NASP–9214–06
NAS Enroute Stage A
Contract FA65WA–1395

NAS OPERATIONAL SUPPORT SYSTEM

USER'S MANUAL

IBM 9020 Data Processing System Basic Assembly Language (BALASM)

Model A3d2.0

15 August 1973

The Basic Assembly Language is a symbolic programming language that provides a convenient way to make full use of the capabilities of the IBM 9020 Data Processing System. The assembly language provides flexibility in program and machine usage and assists the programmer in checking, standardizing, and documenting programs. This revision conforms with the C42 release of the BAL Assembler.

This document replaces NASP–9214–05 and is compatible with the NOSS tapes which support the NAS Model A3d2.0 tape release.

NAS Programming
IBM Federal Systems Division
NAFEC, Atlantic City, New Jersey

NOTICE: This document is stored on magnetic tape. In making changes, please submit a copy of the document containing the marked up changed pages. Use pages from the current level only. If necessary, attach insert material on separate pages, but do not retype material that has not changed. This will only slow down the update process and increase the chance of error.

PREFACE

This document, prepared by the International Business Machines Corporation, is submitted to the Federal Aviation Administration in accordance with the requirements of Contract FA65WA–1395.

These change pages update the NAS Operational Support System (NOSS) User's Manual for the IBM 9020 Data Processing System Basic Assembly Language (BALASM), dated 24 May 1974, to make it compatible with the NOSS tapes which support the NAS Model A3d2.1 System.

In using this document, note that National Airspace System Configuration Management Document (NAS–MD) should be substituted wherever System Program Office Configuration Management Directive (SPO–MD) appears.

CONTENTS

SECTION 1. INTRODUCTION 1–1

Assembly Language Features 1–1

Statement Fields 1–1
- Name Field 1–1
- Operation Field 1–2
- Operand Field 1–2
- Comments Field 1–2
- Identification-Sequence Field 1–3

SECTION 2. WRITING ASSEMBLY LANGUAGE STATEMENTS 2–1

Character Set 2–1

Symbols 2–1
- Relocatable and Absolute Symbols 2–1
- Defining Symbols 2–1
- Previously Defined Symbols 2–2
- External and Entry-Point Symbols 2–2
- General Restrictions on Symbols 2–3

Location Counter 2–3
- Location Counter References 2–3

Self-Defining Values 2–3
- Decimal Self-Defining Value 2–4
- Hexadecimal Self-Defining Value 2–4
- Character Self-Defining Value 2–4
- Symbol Attribute Self-Defining Value 2–4
- Using Self-Defining Values 2–4

Literals 2–4
- Literal Format 2–5
- Literal Pool 2–5

Expressions 2–5

Relative Addressing 2–6
- Evaluation of Expressions 2–6
- Types of Expressions 2–6
- External Symbols in Expressions 2–7

SECTION 3. MACHINE INSTRUCTION STATEMENT 3–1

Instruction Alignment and Checking 3–1

Operand Format 3–1
- Operand Fields and Subfields 3–1
- Effective Addresses – Explicit and Implied 3–1
- Lengths – Explicit and Implied 3–1

Machine Instruction Mnemonic Codes 3–5
- Machine Instruction Examples 3–5
- Extended Mnemonic Codes 3–10

SECTION 4. ASSEMBLY INSTRUCTION STATEMENTS 4–1

Symbol Definition Instructions 4–1
- EQU – Equate Symbol 4–1
- MAX and MIN – Equate Symbol 4–2
- TEQU – Temporarily Equate a Symbol 4–3

Data Definition Instructions 4–3
- DC – Define Constant 4–4
- DS – Define Storage 4–11
- CCW – Define Channel Command Word 4–13

Program Sectioning and Linking 4–13
- First Control Section 4–14
- Symbolic Linkages 4–14
- START – Start Program 4–14
- CSECT – Identify Control Section 4–15
- DSECT – Identify Dummy Section 4–15
- COM – Identify Common Control Section 4–16
- ENTRY – Identify Entry-Point Symbol 4–16
- EXTRN – Identify External Symbol 4–16
- QUAL – Qualify Names 4–16

Base Register Instructions 4–17
- USING – Use Base Address Register 4–17
- DROP – Drop Base Address Register 4–18
- Programming with the USING Instruction 4–18

Listing Control Instructions 4–20
- TITLE – Identify Assembly Output 4–20
- EJECT – Start New Page 4–21
- SPACE – Space Listing 4–21
- DOUBL – Double Space 4–21
- PRINT – Print Optional Data 4–21
- NLIST – Suppress Listing 4–22
- LIST – Resume Listing 4–22

Program Control Instructions 4–22
- ICTL – Input Format Control 4–22
- ISEQ – Input Sequence Checking 4–22
- SSEQ – Suppress Sequence Checking 4–22
- ORG – Reset Location Counter 4–23
- LTORG – Begin Literal Pool 4–23
- CNOP – Conditional No Operation 4–24
- END – End Program 4–25
- SPEM – Suppress Possible Error Messages 4–25
- RPEM – Resume Possible Error Messages 4–25
- LIB – Library Update 4–26
- Assembling With a Compool 4–26
- PSEG – Select Compool Segment 4–26
- DEBUG Instructions 4–27

Appendix A. CHARACTER CODES A–1

Appendix B. HEXADECIMAL-DECIMAL NUMBER CONVERSION B–1

CONTENTS (Continued)

Appendix C. CONSTANT DEFINITION C–1

Appendix D. ASSEMBLY INSTRUCTION REFERENCE D–1

Appendix E. ASSEMBLY DEFINITIONS OF SYSTEM SYMBOLS E–1

Appendix F. BAL PROCEDURES F–1

Appendix G. SAMPLE PROGRAM LISTING G–1

Appendix H. I/O REQUIREMENTS H–1

Appendix I. STORAGE REQUIREMENTS I–1

ILLUSTRATIONS

Figure 4–1. Use of DS Instruction 4–12

Figure 4–2. Base Register Assignment 4–19

Figure 4–3. Base Register Assignment 4–19

Figure 4–4. Addressing Dummy Sections 4–20

Figure 4–5. Addressing External Programs 4–20

Figure 4–6. CNOP Position Specifications 4–24

Figure F–1. Utility Programming System for the IBM 9020 Data Processing System F–1

