boundary, to be assembled.

If there is a symbol in the name field, it will be assigned the value of the leftmost byte of the Channel Command Word after any boundary alignment. The length attribute of the symbol will be eight. Bytes skipped because of alignment will be assembled as zeros.

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	Assembled automatically as all zeros
7-8	48-63	Count

Figure 14. Channel Command Word

DC - Define Constant

The DC instruction is for generating constant data in main storage. Data can be specified as characters, hexadecimal numbers, decimal numbers, and storage addresses. Decimal numbers may be in the form suitable for both fixed-point and floating-point arithmetic operations. The format of the DC instruction statement is:

Name	Operation	Operand
A symbol (optional)	DC	A single operand describing the constant, written in the form shown below

The operand specifies the type of constant and the constant itself. It may also specify an explicit storage length for the constant and indicate how many times the constant is to be duplicated in storage. The format of this operand varies with the constant type. The basic format is either

dtLn'c' or ALn(c)

where:

d
is a decimal self-defining value (from 1 to 65,535) that specifies the number of identical constants to be generated. It is called the duplication factor. If it is omitted, one constant will be produced. A duplication factor cannot be specified for an expression (type A) constant.

Note: A print line is produced for each constant generated. Thus, assembler speed can be increased by keeping duplication factors small and length codes large.

t
is the type code, specifying the type of constant. It can be one of the following letters:

Code	Constant Type	Machine Format
C	character	8-bit BCD code.
X	hexadecimal	Fixed-point binary.
F	decimal	Full-word fixed-point binary.
E	decimal	Short-precision floating-point binary.
H	decimal	Half-word fixed-point binary.
D	decimal	Long-precision floating-point binary.
A	relocatable or absolute expression	Fixed-point binary.

Ln
is the length code written as the letter L followed by n, a decimal value, which is the explicit length (in bytes) of the constant. A length code is not applicable with constant types H, E, and D. If a length code is not given, the implied lengths shown in Figure 15 will be used. An explicit length must not exceed those values shown in Figure 15.

'c'
is the constant itself enclosed in single quotation marks. Note that for constant type A, the expression specifying the constant is enclosed in parentheses (c).

If the operand is invalid, the statement will not be used and will be flagged in the listing.

All constant types except character (C) and hexadecimal (X) will be aligned at appropriate boundaries. Constants are not aligned if an explicit length is given. The boundaries for the various constant types are summarized in Figure 15. Any bytes skipped for alignment will be set to zero.

A symbol in the name field will be given the address value of the first byte assigned after any alignment. The length attribute of the symbol will be the implied (or explicit) length of the constant before the duplication factor is applied.

The implied or explicit length of a constant defined by a single DC statement must not exceed 16 bytes before the duplication factor is applied. If longer constants are required, successive DC statements should be used. The total storage alloted to a constant defined by one DC statement will be the duplication factor times the length of the constant.

The subsequent text, with examples, describes each of the constant types. This material is summarized in Figure 15. Note that the definition of character, hexadecimal, and decimal constants is not limited by the rules pertaining to self-defining values.

Character Constants (C)

A character constant may comprise not more than 16 valid characters. A valid character is a blank or any combination of punches in a card column that translates into the 8-bit IBM Extended BCD Interchange Code. There are 256 such combinations; the table in Appendix A lists the combinations, their eight-bit codes, and, where applicable, their printer graphics.

Each character in the constant is translated into one byte. Boundary alignment is not performed. The number of bytes required for the constant becomes its implied length unless an explicit length is stated. In the following example, the length attribute of FIELD is 12:

```
FIELD    DC       C'TOTAL IS 10'
```

A single quotation mark used as a character is represented in the constant by two single quotation marks. The same rule applies to ampersands. Thus:

```
         DC       C'DON''T'
         DC       C'A,B&&C'
```

Five bytes will be used for each constant.

If more than 16 characters are specified or a length code is given and the size of the constant exceeds the explicit length, the excess rightmost characters will be truncated before applying the duplication factor. The statement will be flagged. For example, the statement:

```
         DC       3CL4'ABCDE'
```

will generate:

```
ABCDABCDABCD
```

If the number of characters is fewer than the explicit length, the constant will be padded by adding the necessary right-hand blanks. The statement:

```
         DC       4CL3'NO'
```

will generate in storage:

```
NObNObNObNOb
```

Reference Summary for DC Statements					
Constant-type Code	Boundary Alignment (If length is implied)	Length (in Bytes)		Duplication Allowed	Truncation/ Padding Side
		Implied	Maximum Explicit		
C	none	variable*	16	yes	right
X	none	variable*	16	yes	left
F	word	4	4	yes	left
H	half-word	2	invalid	yes	left
E	word	4	invalid	yes	none
D	double-word	8	invalid	yes	none
A	word	4	4	no	left
* But not exceeding 16 bytes					

Figure 15. DC Statement Summary

Hexadecimal Constants (X)

A hexadecimal constant may comprise up to 32 hexadecimal digits. The valid hexadecimal digits are:

```
0 1 2 3 4 5 6 7 8 9 A B C D E F
```

A table for converting hexadecimal to decimal is included in Appendix B. The reader also is referred to the section "Self-Defining Values." Each hexadecimal digit represents four bits; hence, every pair of digits will be translated into one byte. Boundary alignment will not be performed. If an odd number of hexadecimal digits is present, the four leftmost bits of the leftmost byte will be set to zero. Unless an explicit length is specified, the number of bytes required for the constant will become its implied length.

An eight-digit hexadecimal constant provides a convenient way to set the bit pattern of a full binary word. The constant in the following example would set the first and third bytes of a word to ones. Note that the preceding DS statement is used to align the constant at a full-word boundary:

```
          DS      0F
TEST      DC      X'FF00FF00'
```

If (1) more than 32 hexadecimal digits are present or (2) a length code is specified and the byte size of the constant exceeds the explicit length, the excess leftmost digits will be truncated before the duplication factor is applied. The statement will be flagged in the listing. In the following statement, the A will be truncated and 6F4E will be used as the constant:

```
ALPHA     DC      3XL2'A6F4E'
```

The resulting constant will be generated three times:

```
6F4E6F4E6F4E
```

If the pairs of digits are fewer than the explicit length, the constant will be padded by adding zeros to the left before applying the duplication factor. Thus:

```
DC        2XL3'2DDA'
```

will generate two 3-byte constants:

```
002DDA002DDA
```

Full-Word Constants (F)

The signed decimal constant in the operand is converted into a binary number. An unsigned number will be assumed to be positive. Negative numbers will be converted to two's complement notation.

If there is no explicit length, the binary number is placed in a full-word aligned at the proper boundary. An implied length of four is assigned. If a length code is present, alignment will not occur; the binary number will be right-justified in the specified number of bytes. An explicit length must not exceed four bytes.

Given the following statement:

```
CONWRD    DC      3F'+658474'
```

three full-word positive constants will be produced. The address value of CONWRD will correspond to the leftmost byte of the first word; the length attribute will be

four. Thus, the expression CONWRD+4 can be used to address the second word symbolically.

The maximum permissible value of a full-word constant depends on the length, as follows:

Length	Highest Value	Lowest Value
4	2,147,483,647	-2,147,483,648
3	8,388,607	-8,388,608
2	32,767	-32,768
1	127	-128

Note: All lengths can be explicit. A length of 4, however, can also be implied.