Figure F–2. Flow of Input to the Assembler F–2

Figure F–3. Sample of BAL Source Deck Accepted by the Assembler F–2

TABLES

Table 2–1. Type Attributes 2–2

Table 3–1. Machine Instruction Format 3–2

Table 3–2. Details of Address Specification 3–4

Table 3–3. Details of Length Specification in SS Instruction 3–5

Table 3–4. Machine Instruction Mnemonic Codes 3–6

Table 3–5. Extended Mnemonics 3–10

Table 4–1. Assembly Instructions 4–1

Table 4–2. Type Subfield Codes 4–5

Table 4–3. Channel Command Word Formats 4–13

Table A–1. Character Codes A–1

Table B–1. Hexadecimal-Decimal Number Conversion B–1

Table C–1 Summary Information for Defining Constants C–1

Table D–1. Reference Summary for Assembly Instructions D–1

Table E–1. System Symbols Defined by Assembly E–1

Table H–1. I/O Requirements H–1

Table I–1. Assembler SE's vs Variables (.WORK2 Available) I–1

Table I–2. Assembler SE's vs Variables (.WORK2 Not Available) I–1

SECTION 1. INTRODUCTION

The basic assembly language, which is referred to in this publication as the assembly language, is a symbolic programming language for the IBM 9020 Data Processing System.

Programs written in the source language are processed by an assembly program. The output consists of an object program suitable for loading and execution, and a listing of both the source program and the resulting object program. As the program is assembled, it is carefully analyzed for actual errors or potential errors in the use of the assembly language.

ASSEMBLY LANGUAGE FEATURES

The assembly language provides mnemonic operation codes for all machine instructions and extended mnemonic codes for certain branch instructions. Because program elements may be named, the programmer can employ symbolic addresses to refer to main storage, general registers, floating-point registers, etc. Data formats can be expressed with maximum flexibility. Listing controls enable the user to head the listing pages, annotate and space the listing, and control its content.

The programmer can shift the clerical burden of calculating base addresses and displacements to the assembly program, and yet retain complete control of base register usage. He decides which registers are to be used as base registers and loads them with the proper values.

Finally, the language provides a simple means of specifying and using a relocatable program that comprises as many or as few control sections as the user requires. The programmer may refer to data and/or transfer control to instruction sequences that are in other programs, i.e., programs that have been assembled separately from his own.

STATEMENT FIELDS

An assembly language source program consists of a sequence of statements punched into cards. An assembly language statement is composed of from one to four fields; starting from left to right, they are: name field, operation field, operand field, and comments field.The identification-sequence field (columns 73–80) is not part of the statement.

There are five general rules that must be observed when writing assembly language statements:

1. Every statement requires an operation field; additional fields are optional.

2. The fields in a statement must be in order, and they must be separated from one another by at least one blank, which acts as the field delimiter.

3. Because a blank is used as a delimiter, the name, operation, and operand fields must not contain embedded blanks. However, a blank may occur within a character self-defining value, a character constant, or a character literal.

4. Column 72 should always be blank.

5. If a card is completely blank (or if there is no operation field), it will be ignored by the assembly program.

In the various examples and statement formats throughout this publication, characters and words that may be written in assembly language statements are printed in capital letters. Some of these characters and words have special meaning to the assembly program (e.g., instruction mnemonics); others are representative examples of what might be written in statements.

Specifications for the various fields are presented in the following text.

Name Field

The name field is used to assign a symbolic name to a statement. Other statements can refer to a particular statement by its symbolic name. If a name is used, it must start in the begin column of the statement and it may occupy up to eight columns. The begin column is normally column 1, but it may be changed by the use of an ICTL assembly instruction (see Section 4). A name is always a symbol and must conform to the rules for symbols (see Section 2). The following example shows the symbol FIELD234 used as a name.

Name	Operation	Operand
FIELD234	DS	CL200

If the begin column is blank, the assembly program assumes that the statement has no name. The begin column is also used to indicate that a card is a comments card (see "Comments Field").

Operation Field

The operation field is used to specify a machine instruction or assembly instruction. This field may start in any column to the right of the begin column, provided that at least one blank separates it from the last character of the name. The operation field may contain any valid mnemonic operation code. The valid machine-instruction mnemonics are listed in Section 3, "Machine Instruction Statements"; the valid assembly-instruction mnemonics are listed in Section 4, "Assembly Instructions." A valid machine-instruction or assembly-instruction mnemonic can not exceed five characters.

The following example shows the mnemonic code for the compare instruction (RR format) used in a statement named COMPARES:

NAME	Operation	Operand
COMPARES	CR	5,6

Operand Field

The contents of the operand field provide the assembly program with information about the instruction specified in the operation field. If a machine instruction has been specified, the operand field specifies such program elements as registers, storage addresses, immediate data, masks, and storage-area lengths. For an assembly instruction, the operand field conveys whatever information the assembly program requires for the particular instruction.

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

Depending on the instruction, the operand field may be composed of one or more subfields, called operands. Each operand must be separated from another by a comma. (Remember that a blank delimits the field; thus, blanks may not intervene between operands and commas.) The two operands in the following example specify general registers 5 and 6.

Name	Operation	Operand
COMPARES	CR	5,6

Comments Field

Comments are strictly for the convenience of the programmer. They permit lines 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 appear to the right of the operand field; at least one blank must separate the comments from the last operand. An entire card can be used for comments by placing an asterisk in the begin column. If multiple lines of comments are desired, they must be written as separate comments cards with an asterisk in the begin column. See the example below.

Name	Operation	Operand	72
*THE ASTERISK IN COLUMN 1 MAKES THIS A COMMENTS CARD *THE ASTERISK IS REQUIRED IN EACH COMMENTS CARD COMPARES CR 5,6 NO ASTERISK NEEDED			SHOULD BE BLANK

The programmer may use comments in instructions that do not require the operand field to be specified. In instructions where an optional operand field is omitted but a comments field is to be provided, the absence of the operand field must be indicated by a comma preceded and followed by one or more blanks. The next example illustrates this rule.