If a value exceeds the limits associated with the length, a constant of zero will be generated before applying the duplication factor. The statement will be flagged in the listing. For example, the following statement would generate 12 bytes of zeros:

```
DC      4FL3'-9500250'
```

Half-Word Constants (H)

The signed decimal constant in the operand is converted into a binary number placed in a properly aligned half-word. A length code is not allowed. The implied length of the constant is two bytes.

If the number is unsigned, a positive value is assumed. Negative numbers will be converted to two's complement notation.

The largest number permitted is 32,767; the smallest is -32,768. If a number exceeds these limits, the constant will be set to zero before the duplication factor is applied. The statement will be flagged.

The following statement will generate two identical half-word positive constants, right-justified within two bytes:

```
DC      2H'256'
```

Short-Precision Floating-point Constants (E)

A short-precision floating-point constant is specified as a decimal fraction (mantissa) and an optional decimal exponent. The maximum allowable range for a floating-point constant is from approximately $(5.4) \times 10^{-79}$ to $(7.2) \times 10^{75}$. The constant will be aligned at a full-word boundary in the proper machine format for use in floating-point operations. A duplication factor may be applied to the constant. A length code, however, may not be used.

The format of the constant portion of the operand is described in the following text.

Fraction: The fraction is a signed decimal number (up to 8 digits) with or without a decimal point. The decimal point can appear before, within, or after the number. If the point is at the right-hand end of the number, it may be omitted. If the sign is omitted, a positive fraction is assumed. A negative fraction is carried in the machine in true form. The fraction, irrespective of its decimal point or sign, must not exceed 2 to the 24th power minus 1 (i.e., 16,777,215). The fraction part of a number converted to the short format will differ by no more than 1 from the exact value rounded to 24 places.

Exponent: The exponent is optional. It may be omitted if the decimal point appears in the fraction at the desired position. If the exponent is specified, it must immediately follow the fraction. It consists of the letter E followed by a signed decimal number denoting the exponent to the base ten. A positive exponent is assumed if the sign is omitted.

A negative exponent indicates that the true decimal point is to the left of the point written (or assumed) in the fraction. A positive exponent indicates that the true decimal point is to the right. The value of the exponent determines how many places to the left or right the true decimal point is located.

For example, to convert the number 46.415 to a floating-point format, any of the following statements could be used. They will all have the same effect:

```
DC      E'46.415'
DC      E'46415E-3'
DC      E'+46415.E-3'
DC      E'.46415E2'
DC      E'4.6415E+1'
```

If either the fraction or the exponent is outside the permissible range, the full word (or words, if a duplication factor is specified) will be set to zero and a flag will appear in the listing. The statement:

```
DC      4E'3.45E76'
```

would generate four full-words of zeros.

Long-Precision Floating-Point Constants (D)

A long-precision floating-point constant is specified as a decimal fraction (mantissa) and an optional decimal exponent. The maximum allowable range for a floating-point constant is from approximately $(5.4) \times 10^{-79}$ to $(7.2) \times 10^{75}$. The constant will be aligned at a double-word boundary in the proper machine format for use in floating-point operations. A duplication factor may be applied to the constant. A length code, however, may not be used.

The format of the constant portion of the operand is described in the following text.

Fraction: The fraction is a signed decimal number (up to 17 digits) with or without a decimal point. The decimal point can appear before, within, or after the number. If the point is the right-hand end of the number, it may be omitted. If the sign is omitted, a positive fraction is assumed. A negative fraction is carried in the machine in true form. The fraction, irrespective of its decimal point or sign must not exceed 2 to the 56th power minus 1 (that is, 72,057,594,037,927,935). The fraction part of a number converted to the long format will differ by no more than 11 from the exact value rounded to 56 places.

Exponent: The exponent is optional. It may be omitted if the decimal point appears in the fraction at the desired position. If the exponent is specified, it must immediately follow the fraction. It consists of the letter E followed by a signed decimal number denoting the exponent to the base ten. A positive exponent is assumed if the sign is omitted.

A negative exponent indicates that the true decimal point is to the left of the point written (or assumed) in the fraction. A positive exponent indicates that the true decimal point is to the right. The value of the exponent determines how many places to the left or right the true point is located.

If either the fraction or exponent is outside the permissible range, the double word (or words, if a duplication factor is specified) will be set to zero. The statement will be flagged.

The following statements illustrate different ways of converting the same number to a long-precision floating-point number:

```
DC      D'-72957'
DC      D'-729.57E+2'
DC      D'-729.57E2'
DC      D'-.72957E5'
DC      D'-7295700.E-2'
```

Expression Constants (A)

An expression constant consists of a relocatable or absolute expression enclosed in parentheses instead of single quotation marks. The value of the expression is generated as a 32-bit value constant. Since the expression frequently represents a storage address, the constant generated from it is commonly called an address constant. Hence, the letter A is used as the type code. Note that if the program is relocated, all address constants generated from relocatable expressions will be changed by the relocating program loader.

An explicit length not exceeding four bytes may be specified for expression constants. However, a duplication factor is not allowed.

Unless a length code is present, the 32-bit constant will be aligned at a full-word boundary and given an implied length of four. Thus, in the following statement, the value of AREA+2, as a 32-bit value, will be placed in the next available full word. ADCON1 will be given a length attribute of four:

```
ADCON1   DC      A(AREA+2)
```

If a length code is given, the constant will not be aligned. The constant will be right-justified in the specified number of bytes. Any excess bits to the left will be truncated. For example, in the statement:

```
ADCON2   DC      AL2(FIELD-256)
```

the rightmost 16 bits of the value of FIELD-256 will be right-justified in the next two bytes. The length attribute of ADCON2 will be two. In this case, FIELD must be equivalent to an absolute symbol. (see below.)

The following considerations govern type A constants:

1. A relocatable expression may be used only if the length is implied (that is, it is four) or if the explicit length is three or four.
2. An expression may have a negative value only if it is an absolute expression. A negative value will be stored in two's complement notation.
3. An expression may not begin with an arithmetic operator.

BASE REGISTER INSTRUCTIONS

The USING and DROP base register assembler instructions enable programmers to use expressions representing core storage locations as operands of machine instruction statements, leaving the assignment of base registers and the calculation of displacements to the assembler.

This feature of the assembler, besides simplifying programming, also 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 must use the USING and DROP instructions described in this section.

USING - Use Base Address Register

The USING instruction indicates that the general register specified in the operand is available for use as a base register. This instruction also states the base-address value that the assembler must assume is in the register at object time. Note that a USING instruction does not load the register specified. It is the programmer's responsibility to see that the specified base-address value is placed into the register. Suggested loading methods are described in the section "Programming with the USING and DROP Instructions." The format of the USING instruction statement is:

Name	Operation	Operand
Not used	USING	A relocatable expression and a simple absolute expression, separated by a comma

The relocatable expression specifies a value that the assembler can use as a base address. The second operand is a simple absolute expression specifying the general register that can be assumed to contain the base address represented by the first operand. The value of the second operand must be from 1 to 15. For example, the statement:

```
USING    *,12
```

tells the assembler it may assume that the current value of the Location Counter will be in general register 12 at object time.

If the programmer changes the value in a base register currently being used, the assembler must be told the new value by means of another USING statement. In the following sequence, ALPHA is a relocatable expression:

```
USING    ALPHA,9
         .
         .
         .
USING    ALPHA+1000,9
```

The assembler will first assume that the value of ALPHA is in register 9. The second statement causes the assembler to assume ALPHA+1000 as the value in register 9.