Name	Operation	Operand
	END	,THESE ARE COMMENTS

Identification-Sequence Field

The identification-sequence field is used for program identification and statement sequence numbers. This field occupies columns 73–80 of the input cards. If the field, or a portion of it, is used for program identification, the identification is punched in every statement card. The assembly program, however, does not normally check this field; it merely reproduces the information in the field on the output listing of the program.

If the identification-sequence field, or a portion of it, is used for statement sequence numbers, the numbers are punched in ascending sequence in successive input cards. By using the ISEQ assembly instruction, the programmer requests the assembly program to verify the ascending order of the numbers which he has punched. The ISEQ assembly instruction is described in Section 4 under "Program Control Instructions."

SECTION 2. WRITING ASSEMBLY LANGUAGE STATEMENTS

Assembly language statements are accepted by the assembly program only if they conform to the established grammatical rules and vocabulary restrictions that are presented in this section. Subsequent sections of this publication deal with the format and content of the specific types of assembly statements (i.e., machine instructions and assembly instructions). Both types of instructions are formed by using the basic elements described here. Many of the points introduced in this section will be amplified in subsequent sections.

CHARACTER SET

Assembly language stements may be written using the following characters:

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

The above characters are identified by the punch combinations listed in Appendix A.

Note: A blank is included in the set of permissible special characters, and subsequent references to special characters shall be understood to designate all of the specified special characters. References to numerical characters shall be understood to designate the decimal digits. References to hexadecimal digits shall be understood to designate the digits 0–9 and letters A–F. References to letters shall be understood to include the $ (except for a $ as the first character of a symbol).

SYMBOLS

Storage areas, instructions, registers, and other elements may be given symbolic names for the purpose of referring to them in the program. The symbolic name is called a symbol and may contain a maximum of eight characters. While the first character of a symbol must be a letter, the remainder may be letters, decimal digits, or any combination of the two. Special characters must not be used in symbols. Any violation of these rules will be noted with an error message in the program listing, and the symbol will not be used.

The following are valid symbols:

READER	**LOOP2**
A23456	**N**
X4F2	**S4**
AB$C	

The following symbols are invalid, for the reasons noted:

256B	**(First character is not alphabetic)**
RECORDAREA2	**(More than eight characters)**
BCD#34	**(Contains a special character)**
INPUT AREA	**(Contains a special character)**
$ABC	**(The $may be used within a symbol, but must not be the first character)**

Relocatable and Absolute Symbols

Most of the symbols in a program name a location (i.e., they represent the location of a specific data field or program instruction). Some symbols name an arbitrary value, one that may designate items such as a register, a length, or a displacement.

Subsequent portions of this publication speak of relocatable and absolute symbols. A relocatable symbol has a value that is tentative; that is, the value of the symbol may be subject to change if the program is loaded anywhere but at its assembled location.

Symbols can also be assigned arbitrary values other than relocatable addresses by use of the EQU, TEQU, MAX, and MIN assembly instructions (see Section 4). The arbitrary values can 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 absolute symbols that have a value that is fixed (i.e., the value is not related to the program's load location and is not subject to relocation). All other symbols are relocatable.

Defining Symbols

Symbols are meaningful when used as the names of statements or in statement operands. A symbol must be defined somewhere in the program in order to be used as an operand. It is defined when it is either used as the name of a statement or identified as an external symbol. A group of

special symbols, system symbols, has an implied definition and need not be defined in the program (see Table 7 in Appendix E).

When the assembly program encounters a symbol that names a statement, it analyzes the statement and develops a set of characteristics, or attributes, which it associates with the symbol. When the assembly program encounters the symbol employed as an operand, it is able to obtain from these attributes any required information about the statement that the symbol names. The attributes pertinent to the subsequent discussion of the assembly language are value, length, scale, and type.

VALUE ATTRIBUTE

Storage address of the leftmost byte of the field allocated to the statement named by the symbol. The value attribute of an absolute symbol is the value equated to the symbol.

LENGTH ATTRIBUTE

The number of bytes allocated to the statement named by the symbol. In the case of data definition statements, the length is the number of bytes allocated to the first element of the definition. The length is called the implied length of the symbol. The convenience of having implied length becomes apparent elsewhere in this publication, particularly in the discussion of the symbolic format of SS type machine instructions and in the discussion of the data definition instructions.

SCALE ATTRIBUTE

Certain data statements allow for a scaling modifier (see Section 4).

TYPE ATTRIBUTE

Each symbol receives a type code value which depends on how the symbol was defined. Table 2–1 lists these types.

Table 2–1. Type Attributes

0 – Hex (X)
1 – EBCDIC (C)
2 – Packed decimal (P)
3 – Zoned decimal (Z)

Table 2–1. Type Attributes (Continued)

4 – Fixed point halfword (H)
5 – Fixed point fullword (F)
6 – Floating point single (E)
7 – Floating point double (D)
8 – Instruction label (I)
9 – Address constant (A)
10 – Base displacement (S)

Previously Defined Symbols

Sometimes the programmer will be required to use a previously defined symbol in the operand field. Previously defined means that the symbol has been defined in a prior statement, i.e., earlier in the statement sequence. The following example shows how the symbol TEST is defined in the first statement and used in the second statement as a previously defined symbol.

Name	Operation	Operand
TEST	CR	5,6
LOOP	EQU	TEST

External and Entry-Point Symbols

Symbols are normally defined in the same program in which they are referred to (i.e., used as operands). Symbolic linkages between independently assembled programs may be effected by defining symbols in one program and using them as operands in another program. The programs that define and use the symbolic linkages may be assembled independently and then executed together.

The symbol that will serve as a linkage symbol must be so designated, both in the program where it is defined and in the program where it is used as an operand. It is termed an external symbol in the program that uses it as an operand and an entry-point symbol in the program that defines it. For example, if one program contains an instruction named STARTER, and another program uses STARTER as an operand, the first program must designate STARTER as an entry-point symbol, and the other program must designate it as an external symbol.

External and entry-point symbols are always relocatable. They are subject to certain usage restrictions that are discussed at pertinent places elsewhere in this publication. The value attribute of each entry-point symbol will be assigned by the loader to the corresponding external symbol when both programs are loaded.

The programmer must indicate to the assembly program which of the symbols are external and which are entry points. The EXTRN and ENTRY assembly instructions are provided for this purpose and are described in Section 4. There is one exception to the requirement that entry-point symbols be specified as such to the assembly program. A program name (defined in the name field of a START statement) is considered an entry point and may be used as one. However, the program name does not have to be identified as an entry point by the ENTRY instruction.

General Restrictions On Symbols

A symbol may be defined only once in an assembly; that is, each symbol used as the name of a statement must be unique to that assembly. However, the symbols used as names in TEQU assembly statements or as control section names (i.e., defined in CSECT or DSECT assembly statements) are excepted from this restriction. Because a control section may be terminated and then resumed at any subsequent point, the CSECT or DSECT statement that resumes the section must be named by the same symbol that initially name the section. Therefore, the symbol that names the section must be repeated. Such usage is not considered to be duplication of a symbol definition. Symbols in the name field of debug cards (DUMP, DUMPC, DUMPR) are not considered definitions.

If a symbol is used as a name more than once (other than the previously noted exceptions), only the first usage will be recognized; each subsequent usage of the symbol as a name will be ignored. Every usage of the name will be flagged in the program listing.

LOCATION COUNTER

A location counter is used by the assembly program to assign consecutive storage addresses to program statements. As each machine instruction or data area is assembled, the location counter is first adjusted to the proper boundary for the item, if adjustment is necessary, and then incremented by the length of the assembled item. If the statement is named by a symbol, the value attribute of the symbol is the value of the location counter after boundary adjustment, but before addition of the length. Such symbols are always relocatable.

The assembly program maintains a location counter for each control section of the program and manipulates each as previously described. Program statements for each section are assigned consecutively from the location counter for that section. The location counter for each successively declared control section assigns locations in consecutively higher areas of storage. Thus, if a program has multiple control sections, all statements identified as belonging to the first control section will be assigned from the location counter for section 1, the statements for the second control section will be assigned from the location counter for section 2, etc. This procedure is followed whether the statements from different control sections are interspersed or written in control section sequence.

The location counter setting can be controlled by using the START and ORG assembly instructions, which are described in Section 4. The counter affected by either of these assembly instructions is the counter for the control section in which the instruction is written. Certain other assembly instructions also affect the value of the location counter, and they are discussed in Section 4.

Location Counter References

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

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 assembly language provides this facility through the self-defining value, which can be a decimal, hexadecimal, or character representation, or an attribute of a symbol.

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 assembly 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 as a literal, its address is assembled into the instruction.

Decimal Self-Defining Values

A decimal value is an unsigned number of from one to eight decimal digits. High-order zeros may be used (e.g., 007). A decimal value is assembled as its binary equivalent. However, a decimal self-defining value of more than eight digits will not be used. Some examples of decimal self-defining values are: 8, 147, 4092, 00021, and 10201040.

Hexadecimal Self-Defining Value

A hexadecimal value is an unsigned number of from one to six hexadecimal digits. The digits must be enclosed by single quotation marks and preceded by the letter X. The hexadecimal digits represent the 16 possible combinations of 4 bits. Each hexadecimal digit is assembled as its four-bit value. Any hexadecimal self-defining value of more than six digits will not be used.

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'12FA'
X'F4F' X'00CD' X'00E0'

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

Character Self-Defining Value

A character value is a sequence of from one to three characters. The character sequence must be enclosed by single quotation marks, and it may be preceded by the letter C (this is not mandatory). All letters, decimal digits, and special characters may be used in a character value. In addition, any of the remaining 256 punch combinations listed in Appendix A may be designated in a character self-defining value. Examples of character self-defining values are:

C'/' C' ' (blank)
C'ABC' C'13'
C'X1' '*+6'

The representation of a quotation mark as a character is a special consideration, because the quotation mark is also the delimiter that indicates the beginning and end of the character sequence. Therefore, for each single quotation mark desired in a character value, two single quotation marks must be written. For example, the character value A'# would be written as 'A' '#', while a single quotation mark followed by two blanks would be written as ''' '.

Each character in the character sequence is assembled as its eight-bit code equivalent. The two quotation marks that must be used to represent a single quotation mark within the character sequence are assembled as a single quotation mark. Any decimal digits appearing in the character sequence will be assembled into their internal eight-bit representations.

Symbol Attribute Self-Defining Value

Every defined symbol has various attributes associated with it. Three of these attributes (length, type, and scale) may be used as a self-defining value by specifying the symbol preceded by an attribute specification. Hence, if BETA is defined as: BETA DC FL3S4' – 43', then L'BETA is 3, T'BETA is 5, and S'BETA is 4.

Using Self-Defining Values

The same value can be represented by any one of the three types of self-defining values. The selection of a particular type of representation is left to the user. Decimal representation, for example, might be desired for specifying register numbers; hexadecimal representations for specifying mask; and character representations for specifying immediate data is SI machine instructions. An attribute might be used as an increment.

LITERALS

Programs require a number of constants (actual data to be operated on, data for calculations or indexing, addresses of instructions, words and phrases for printed output, etc.). The assembly language provides the mechanism for defining data constants and address constants (and assigning symbols to them) through the DC assembly instruction. To designate the constant that he has defined, the user employs the symbol which names that constant as an operand in a machine instruction. However, this procedure of defining the constant may prove cumbersome for constants that will

not be used repeatedly. The user has the option of simultaneously defining and using a constant in the machine instruction where it is required. This is accomplished by using a literal as an operand in an instruction. (No other symbols or operators may appear in the operand field, other than a specified length, and only one literal is allowed per statement.)

Literals may be used wherever a relocatable symbol is permitted as an operand in machine or data description instructions only. Literals are considered relocatable, because the address of the literal, rather than the literal itself, will be assembled in the statement that employs a literal. The assembly program generates the literals, collects them, and places them in a specific area of storage, as explained in the subsection "Literal Pool." A literal is not to be confused with the immediate data in an SI instruction. Immediate data is assembled into the instruction.