If the value of the second operand is zero, implying no base addressing, the first operand should also have a value of zero. If it does not, zero will be used instead of the actual value and the statement will be flagged in the listing. The implications of using register zero as a base register are discussed later in "Base Register Zero."

A USING statement will not be used if either of its operands are incorrect. A flag will appear in the listing. Any symbol in the name field will not be used.

DROP - Drop Register

The DROP instruction specifies a previously available register that may no longer be used as a base register.

Name	Operation	Operand
Not used	DROP	A simple absolute expression

The expression indicates a general register that previously had been named in a USING statement and now is unavailable for base addressing. The following statement, for example, removes register 11 from the list of available registers:

```
DROP     11
```

The DROP statement is ignored if the register it designates had never appeared in a USING statement. If the value of the expression exceeds 15, the statement will not be used and will be flagged in the listing. Any symbol in the name field will not be used.

It is not necessary to use a DROP statement when the base address in a register changes as a result of 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 restored to the list of available registers by a subsequent USING instruction.

Programming with the USING and DROP Instructions

The USING and DROP instructions may be used anywhere in a program, as often as needed. They provide the assembler with the information it needs to construct a "register table." Entries in the table are added, deleted, and changed by the assembler as each USING and DROP instruction is processed.

Whenever an effective address is specified in a machine instruction statement, the assembler consults this table to determine whether there is an available register containing a suitable base address. If more than one register will produce a valid displacement (that is, a displacement not exceeding 4095), the register whose contents produce the smallest displacement will be used. If two or more registers will produce the same displacement, the highest numbered register will be used. If there is no register that will produce a valid displacement, the corresponding base register and displacement fields will be set to zero; the statement will be flagged.

The sequence of instructions in Figure 16 illustrates the assignment of base registers. Instructions that load the registers are not shown.

```
      0000    PGMNME    START   0
                        USING   *,11
                        USING   *+4096,12
                        USING   *+8192,13
                        USING   *+4500,14
                          .
                          .
                          .
      2000    ALPHA     MR      1,2
                          .
                          .
                          .
      5500    BETA      SR      1,2
                          .
                          .
                          .
              B1        BC      15,ALPHA
              B2        BC      15,BETA
              B3        BC      15,GAMMA
                          .
                          .
                          .
      9750    GAMMA     AR      1,2
                        DROP    11

B1--Although the effective address represented by ALPHA can be wholly
    contained in the displacement field without a base address, base register 11 is nonetheless assigned since to use base register 0 would make the program nonrelocatable (see below). Because the value in register 11 is zero, the displacement will be 2000.

B2--Either register 12 or 14 would produce valid displacements; register 14 is used, however, because it produces the smaller displacement, which is 1000.

B3--Only register 13 can be used as the base register; the calculated displacement is 1558.
```

Figure 16. Example of Coding with USING and DROP Instructions

LOADING REGISTERS

Several methods exist for loading general registers that will be used for base addressing. However, for a program to be relocated when it is loaded, at least one of the base registers must be loaded with a relocatable address, using either of the instructions described below. The exact method of using these instructions can differ from the examples shown.

Branch and Link (BALR or BAL) Instruction

In the sequence below, the BALR instruction loads into register 5 the address of the first storage location after the BALR instruction. The USING instruction indicates to the assembler that register 5 contains this location:

```
                  BALR    5,0
          USING   *,5
```

When using this method, the USING instruction must immediately follow the BALR instruction.

Load Full-Word (L) Instruction

In the following coding, the value of RGLOAD is generated as a constant. RGLOAD is some symbol defined elsewhere in the program. This value, which is also specified in the USING instruction, is inserted into register 6 with the Load (L) instruction.

```
CNSTNT   DC      A(RGLOAD)
         .
         .
         .
         L       6,CNSTNT
         .
         .
         USING   RGLOAD,6
```

Note that if the symbol RGLOAD was used in the load instruction, register 6 would contain the full-word located at RGLOAD rather than the value of RGLOAD itself.

The Load instruction should precede the USING instruction to insure that the assumed contents of the register are, in fact, in the register when the program is executed. Otherwise, the assembler would use the specified register as a base register in machine instructions before the load instruction was encountered. This could lead to undesirable results when the program is executed. Observe, however, that the USING instruction need not immediately follow the load instruction, although it is recommended that the two instructions be consecutive.

If one register has been initialized by the Branch-and-Link or Load instruction, other registers may be loaded from it by other instructions. Thus, in the following example, the Load Address (LA) instruction causes 4,080 to be added to the contents of register 4 and the resulting total to be placed in register 3:

```
         BALR    4,0
         USING   HERE,4
HERE     LA      3,4080(0,4)
           .
           .
           .
         USING   HERE+4080,3
```

Note that the LA instruction could have been written alternately as LA 3,4080(4).

Base Register Zero

The specification of general register 0 as a base register indicates that a quantity of zero is to be used as the base address, regardless of the contents of general register 0. Therefore, if general register 0 is made available by a USING instruction for base addressing, the program will not be relocatable when there is no other general register available for referencing locations below location 4096. Figure 17 illustrates a program that would not be relocatable; any reference to AREA1 will require the use of register 0, since register 2 cannot produce a valid displacement. References to AREA2, however, will make use of register 2.

This restriction does not prevent a relocatable program from referring to actual storage locations by means of absolute expressions. For example, to reference a permanently allocated interrupt location at storage address 24, the following statement is perfectly correct:

```
LPSW     24
```

```
0000                START   0
                    USING   *,0
                    USING   *+2048,2
                      .
                      .
                      .
2000     AREA1      DS      20H
                      .
                      .
                      .
4000     AREA2      DS      10F
```

Figure 17. Example of Coding Using Base Register Zero

PROGRAM LINKING INSTRUCTIONS

The program linking assembler instructions allow the programmer to symbolically link independently assembled programs that will be loaded and executed together. Symbolic linkages between programs are created by means of symbols that are defined in one program and used as operands in another program. Such symbols are termed linkage symbols. A linkage symbol is called an "entry-point symbol" in the program in which it is defined; it is an "external symbol" in the program in which it is used as an operand. External and entry-point symbols are also described in the section "Symbols."

Every linkage symbol must be properly identified as such in the source program. A linkage symbol used as an external symbol is identified in each using program by the EXTRN instruction. A linkage symbol used

as an entry point must be identified in the defining program by the ENTRY instruction.

A program name (defined in the name field of a START statement) is also considered an entry point. A program name, however, does not have to be identified as an entry point by the ENTRY instruction.

ENTRY - Identify Entry-Point Symbol

The ENTRY instruction identifies an entry-point symbol to the program. Each such entry-point symbol (except a program name) must be identified by a separate ENTRY instruction. The format of the ENTRY instruction statement is:

Name	Operation	Operand
Not used	ENTRY	A relocatable symbol

The relocatable symbol in the operand field is a symbol defined elsewhere in the program, which may be used as an entry point by other programs. A symbol that is not defined in the program will be flagged in the listing as an undefined symbol. Any symbol in the name field will not be used.

An ENTRY statement must be immediately preceded by either the START statement, an EXTRN statement, or another ENTRY statement. It cannot appear in a program unless the START statement has been used.

If an ENTRY statement is incorrectly placed or if the operand is invalid, the statement will not be used. An error flag will appear in the listing.