Note: Throughout this publication, the term literal is used interchangeably to designate the literal operand and the constant that it provides. For example, the discussion of the literal pool indicates that "literals generated by the assembly program are collected. . .", meaning that the constants generated from literal operands are collected. The term literal is used customarily to indicate both the written format and the constant being specified.

Literal Format

Whether a constant is provided by a DC assembly instruction or as a literal, the assembly program requires a description of the type of constant being specified as well as the constant itself. This descriptive information assists the assembly program in assembling the constant correctly.

The method of describing and specifying a constant as a literal is identical to the method of providing it as the operand of a DC assembly instruction; that is, the literal is written as if it were the DC operand. The only difference is that the literal must start with an equal sign (=), which indicates to the assembly program that a literal follows. Refer to the discussion of the DC assembly instruction operand format (Section 4) for the means of specifying a literal and for examples of literals.

WARNING ON THE USE OF LITERALS

1. An asterisk should not appear in an address constant, because the location counter reference will be an address in the literal pool, not the address of the instruction containing the literal.
2. S-type address constants should not appear in literals.
3. No literals may occur within literals, for example:

```
valid:     DC     A(='ABC')
invalid:   DC     A[=A(='ABC')]
```

Literal Pool

The literals generated by the assembly program are collected and placed in a special area called the literal pool. As explained previously, the location of the literal in the pool, rather than the literal itself, is assembled in the statement employing a literal. The position of the literal pool may be controlled by the programmer.

The programmer may also specify that multiple literal pools be created. However, the sequence in which literals are ordered within the pool is controlled by the assembly program. Further information on positioning the literal pool(s) is in Section 4 under "LTORG – Begin Literal Pool."

If the programmer does not specify the location of the literal pool, all collected literals will be assembled at the end of the first control section.

EXPRESSIONS

Expressions are used to specify the various operands of machine-instruction and assembly-instruction statements. An expression consists of one term or some arithmetic combination of terms. The terms that may be used alone or in combination with each other are symbols, self-defining values, and references to the location counter. The arithmetic operators used to combine terms in an expression are:

+ addition	* multiplication
– subtraction	/ division

A literal may be used as a term but may not be combined with any other terms. Thus, an expression may contain only one term when that term is a literal. An expression may not contain two terms or two operators in succession. Also, the maximum number of terms permitted in an expression is five, and parentheses are not allowed.

Note: An expression is defined as consisting of one or more terms. In some instances, it is necessary to differentiate between expressions containing one term and expressions containing more than one term. In such cases, the

expressions are designated as single term or multiterm, respectively.

An expression may have all the attributes of a symbol except qualification. The length, type, and scaling attributes are those of the leftmost symbol or integer in the expression. The attributes length, type, and scale factor associated with a symbol may be used in an expression by preceding the symbol with L ', T ', or S '. For example, to refer to the length attribute associated with ALPHA, L 'ALPHA is written.

The following are examples of expressions:

*	FIELD2
AREA1+X '2D '	L 'BETA*10
*+32	EXIT–ENTRY+1
N–25	29

Note: The asterisk was used both as a multiplier (L'BETA*10), and as a reference to the location counter (* and *+32). The expression *+32 is an example of relative addressing, which is explained in this section.

The following examples whos violation of the rules that expressions may not contain two terms or two operators in succession; all are invalid:

AREAX 'C '	
FIELD+–	15RECORD/X '3

RELATIVE ADDRESSING

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

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

Evaluation of Expressions

The assembly program evaluates each expression in an operand field separately. The evaluation procedure is:

1. Each term is given its numerical value. The value of an expression is calculated using unsigned integer arithmetic modulo 2^{32}.

2. The arithmetic operations are performed moving from left to right. However, multiplication and/or division are performed before addition and subtraction. Thus A+B*C is evaluated as A+ (B*C), not (A+B) *C. Division by zero is defined to be equal to zero; remainders after division are dropped.

3. The computed result is considered the value of the expression.

If the expression has been used to specify a value for a symbol, the value of the expression is considered to be the value attribute of the symbol. Thus, an EQU assembly instruction may equate a symbol to an expression, in which case the value of the expression becomes the value attribute of the symbol. The length attribute of a single-term expression is the implied length of the term. For a multiterm expression, the length attribute is the implied length of the leftmost term in the expression.

Types of Expressions

The assembly language permits the use of absolute and relocatable expressions. Before the properties of these expressions can be defined, it is necessary to define the terms that may be used to form them. It has already been established that symbols, references to the location counter, and self-defining values are the terms used in expressions. These terms may be classified as:

Relocatable Terms	Absolute Terms
Relocatable symbols	Absolute symbols
Location counter	Self-defining values

Note: Literal terms are a special case and are discussed in the subsection "Relocatable Expressions."

ABSOLUTE EXPRESSIONS

An absolute expression may contain any combination of absolute terms. It may also contain relocatable terms,

alone or in combination with absolute terms, provided that the relocatable terms are used as follows:

1. There must be an even number of relocatable terms.
2. Each relocatable term must be paired with another relocatable term (with the opposite sign) from the same control section; that is, the effects of relocation must be nullified. Because a reference to the location counter is relocatable, it must be paired with another relocatable term in order to be used in an absolute expression.
3. No relocatable term may enter a multiply or divide operation; that is, no relocatable term may be preceded or followed by a multiplication or division sign.
4. Symbols that have a different relocation value (e.g., symbol in common and an EXTRN) must not appear in the same expression.

Although the address values of all relocatable terms are subject to change, there is a constant (i.e., absolute) difference between the values of each pair of terms from a control section. This being true, the entire expression consists of absolute terms and/or pairs of terms that represent absolute values.

The following examples illustrate absolute expressions, where A is any absolute symbol, and X and Y are relocatable symbols from the same control section.

Y+A–X	2048
Y–X	A*A

RELOCATABLE EXPRESSIONS

A relocatable expression is one whose value would change by n if the program were loaded n bytes away from its assembled location.