In the following sequence, SQRT is identified as an entry-point symbol. Note that the ENTRY statement appears immediately after the START statement:

```
SUBRO    START   0
         ENTRY   SQRT
         .
         .
         .
SQRT     STM     1,10,SAVE
```

EXTRN - Identify External Symbol

The EXTRN instruction identifies a linkage symbol as an external symbol that will be referred to in this program. Each such external symbol must be identified by a separate EXTRN instruction. The format of the EXTRN instruction statement is:

Name	Operation	Operand
Not used	EXTRN	A relocatable symbol

The relocatable symbol in the operand field must be defined in another program, and identified in that program as an entry-point symbol by either the START or ENTRY instruction. Any symbol in the name field will not be used.

An EXTRN statement must be immediately preceded by either the START statement, an ENTRY statement, or another EXTRN statement. An EXTRN statement cannot appear in a program unless the START statement has been used. Not more than 14 EXTRN statements may appear in a program. If there are more than 14 statements, the symbol in each excess statement will be flagged as undefined.

If an EXTRN statement is incorrectly placed or if the operand is invalid, the statement will not be used. An error flag will appear in the listing.

As an example, if MTPLY is an entry-point symbol in another program, the using program identifies it as an external symbol, thus:

```
         EXTRN   MTPLY
```

The correct use of an external symbol elsewhere in a program is described below.

LINKING CONVENTIONS

The only way that an external symbol may be referenced is to (1) identify it with the EXTRN instruction, (2) create an address constant from the external symbol, (3) load the constant into a general register, and (4) branch to the address via the register or use the register for base addressing.

For example, to link to a program named SINE, the following coding might be used:

```
PROGA    START   1000
         EXTRN   SINE
           .
           .
           .
         L       4,ADSINE
         BALR    15,4
           .
           .
           .
ADSINE   DC      A(SINE)
```

In this example, SINE would be given a value of zero at assembly time; four bytes of zeros would be reserved at the symbolic location ADSINE. When the programs are loaded, the relocating loader will add to the four bytes of zeros the effective address assigned to SINE.

If the programmer wished to link, say, to a location 12 bytes past SINE, the constant could be created as follows:

```
ADSINE   DC      A(SINE+12)
```

The relocating program loader will add 12 to the effective address of SINE and place the sum in the four bytes at ADSINE. The expression in which the external symbol is used must be a relocatable expression.

Another method of linking to SINE+12 is:

```
         START   1000
         EXTRN   SINE
           .
           .
           .
         USING   SINE,4
         L       4,ADSINE
           .
           .
           .
       { BAL     15,SINE+12 }
       { BAL     15,12(0,4) }
       { BAL     15,12(4)   }
           .
           .
ADSINE   DC      A(SINE)
```

In the above sequence, either BAL instruction can be used; if BAL 15,12(0,4) or BAL 15,12(4) is used, the USING statement may be omitted, since implicit base addressing is not involved.

A return branch from the program named SINE may be made via the registers without making any reference to a linkage symbol. Thus, if the branch to SINE was:

```
         BALR    10,4
```

the return branch may be:

```
         BCR     15,10
```

Limitations on Program Linking

The order in which independently assembled programs are loaded generally determines the extent to which they can link to one another. The program(s) containing the entry point(s) must be loaded before the program(s) that will reference these points as external symbols. Note, however, that program names are not affected by this restriction. A program loaded first may refer to programs loaded after it by their names, using the facilities of the relocating loader. In addition, the use of relocating loader control cards can remove all restrictions on linking.

In the following situation, two independently assembled programs, Program A and Program B, are to be executed together. Each program contains the coding shown in Figure 18.

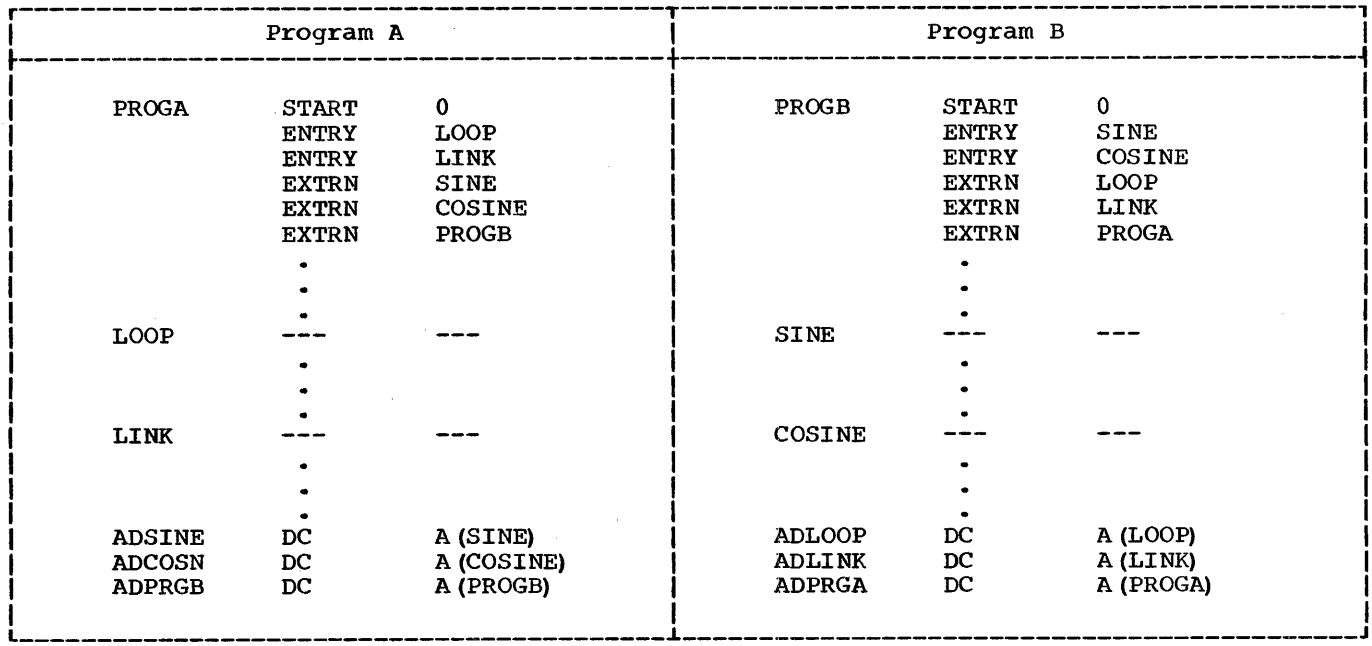
If Program A is loaded first, it can refer to Program B only by its name, PROGB. Program B however, can refer to Program A by its name, PROGA, and its entry points, LOOP and LINK. If the loading order is reversed, then Program B can refer to Program A only by its name, whereas Program A can refer to Program B by its name and by its entry points, SINE and COSINE.

Thus, if a common data area is to be used by two independently assembled programs, the data area should be assembled separately and then loaded first to enable both programs to refer freely to it.

Program Relocation and Linking

Programs that will be linked together at object time must be relocatable. To be relocatable, a program must:

1. Contain all the information required by the relocating loader.
2. Not use absolute expressions to refer to any area that can be relocated.



Program A			Program B		
PROGA	START	0	PROGB	START	0
	ENTRY	LOOP		ENTRY	SINE
	ENTRY	LINK		ENTRY	COSINE
	EXTRN	SINE		EXTRN	LOOP
	EXTRN	COSINE		EXTRN	LINK
	EXTRN	PROGB		EXTRN	PROGA
	.			.	
	.			.	
	.			.	
LOOP	---	---	SINE	---	---
	.			.	
	.			.	
	.			.	
LINK	---	---	COSINE	---	---
	.			.	
	.			.	
	.			.	
ADSINE	DC	A (SINE)	ADLOOP	DC	A (LOOP)
ADCOSN	DC	A (COSINE)	ADLINK	DC	A (LINK)
ADPRGB	DC	A (PROGB)	ADPRGA	DC	A (PROGA)

Figure 18. Example of Program Linking

3. Identify all entry-point and external symbols that will be used by the ENTRY and EXTRN instructions, respectively.
4. Specify all address constants (type A constants) that represent relocatable expressions with a length of three or four.
5. Not use general register zero as a base register.

ASSEMBLER INSTRUCTION SUMMARY

Figure 19 contains all of the assembler instructions and the contents of their name and the operand fields.

Reference Summary for Assembler Instructions		
Name Field	Mnemonic	Operand Field
Not used	ICTL	The decimal value 1 or 25
An optional symbol	START	A self-defining value, a comma, or blank
Not used	ENTRY	A relocatable symbol
Not used	EXTRN	A relocatable symbol
Not used	CNOP	Two decimal values separated by a comma
An optional symbol	CCW	Four operands separated by commas
An optional symbol	DC	A single operand describing the constant
Not used	DROP	A simple absolute expression
An optional symbol	DS	A single operand describing the area to be reserved
Not used	EJECT	Not used
A required symbol	EQU	An expression
Not used	ORG	A relocatable expression
Not used	SPACE	A decimal value not exceeding 63
Not used	USING	A relocatable expression and a simple absolute expression, separated by a comma
Not used	END	A relocatable expression, a comma, or blank

Figure 19. Assembler Instruction Summary

This section describes those operations of the assembler program that have a direct bearing on preparing programs for assembly. Note that the use of the Basic Assembler is described in detail in the publication IBM System/360 Basic Programming Support Operating Guide, Form C28-6557.

ASSEMBLER PROCESSING

The assembler is a two-phase program. It is provided as two decks of cards, one for each phase.

Phase 1

During the first phase, the assembler produces a symbol table (subsequently described) and intermediate text for use in the second phase. When the tape option is used, the intermediate text (but not the symbol table) is placed on tape. When the IBM 1442-2 Card Read-Punch is used in a card option system, this intermediate text is punched into the first 24 columns of each source program card. Because the intermediate text punched into the source card is still symbolic and pertains to the statement portion of the particular card only, the source program can be reassembled without being repunched. When the IBM 1402 Card Read-Punch is used in a card option system, this intermediate text is punched into the first 24 columns of a new card along with the first 47 columns of the source statement, column 72, and the Identification-Sequence Field (columns 73-80). If Phase 1 is successful, a 12 punch will appear in the first column of every card containing intermediate text.

The input to the first phase consists of the Phase 1 deck of the assembler followed by the source program. If the card option is used, blank cards must be available in the punch unit, for the symbol table.

One card will be punched for every six symbols defined in the program. The maximum number of symbols that can be defined is determined by main storage size, as explained in the section "Symbol Table." If the assembler is operating on a machine with 8,192 storage bytes, approximately 50 blank cards will be sufficient to handle the maximum number of symbols allowed; for 16,384 bytes, 230 blank cards; for 32,768 bytes, 570 cards; and for 65,536 bytes, 690 cards.

Phase 2

The assembler produces the program listing and object program during the second phase. The format of Phase 2 input varies with the option used.

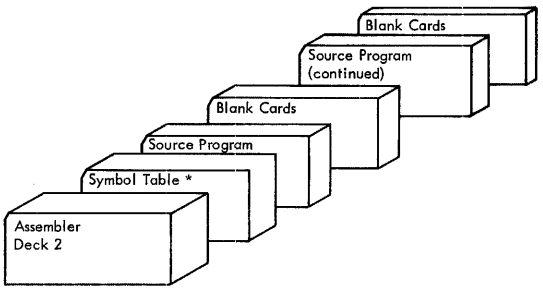
For the tape option, input is on cards and tape. The card input consists of the Phase 2 deck of the assembler. The tape input is the tape created in Phase 1. If the object program is to be produced on cards, blank cards should be provided at the approximate ratio of 10 blank cards for every 100 original source program cards. If the object program is to be placed on tape, blank cards are not required.

For card option, the second deck of the assembler is loaded followed by the repunched source program when the IBM 1442-2 Card Read-Punch is used, and by the newly punched intermediate deck when the IBM 1402 Card Read-Punch is used. If the second phase does not immediately follow the first phase, the symbol table will not be in storage. Consequently, it will be necessary to load the symbol table deck produced by Phase 1. It is placed between the assembler and source program decks. (See Figure 20.)

When the IBM 1442-2 Card Read-Punch is used, the assembler will accumulate the assembled object program in storage. When the storage area is full, and the next input card is not blank, the operator will be notified to insert blank cards in the 1442-2 Card Read-Punch for punching the object program. As each blank card is punched, it will be directed to the stacker reserved for the object deck. If a blank card is encountered when none is needed, the card will be directed to the other stacker, which is for the input cards. The remaining source cards will then be read, and the cycle repeated.

Operator intervention may be avoided, in a 1442 card system, by interleaving blank cards with the source program before starting Phase 2 (see Figure 20) at approximately the following ratios:

Main Storage Size	Approximate Ratio of Blank Cards to Source Program Cards
8,192	15 blanks every 150 source cards
16,384	80 blanks every 800 source cards
32,384	200 blanks every 2000 source cards
65,536	450 blanks every 4500 source cards



*Only required when Phase 2 does not immediately follow Phase 1.

Figure 20. Phase 2 Input for Use with IBM 1442-2 Card Read-Punch

If these ratios are observed, it should not be necessary for the operator to intervene and the time required to assemble the program will be reduced.

Blank cards may also be interleaved for Phase 1; their presence will not affect this phase of the assembly, except for time required to read the blank cards.

When the IBM 1402 Card Read-Punch is used, the assembler will punch an object program card as soon as one is assembled in storage.

PROGRAM LISTING

A program listing (if requested) will be produced for every assembly, provided an IBM 1443 Model 2 Printer, IBM 1403 Printer, or an IBM 1052 Printer-Keyboard is available. Each statement in the source program will appear on a separate line of the listing unless the suppress option is used. If the suppress option is used, only those statements containing errors will be listed. The programmer avails himself of the suppress option by indicating to the machine operator that he does not wish a listing. More detailed information on the suppress option is contained in the description of "Configuration Cards" in the publication IBM System/360 Basic Programming Support Operating Guide for Basic Assembler and Utilities, Form C28-6557.

The program listing will consist of five fields, arranged from left to right, as follows.

Flags: This field (print positions 1-10) will contain, left-justified, a flag(s) to signal possible errors in the statement. Each flag will be represented by a single alphabetic character. See the topic "Error Notification."

Location Counter: This field (print positions 12-17) will contain the Location Counter value (in hexadecimal) assigned to the statement.

Assembled Output: This field (print positions 20-39) will contain the hexadecimal representation of the binary digits generated from the statement.