Relocatable expressions may contain relocatable terms alone or in combination with absolute terms. The relocatable terms must be used as follows:

1. There must be an odd number (1,3, or 5) of relocatable terms. The terms may be preceded by a plus or minus sign (a plus sign is assumed if a term is unsigned).
2. If there are three or five relocatable terms, each relocatable term except one must be paired with another relocatable term (with the opposite sign) from the same control section. Note that a location counter reference is a relocatable term.
3. The unpaired (odd) relocatable term may be preceded by a plus or minus sign. If a minus sign remains, the expression is negatively relocatable; otherwise, it is simply relocatable. However, negatively relocatable expressions may appear only in A-type address constants.
4. No relocatable term may enter into a multiply or divide operation; that is, no relocatable term may be preceded or followed by a multiply or divide sign.
5. All relocatable terms in an expression must be in the same control section or in the same DSECT.
6. An expression may not contain an external symbol and any other relocatable term.

Because each pair of terms from a control section represents an absolute value (i.e., there is a constant difference between them, regardless of relocation possibilities), a relocatable expression represents only one relocatable value. The rest of the values are absolute, being represented by absolute terms and/or pairs of relocatable terms.

Note: A literal is always a single-term relocatable expression; it must be used alone.

The following examples illustrate relocatable expressions, where A is an Absolute symbol, and X and Y are relocatable symbols from the same control section.

X+2	–X	**(Negatively relocatable)**
X–8*A	=X'4'	**(A literal containing the hexadecimal digit 4)**
X–Y+X	*+32	

External Symbols in Expressions

External symbols (i.e., those defined by the EXTRN assembly instruction) may be used in expressions. An expression containing an external symbol may not contain any other relocatable symbols. This restriction applies to all expressions.

External symbols may be used in USING statements, A-type address constants, field 2 of a CCW, and machine-instruction statements in expressions of the form:

External symbol + absolute terms

An external symbol may not appear in an expression with other relocatable symbols even though the result may be of the specified form.

SECTION 3. MACHINE INSTRUCTION STATEMENT

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

The symbolic format of a machine instruction parallels, but does not duplicate, its actual format. A mnemonic operation code is written in the operation field, and one or more operands are written in the operand field. Comments may be appended to a machine-instruction statement as previously explained in Section 1.

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

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

INSTRUCTION ALIGNMENT AND CHECKING

All machine instructions are aligned automatically by the assembly program on halfword boundaries. Bytes that are skipped because of alignment are set to zero. All expressions that specify storage addresses are checked to ensure that they refer to the proper boundaries (halfword, fullword, or doubleword) for the instructions in which they are used. If a base register and displacement are explicit, the displacement is checked. For statements with an implied base register and displacement, the effective address is checked. Register numbers are also checked to ensure that they specify the proper registers, as follows:

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

2. Double shift, fullword multiply, and fullword divide instructions must specify an even-numbered general register.

OPERAND FORMAT

Table 3–1 shows the symbolic operand formats for each variation of the basic machine-instruction formats. The table also shows the size, in bits, of each machine-instruction field (see the number at the top of each field) and it designates the instructions that use each operand format. All symbolic operand formats are indicated in the table as a series of codes (e.g., R1, S2, D2, and L). These codes are explained in the notes for Table 3–1. The notes provide other information pertinent to the use of symbolic operands and should be considered an integral part of the table.

Note: More than one operand format is shown for each variation of each machine-instruction format, except the RR type. The selection of an operand pattern depends on what is to be specified and whether it is to be specified explicitly or implicitly. This is discussed in the subsections on "Effective Addresses – – – Explicit and Implied" and "Lengths – – – Explicit and Implied."

Operand Fields and Subfields

Table 3–1 shows that some symbolic operands are written as a single field and other operands are written as a field followed by one or two subfields. For example, effective addresses consist of the contents of a base register and a displacement. A symbolic operand that specifies a base and displacement is written as a displacement field followed by a base register subfield. In the RX format, both an index register subfield and a base register subfield are written; in the SS format, both a length subfield and a base register subfield are written.

A comma must be written to separate operands. Parentheses must be written to enclose a subfield or subfields, and a comma must be written to separate two subfields within parentheses. When parentheses are used to enclose one subfield, and the subfield is omitted, the parentheses must be omitted. In the case of two subfields that are separated by a comma and enclosed by parentheses, the following rules apply:

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

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

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

All fields and subfields in a symbolic operand may be represented either by absolute or relocatable expressions, depending on what the field requires. (An expression consists of one term or a series of arithmetically combined terms.)

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

Table 3–1. Machine Instruction Format

	Basic Machine Format	Operand Field Format	Applicable Instructions
RR	8 Operation Code | 4 R1 | 4 R2	R1,R2	All RR instructions except DLY, LI, SPM, and SVC
	8 Operation Code | 4 R1 | 4 (R2)	R1	SPM, LI
	8 Operation Code | 8 I	I	SVC, DLY
RX	8 Operation Code | 4 R1 | 4 X2 | 4 B2 | 12 D2	R1, D2(X2,B2) R1, S2(X2)	All RX instructions
RS	8 Operation Code | 4 R1 | 4 R3 | 4 B2 | 12 D2	R1, R3, D2(B2) R1, R3, S2	BXH, BXLE, LM, and STM
	8 Operation Code | 4 R1 | 4 (R3) | 4 B2 | 12 D2	R1, D2(B2) R1, S2 R1, I2	All shift instructions
SI	8 Operation Code | 8 I2 | 4 B1 | 12 D1	D1(B1), I2 S1,I2	All SI instructions except LPSW, SSM, SPCI, HIO, SIO, TIO, TCH, LPSB and SPSB
	8 Operation Code | 8 (I2) | 4 B1 | 12 D1	D1, (B1) S1	LPSW, SSM, HIO, SIO, TIO, TCH, SPCI, LPSB, SPSB

Table 3–1. Machine Instruction Format (Continued)

	Basic Machine Format	Operand Field Format	Applicable Instructions
	8 Operation Code | 4 L1 | 4 L2 | 4 B1 | 12 D1 | 4 B2 | 12 D2	D1(L1, B1), D2(L2, B2) S1(L), S2(L2)	PACK, UNPK, MVO, AP, CP, DP, MP, SP, ZAP
SS	8 Operation Code | 8 L | 4 B1 | 12 D1 | 4 B2 | 12 D2	D1(L, B1), D2(B2) S1(L), S2	NC, OC, XC, CLC, MVC, MVN, MVZ, TR, TRT, ED, EDMK