Source Statement: This field (print positions 40-111) will contain a column-for-column reproduction of the contents of the source statement. For the 1402 card-option, where statements begin in column 1, only columns 1-47 will be reproduced.

Identification-Sequence Field: This field (print positions 113-119) will be a reproduction of columns 73-80 of the source card.

ERROR NOTIFICATION

The flags produced on the program listing for various source program errors are shown in the following list. Any error that causes the assembler to either a) ignore the instruction or b) assemble zeros in the operand field of the instruction will halt further evaluation of the instruction for other errors. Therefore, when correcting such an error, the user is advised to check for any other errors in the instruction.

Flag	Cause
* A	Expression not simply relocatable.
* B	START, EXTRN, ENTRY or ICTL out of order.
* C	Location counter overflow.
* E	More than 14 EXTRNs.
* F	Operand field format error or self-defining value in operand field too large.
* G	DC, D, or E range error.
I	Expression can not be mapped into base and displacement.
* J	Symbol table full.
K	Relocation list dictionary buffer table full.
* L	Name field error.
* M	Multiple defined symbol.

*	N	Statement not used. This flag is normally accompanied by other flags which define the reason the statement was not used. If it appears alone, it indicates that the statement was completely extraneous. If the flag (N) appears by itself when using a 1442 card option system, it indicates that the source statement has been modified since a previous assembly but the intermediate text field (columns 1-24) has not been left blank. See section "Reassembly Procedure."
*	O	Invalid OP code.
	R	Expression not absolute.
*	S	Specification error.
*	T	Value too large.
	U	Undefined symbol.
*	V	ORG or EQU symbol not previously defined.
	W	Unused mask bits (37-39) in CCW not zero.
	X	Duplicate entry statement.
*	Y	Negative expression.
*	Z	Column 72 not blank.

Note: The * indicates those flags which may be punched in the intermediate text cards produced by Phase 1 in card-option systems. For systems without the ability to produce program listings, these flags provide a limited form of error notification. It should be noted that the intermediate text cards produced by Phase 1 contain an A, B, or C in column 1 if they are error free. Cards in error have a J, K, L, or M in column 1. Error flags are located in columns 23-24 on cards with a J or K in column 1. The error flags appear in columns 21-24 on cards beginning with L or M.

OBJECT PROGRAM OUTPUT

The object program is generated by the assembler as a deck of cards or card images on tape acceptable as input to the loaders. It is the programmer's responsibility to inform the operator about the medium (cards or tape) on which the object deck is to be placed. Detailed information on this option can be found in the publication IBM System/360 Basic Programming Support Operating Guide, Form C28-6557. Four types of cards constitute the object program deck. It should be noted that detailed descriptions of each of the four types of cards may be found in the publication IBM System/360 Basic Programming Support Basic Utilities, Form C28-6505. General descriptions of each follow.

External Symbol Dictionary (ESD) Card

An ESD card is generated for each START, ENTRY, and EXTRN statement. The ESD card contains coded information that is used by the relocating loader.

Text (TXT) Card

The Text cards contain the output assembled from the source program. Up to 56 contiguous bytes of output are punched into each Text card. Each Text card also contains the storage address at which the first byte in the card is to be loaded.

Relocation List Dictionary (RLD) Card

The purpose of RLD cards is to indicate to the relocating loader those address constants that will have to be changed if the program is loaded at a location different from its assembled location. Address constants of this type are defined in the source program by (1) relocatable expressions in type A DC statements and (2) relocatable expressions specifying data addresses in CCW statements; that is, the second operands of CCW statements. Up to 13 address constants are punched into each RLD card.

The maximum number of address constants (that is, the type described above) that can be defined in a program is determined by the size of main storage thus:

Main Storage Size (in Bytes)	Maximum Number of Address Constants
8,192	30
16,384	60
32,768	120
65,536	240

Load End Card

This card is produced when the assembler encounters the END statement. The Load End card also contains the address to which control is to be transferred when the program has been loaded, if one was specified in the END statement.

PATCHING OBJECT PROGRAMS

The programmer may modify his object program at execution time through the use of a Replace card. This card is completely described in the publication IBM System/360 Basic Programming Support Basic Utilities, Form C28-6505.

REASSEMBLY PROCEDURE

A special reassembly procedure is provided for assemblies using the IBM 1442-2 Card Read-Punch without tape. This procedure will enable a partially or completely assembled program to be reassembled in less time than a new assembly would require.

The program that is to use the reassembly procedure may be changed in any manner. New symbols can be added and existing ones redefined, provided that the symbol table is not full. New statements also can be included in the program.

The reassembly procedure is faster than the new assembly procedure because the assembler does not have to repunch the first 24 columns of those source program cards whose statements have not been changed. Hence, the fewer the changes, the faster the assembly.

The input to the first phase of a reassembly consists of the first deck of the assembler, followed in order by the previously punched symbol table decks, the source program with any changes, and the necessary number of blank cards into which a new symbol table will be punched. Note that any changed source program cards must be repunched, leaving columns 1-24 blank. This also applies to source program cards that did not have a 12-punch in column 1 as the result of the previous assembly.

The Phase 2 input and output of a reassembly is identical with the second phase of a new assembly (see topic "Phase 2").

SYMBOL TABLE

For every program assembled, a table is created of the symbols in that program. This is the symbol table; each entry in the table records the attributes and other pertinent information about a particular symbol.

The maximum size of the symbol table and, hence, the maximum number of symbols that can be defined in a program is determined by the size of main storage, thus:

Approximate Main Storage Size (in Bytes)	Approximate Number of Symbols in Table
8,192	275
16,384	1299
32,768	3347
65,536	4094

All symbols defined in a program (including the program name and external symbols) are entered in the symbol table providing the following conditions are met:

1. The symbol table is not full.
2. The symbol conforms to the rules governing symbol specifications (see the topic "Symbols").
3. The symbol does not appear in the name field of an assembler instruction that does not allow the specification of a name. See Figure 19 for a list of these instructions.
4. The symbol is not already contained in the symbol table. For multiple defined symbols, only the first definition of the symbol results in an entry in the symbol table. Additional definitions of the same symbol are simply flagged.

Any reference in the operand field to a symbol not in the symbol table will be considered undefined; the statement will be flagged. An undefined symbol in a machine instruction statement will cause the entire instruction (except the operation code) to be set to zero.

Symbol Table Overflow

If there are undefined symbols because the symbol table is full, three corrective procedures are available:

1. The assembled object deck produced by the assembler can be corrected with Replace (REP) cards before loading the program. Replace cards, a feature of the loaders, are used to alter an object deck on a byte-for-byte basis.
2. Reduce the number of symbols and then reassemble or run a new assembly.
3. Divide the program into segments and assemble each program segment separately.

Relative addressing may be used to reduce the number of symbols defined in a program. For example, the following sequence:

```
BEGIN    LA      3,10
         LA      1,0
LOOP     L       2,AUGEND(1)
         A       2,ADDEND(1)
         ST      2,SUM(1)
         LA      1,4(1)
         BCT     3,LOOP
         BC      15,OUT
AUGEND   DS      10F
ADDEND   DS      10F
SUM      DS      10F
OUT      LR      3,1
         .
         .
         .
```

could also be written:

```
BEGIN    LA      3,10
         LA      1,0
         L       2,AUGEND(1)
         A       2,AUGEND+40(1)
         ST      2,AUGEND+80(1)
         LA      1,4(1)
         BCT     3,*-16
         BC      15,OUT
AUGEND   DS      30F
OUT      LR      3,1
         .
         .
         .
```

thus eliminating four symbols. Note that the branch address of the BC instruction is given relative to AUGEND rather than the Location Counter, since any boundary alignment caused by the DS statement would change the number of bytes between the BC and LR instruction.

Note: Using the IBM 1442-2 Card Read-Punch reassembly procedure, the programmer must eliminate all undefined symbols from those cards that refer to such symbols in the operand field. The cards in which the undefined symbols appear in the name field can be left as they are. Since the symbol table is full, no new symbols may be defined for the reassembly.

If, in addition to reducing the number of symbols, the programmer wants to replace defined symbols (that is, symbols in the symbol table) with new symbols, the entire source program deck, with changes, must (for the IBM 1442-2 Card Read-Punch card option) be reproduced with columns 1-24 blank prior to assembling the program. For the tape option or the IBM 1402 Card Read-Punch card option, the source deck with the desired changes can be used as is.

APPENDIX A. CHARACTER CODES

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	' (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	

The table in this appendix provides for direct conversion of decimal and hexadecimal numbers in these ranges:

Hexadecimal	Decimal
000 to FFF	0000 to 4095

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

Hexadecimal	Decimal
1000	4096
2000	8192
3000	12288
4000	16384
5000	20480
6000	24576
7000	28672
8000	32768
9000	36864
A000	40960
B000	45056
C000	49152
D000	53248
E000	57344
F000	61440

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
010	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
020	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
030	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
040	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
050	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
060	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
070	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
080	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
090	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A0	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B0	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C0	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D0	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E0	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F0	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255
100	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
110	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
120	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
130	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
140	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
150	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
160	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
170	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
180	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
190	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A0	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B0	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C0	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D0	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E0	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F0	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
200	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
210	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
220	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
230	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
240	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
250	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
260	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
270	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
280	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
290	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A0	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B0	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C0	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D0	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E0	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F0	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767
300	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
310	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
320	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
330	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
340	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
350	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
360	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
370	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
380	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
390	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A0	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B0	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C0	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D0	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E0	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F0	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
400	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
410	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
420	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
430	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
440	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
450	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
460	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
470	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
480	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
490	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A0	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B0	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C0	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D0	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E0	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F0	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
500	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
510	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
520	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
530	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
540	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
550	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
560	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
570	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
580	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
590	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A0	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B0	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C0	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D0	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E0	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F0	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
600	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
610	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
620	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
630	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
640	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
650	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
660	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
670	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663
680	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
690	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A0	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B0	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C0	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D0	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E0	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F0	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791
700	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
710	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
720	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
730	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
740	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
750	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
760	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
770	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
780	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
790	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A0	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B0	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C0	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D0	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E0	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F0	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
800	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
810	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
820	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
830	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
840	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
850	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
860	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
870	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
880	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
890	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A0	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B0	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C0	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D0	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E0	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F0	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303
900	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
910	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
920	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
930	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
940	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
950	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
960	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
970	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
980	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
990	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A0	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B0	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C0	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D0	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E0	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F0	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A00	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A10	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A20	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A30	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A40	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A50	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A60	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A70	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A80	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A90	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA0	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB0	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC0	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
AD0	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE0	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF0	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
B00	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B10	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B20	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B30	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B40	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B50	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B60	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B70	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B80	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B90	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA0	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB0	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC0	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD0	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE0	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF0	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C00	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C10	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C20	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C30	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C40	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C50	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C60	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C70	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C80	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C90	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA0	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB0	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC0	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD0	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE0	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF0	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327
D00	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D10	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D20	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D30	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D40	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D50	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D60	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D70	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D80	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D90	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA0	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB0	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC0	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD0	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE0	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF0	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
E00	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E10	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E20	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E30	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E40	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E50	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E60	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E70	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E80	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E90	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA0	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB0	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC0	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED0	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE0	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF0	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839
F00	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F10	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F20	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F30	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F40	3904	0905	0906	0907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F50	3920	0921	0922	0923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F60	3936	0937	0938	0939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F70	3952	0953	0954	0955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F80	3968	0969	0970	0971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F90	3984	0985	0986	0987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA0	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB0	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC0	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD0	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE0	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF0	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

This test program sorts, in ascending sequence, the 16 hexadecimal characters located at 'IN' and stores them at 'OUT'. (The following example is used to demonstrate instruction mix rather than model coding.)

```
FLAGS      LOC.CTR.     OBJECT    V1L0  SOURCE STATEMENT
                                         ICTL  25
           000000                        SAMPLE START 0            STARTING ADDR
           000000  05 D0                 GO     BALR  13,0         SET UP BASE REGISTER
                                  000002        USING *,13
           000002  D2 3F D 09E D 05E            MVC   OUT(64),IN  MOVE DATA TO OUT
           000008  41 60 D 09E                  LA    6,OUT       POINT TO TABLE TOP
           00000C  41 70 0 00F                  LA    7,15        SET FOR 15 PASSES
           000010  41 40 0 038          SET     LA    4,56        SET INDEX REGISTER
           000014  58 20 6 000                  L     2,0(0,6)    LOAD FROM TABLE TOP
           000018  58 34 6 004          LOAD    L     3,4(4,6)    LOAD FROM TABLE
           00001C  15 23                        CLR   2,3         COMPARE VALUES
           00001E  47 C0 D 02A                  BC    12,SUB      TOP = OR LESS BRANCH
           000022  17 23                        XR    2,3         EXCHANGE VALUES
           000024  17 32                        XR    3,2         EXCHANGE VALUES
           000026  17 23                        XR    2,3         EXCHANGE VALUES
           000028  50 34 6 004                  ST    3,4(4,6)    STORE LARGER BACK
           00002C  5B 40 D 05A          SUB     S     4,CON4      REDUCE INDEX
           000030  47 A0 D 016                  BC    10,LOAD     LOOP IF MORE TO SORT
           000034  50 20 6 000                  ST    2,0(0,6)    STORE IN TABLE TOP
           000038  5B 70 D 056                  S     7,CON1      REDUCE PASS COUNTER
           00003C  47 70 D 042                  BC    7,LOOP
           000040  82 00 D 0DE                  LPSW  ENDRUN      END OF RUN
           000044  41 66 0 004          LOOP    LA    6,4(6)
           000048  48 20 D 010                  LH    2,SET+2     MODIFY
           00004C  5B 20 D 05A                  S     2,CON4       INDEX
           000050  40 20 D 010                  STH   2,SET+2       INSTRUCTION
           000054  47 F0 D 00E                  BC    15,SET      RETURN
           000058  00000001             CON1    DC    F'1'        CONSTANT OF 1
           00005C  00000004             CON4    DC    F'4'        CONSTANT OF 4
           000060  00000005             IN      DC    X'00000005'
           000064  0000000A                     DC    X'0000000A'
           000068  00000001                     DC    X'00000001'
           00006C  00000007                     DC    X'00000007'
           000070  00000003                     DC    X'00000003'
           000074  0000000C                     DC    X'0000000C'
           000078  0000000F                     DC    X'0000000F'
           00007C  00000009                     DC    X'00000009'
           000080  0000000B                     DC    X'0000000B'
           000084  00000004                     DC    X'00000004'
           000088  00000000                     DC    X'00000000'
           00008C  0000000E                     DC    X'0000000E'
           000090  00000006                     DC    X'00000006'
           000094  0000000D                     DC    X'0000000D'
           000098  00000002                     DC    X'00000002'
           00009C  00000008                     DC    X'00000008'
           0000A0                        OUT     DS    16F         OUTPUT AND WORK AREA
                                                CNOP  0,8  ENSURE BOUNDARY ALIGNMENT
           0000E0  0002000000000000     ENDRUN  DC    X'0002000000000000'    PSW
           000000                               END   GO
```

APPENDIX D. SYSTEM/360 ASSEMBLERS-LANGUAGE FEATURES COMPARISON CHART

Features not shown below are common to all assemblers. In the chart:

Dash = Not allowed.
X = as defined in Operating System/360 Assembler Language Manual.