Notes for Table 3–1

1. *R1, R2, and R3 are absolute expressions that specify general or floating-point registers. The general register numbers are 0 through 15; floating-point register numbers are 0, 2, 4, and 6.*
2. *D1 and D2 are absolute expressions that specify displacements. A value of 0–4095 may be specified. See the subsequent discussion on effective addresses and Table 3–2.*
3. *B1 and B2 are absolute expressions that specify base registers. Register numbers are 0–15. See the subsequent discussion on effective addresses and Table 3–2.*
4. *X2 is an absolute expression that specifies an index register. Register numbers are 0–15. If B2 is specified, X2 should not be omitted; and when indexing is not desired, X2 must be specified as 0. See Table 3–2.*
5. *L, L1, and L2 are absolute expressions that specify field lengths. An L expression can specify a value of 1–256. L1 and L2 expressions can specify a value of 1–16. In all cases, the assembled value will be one less than the specified value. See the subsequent discussion on explicit and implied lengths and Table 3–3. An L, L1, or L2 field of 0 is equivalent to 1.*
6. *I and I2 are absolute expressions that provide immediate data. The value of the expression may be 0–255.*
7. *S1 and S2 are absolute or relocatable expressions that specify an effective address. See the subsequent discussion on effective addresses and Table 3–2.*
8. *RR, RS, and SI instruction fields that are crossed out in the machine formats are not examined during instruction execution. The fields are not written in the symbolic operand, but are assembled as binary zeros.*

Effective Addresses – Explicit and Implied

An effective address is composed of a displacement plus the contents of a base register. (In the case of RX instructions, the contents of an index register are also used to derive the effective address.) The programmer writes an explicit effective address by specifying the displacement and the base register number. He may also write an implied effective address by specifying an absolute or relocatable address. The assembly program has the facility to select a base register and compute a displacement, thereby generating an explicit address from an implied address, provided that it has been informed (1) what base registers are available to it and (2) what each register contains. The programmer conveys this information to the assembly program through the USING and DROP assembly instructions which are described in Section 5 under "Base Register Instructions." Additional addressing considerations pertain when programs have more than one control section. These considerations also are discussed in "Base Register Instructions."

Table 3–1 shows two types of addressing formats for RX, RS, SI, and SS instructions. In each case, the first type shows the method of specifying an effective address explicitly, as a base register and displacement. The second type indicates how to specify an effective address implicitly, as an expression. For example, the load multiple instruction (RS format) may have either of the following symbolic operands:

R1, R3, D2(B2)	**Explicit address**
R1, R3, S2	**Implicit address**

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

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

1. The base register assembly instructions (USING and DROP) must be used as explained in Section 4.

2. An explicit base register must not accompany the implicit address.

For example, assume that FIELD is a relocatable symbol, which has been assigned a value of 7400. Assume also that the assembly program has been notified (by a USING instruction) that general register 12 currently contains a relocatable value of 4096 and is available as a base register. The following example shows a machine-instruction statement as it would be written in assembly language and as it would be assembled. Note that the value of D2 is the difference between 7400 and 4096 and that X2 is assembled as zero, because it was omitted. The assembled instruction is presented in decimal (op code is in hexadecimal).

Assembly language statement:

ST 4,FIELD

Assembled instruction:

Op Code	R1	X2	B2	D2
50	4	0	12	3304

To summarize, an effective address may be specified explicitly as a base register and displacement (and index register for RX instructions) by the formats shown in the first column of Table 3–2. The address may be specified implicitly by the formats shown in the second column. Table 3–2 is an elaboration of the operand information summarized in notes 2, 3, 4, 7, and 8 for Table 3–1. Note that the two storage addresses required by the SS instructions are presented separately; an implicit address may be used for one while an explicit address is used for the other.

Table 3–2. Details of Address Specification

Type	Explicit Effective Address	Implicit Effective Address
RX	D2(X2,B2)	S2(X2)
	D2(0,B2)*	S2
RS	D2(B2)	S2

Table 3–2. Details of Address Specification (Continued)

Type	Explicit Effective Address	Implicit Effective Address
SI	D1(B1)	S1
SS	D1(L1,B1)	S1(L1)
	D1(L,B1)	S1(L)
	D2(L2,B2)	S2(L2)
	D1(B1)	S1
	D2(B2)	S2

*A zero must be supplied when it is desired to omit an index register specification in an RX explicit address, or the base register will be interpreted as the index register. Note that this will not cause the instruction to execute differently.

Lengths – Explicit and Implied

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

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

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

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

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

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

Although length fields are usually in terms of bytes, the MVW instruction is an exception to this, i.e., its length field specifies words.

To summarize, the length required in an SS instruction may be specified explicitly by the formats shown in the first column of Table 3–3 or may be implied by the formats shown in the second column. This table is an elaboration of the operand information summarized in note 5 for Table 3–1. Note that the two lengths required in one of the SS instruction formats are presented separately; an implied length may be used for one while an explicit length is used for the other.

Table 3–3. Details of Length Specification in SS Instruction

Explicit Length	Implicit Length
D1(L1,B1)	D1(,B1) (Implicit Length of 1)
S1(L1)	S1
D1(L,B1)	D1(,B1) (Implicit Length of 1)
S1(L)	S1
D2(L2,B2)	D2(,B2)
S2(L2)	S2

MACHINE INSTRUCTION MNEMONIC CODES

This subsection contains an alphabetical listing of the mnemonic operation codes of all the machine instructions that can be represented in the assembly language. The column headings in Table 3–4 and the information each column provides are:

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

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

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

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

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

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

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

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

Mnemonic codes for functions involving data usually indicate the data types by letters that correspond to those for the data types in the DC assembly instruction (see Section 4). Furthermore, letters U and W have been added to indicate short and long, unnormalized floating-point operations, respectively. For example, AE indicates Add Normalized Short, whereas AU indicates Add Unnormalized Short. Where applicable, fullword fixed-point data is implied if the data type is omitted.