Feature	Basic Programming Support/360: Basic Assembler	7090/7094 Support Package Assembler	Other System/360 Assemblers[1]	OS/360 Assembler
No. of Continuation Cards/Statement (exclusive of macro-instructions)	0	0	1	2
Input Character Code	EBCDIC	BCD or EBCDIC	EBCDIC	EBCDIC
ELEMENTS:				
Maximum Characters per symbol	6	6	8	8
Character self-defining terms	1 Char. only	X	X	X
Binary self-defining terms	--	--	X	X
Length attribute reference	--	--	X	X
Literals	--	--	X	X
Extended mnemonics	--	X	X	X
Maximum Location Counter value	$2^{16}-1$	$2^{24}-1$	$2^{24}-1$	$2^{24}-1$
Multiple Control Sections per assembly	--	--	X	X
EXPRESSIONS:				
Operators	+-*	+-*/	+-*/	+-*/
Number of terms	3	16	3	16
Number of parentheses	--	--	1 Level	5 Levels
Complex relocatability	--	--	X	X
ASSEMBLER INSTRUCTIONS:				
DC and DS				
Expressions allowed as modifiers	--	--	--	X
Multiple operands	--	--	--	X
Multiple constants in an operand	--	--	Except Address Consts.	X

(Continued)

Appendix D: Assembler Languages--Features Comparison Chart (Continued)

Feature	Basic Programming Support/360: Basic Assembler	7090/7094 Support Package Assembler	Other System/360 Assemblers[1]	OS/360 Assembler
Bit length specifications	--	--	--	X
Scale modifier	--	--	X	X
Exponent Modifier	--	--	X	X
DC types	Except B, P, Z, V, Y, S	Except B, Y, V	X	X
DC duplication factor	Except A	Except A, S	Except S	X
DC duplication factor of zero	--	--	Except S	X
DC length modifier	Except H, E, D	Except S	X	X
DS types	Only C, H, F, D	Only C, H, F, D	X	X
DS length modifier	Only C	Only C	X	X
DS maximum length modifier	256	256	256	65,535
DS constant subfield permitted	--	--	X	X
COPY	--	--	--	X
CSECT	--	--	X	X
DSECT	--	--	X	X
ISEQ	--	--	X	X
LTORG	--	--	X	X
PRINT	--	--	X	X
TITLE	--	X	X	X
COM	--	--	--	X
ICTL	1 oprnd 1 or 25 only	1 oprnd	X	X
USING	2 oprnds oprnd 1 reloc only	2 oprnds oprnd 1 reloc only	6 oprnds	X
DROP	1 oprnd only	1 oprnd only	5 oprnds	X

(Continued)

Appendix D: Assembler Languages--Features Comparison Chart (Continued)

Feature	Basic Programming Support/360: Basic Assembler	7090/7094 Support Package Assembler	Other System/360 Assemblers[1]	OS/360 Assembler
CCW	oprnd 2 reloc only	oprnd 2 reloc only	X	X
ORG	no blank oprnd	no blank oprnd	X	X
ENTRY	1 oprnd only	1 oprnd only	1 oprnd only	X
EXTRN	max 14 1 oprnd only	1 oprnd only	1 oprnd only	X
CNOP	2 dec digits	2 dec digits	2 dec digits	X
PUNCH	--	--	--	X
REPRO	--	--	X	X
Macro Instructions	--	--	X	X

APPENDIX E. HEXADECIMAL TO MNEMONIC OPERATION CODE TABLE

The table in this appendix provides for easy conversion from the hexadecimal equivalent of the actual machine operation codes to their associated assembler mnemonic operation codes.

First Hexadecimal Digit \ Second Hexadecimal Digit	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0					SPM	BALR	BCTR	BCR	SSK	ISK	SVC						
1	LPR	LNR	LTR	LCR	NR	CLR	OR	XR	LR	CR	AR	SR	MR	DR	ALR	SLR	RR
2	LPDR	LNDR	LTDR	LCDR	HDR				LDR	CDR	ADR	SDR	MDR	DDR	AWR	SWR	RR
3	LPER	LNER	LTER	LCER	HER				LER	CER	AER	SER	MER	DER	AUR	SUR	RR
4	STH	LA	STC	IC	EX	BAL	BCT	BC	LH	CH	AH	SH	MH		CVD	CVB	RX
5	ST				N	CL	O	X	L	C	A	S	M	D	AL	SL	RX
6	STD								LD	CD	AD	SD	MD	DD	AW	SW	RX
7	STE								LE	CE	AE	SE	ME	DE	AU	SU	RX
8	SSM		LPSW		WRD	RDD	BXH	BXLE	SRL	SLL	SRA	SLA	SRDL	SLDL	SRDA	SLDA	RS
9	STM	TM	MVI	TS	NI	CLI	OI	XI	LM				SIO	TIO	HIO	TCH	RS or SI
A																	RS or SI
B																	RS or SI
C																	SS
D		MVN	MVC	MVZ	NC	CLC	OC	XC					TR	TRT	ED	EDMK	SS
E																	SS
F		MVO	PACK	UNPK					ZAP	CP	AP	SP	MP	DP			SS

2/65:20M-EO-68

READER'S COMMENTS

Title: IBM System/360 Basic Programming Support
Basic Assembler Language

Form: C28-6503-3

Is the material:	Yes	No
Easy to Read?	___	___
Well organized?	___	___
Complete?	___	___
Well illustrated?	___	___
Accurate?	___	___
Suitable for its intended audience?	___	___

How did you use this publication?
___ As an introduction to the subject ___ For additional knowledge
Other _____

fold

Please check the items that describe your position:
___ Customer personnel ___ Operator ___ Sales Representative
___ IBM personnel ___ Programmer ___ Systems Engineer
___ Manager ___ Customer Engineer ___ Trainee
___ Systems Analyst ___ Instructor Other _____

Please check specific criticism(s), give page number(s), and explain below:
___ Clarification on page(s)
___ Addition on page(s)
___ Deletion on page(s)
___ Error on page(s)

Explanation:

CUT ALONG LINE

fold

Name _____

Address _____

FOLD ON TWO LINES, STAPLE AND MAIL
No Postage Necessary if Mailed in U.S.A.

staple

fold

fold

FIRST CLASS
PERMIT NO. 81

POUGHKEEPSIE, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY

IBM CORPORATION
P. O. BOX 390
POUGHKEEPSIE, N. Y. 12602

ATTN: PROGRAMMING SYSTEMS PUBLICATIONS
DEPT. D58

fold

fold

CUT ALONG LINE

Printed in U.S.A.

C28-6503-3

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601

staple