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

Machine Instruction Examples

The examples that follow are grouped according to machine-instruction format. They illustrate the various symbolic operand formats, which are summarized in Table 3–1 and detailed in Tables 3–2 and 3–3. 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 to absolute values elsewhere.

Implied addressing, control section addressing, and the function of the USING assembly instruction are not considered here. For discussion of these considerations and for examples of coding sequences that illustrate them, refer to Section 4, "Program Sectioning and Linking Instructions" and "Base Register Instructions."

RR FORMAT

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

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

Table 3–4. Machine Instruction 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 Halfword	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	R1,D2(X2,B2)
BCR	Branch on Condition	07	RR	R1,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 Halfword	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
CSS	Convert and Sort Symbols	02	RR	R1,R2
CVB	Convert to Binary	4F	RX	R1,D2(X2,B2)
CVD	Convert to Decimal	4E	RX	R1,D2(X2,B2)
CVWL	Convert Weather Lines	03	RR	R1,R2
D	Divide	5D	RX	R1,D2(X2,B2)
DD	Divide, Long	6D	RX	R1,D2(X2,B2)
DDR	Divide, Long	2D	RR	R1,R2
DE	Divide, Short	7D	RX	R1,D2(X2,B2)
DER	Divide, Short	3D	RR	R1,R2
DIAG	Diagnose	83	SI	DI(B1),I2
DLY	Delay	0B	RR	I
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)

Table 3–4. Machine Instruction Mnemonic Codes (Continued)

Mnemonic Code	Instruction	Operation Code	Basic Machine Format	Operand Field Format
EDMK	Edit and Mark	DF	SS	D1(L1,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 Input/Output	9E	SI	D1(B1)
IATR	Insert Address Translator	0E	RR	R1,R2
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)
LC	Load Chain	52	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)
LDA	Load Data Address	99	RS	R1,D2(B2)
LDR	Load, Long	28	RR	R1,R2
LE	Load, Short	78	RX	R1,D2(X2,B2)
LER	Load, Short	38	RR	R1,R2
LH	Load Halfword	48	RX	R1,D2(X2,B2)
LI	Load Identity	0C	RR	R1
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
LPSB	Load Preferential Storage Base	A1	SI	D1(B1)
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 Halfword	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)
MVW	Move Word	D8	SS	D1,(L,B1),D2(B2)

Table 3–4. Machine Instruction Mnemonic Codes (Continued)

Mnemonic Code	Instruction	Operation Code	Basic Machine Format	Operand Field Format
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(L,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
RPSB	Repack Symbols	0F	RR	R1,R2
S	Subtract	5B	RX	R1,D2(X2,B2)
SATR	Set Address Translator	0D	RR	R1,R2
SCON	Set Configuration	01	RR	R1,R2
SD	Subtract Normalized, Long	6B	RX	R1,D2(X2,B2)
SDR	Subtract Normalized, Long	2B	RR	R1,R2
SE	Subtract Normalized, Short	7B	RX	R1,D2(X2,B2)
SER	Subtract Normalized, Short	3B	RR	R1,R2
SH	Subtract Halfword	4B	RX	R1,D2(X2,B2)
SIO	Start Input/Output	9C	SI	D1(B1)
SIOP	Start I/O Processor	9A	SI	D1(B1),I2
SL	Subtract Logical	5F	RX	R1,D2(X2,B2)
SLA	Shift Left Short Algebraic	8B	RS	R1,D2(B2)
SLDA	Shift Left Long Algebraic	8F	RS	R1,D2(B2)
SLDL	Shift Left Double Logical	8D	RS	R1,D2(B2)
SLL	Shift Left Short Logical	89	RS	R1,D2(B2)
SLR	Subtract Logical	1F	RR	R1,R2
SP	Subtract Decimal	FB	SS	D1(L1,B1),D2(L2,B2)
SPCI	Set PCI	9B	SI	D1(B1)
SPM	Set Program Mask	04	RR	R1
SPSB	Store Preferential Storage Base	A0	SI	D1(B1)
SR	Subtract	1B	RR	R1,R2
SRA	Shift Right Short Algebraic	8A	RS	R1,D2(B2)
SRDA	Shift Right Long Algebraic	8E	RS	R1,D2(B2)
SRDL	Shift Right Long Logical	8C	RS	R1,D2(B2)
SRL	Shift Right Short Logical	88	RS	R1,D2,(B2)
SSK	Set Storage Key	08	RR	R1,R2
SSM	Set System Mask	80	SI	D1(B1)
ST	Store	50	RX	R1,D2(X2,B2)
STC	Store Character	42	RX	R1,D2(X2,B2)
STD	Store Long	60	RX	R1,D2(X2,B2)
STE	Store Short	70	RX	R1,D2(X2,B2)
STH	Store Halfword	40	RX	R1,D2(X2,B2)
STM	Store Multiple	90	RS	R1,R3,D2(B2)
SU	Subtract Unnormalized, Short	7F	RX	R1,D2(X2,B2)
SUR	Subtract Unnormalized, Short	3F	RR	R1,R2
SVC	Supervisor Call	0A	RR	I

Table 3–4. Machine Instruction Mnemonic Codes (Continued)

Mnemonic Code	Instruction	Operation Code	Basic Machine Format	Operand Field Format
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 Input/Output	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 OR	17	RR	R1,R2
ZAP	Zero and Add Decimal	F8	SS	D1(L1,B1),D2(L2,B2)

RX FORMAT

Name	Operation	Operand
ALPHA1	L	1, 39(4,10)
ALPHA2	L	REG1, 39(4,TEN)
BETA1	L	2, ZETA(4)
BETA2	L	REG2,ZETA(REG4)
GAMMA1	L	2,ZETA
GAMMA2	L	REG2, ZETA
GAMMA3	L	2, =F '1000'

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. The second operand of GAMMA3 is a literal.

RS FORMAT

Name	Operation	Operand
ALPHA1	BXH	1,2,20(14)
ALPHA2	BXH	REG1,REG2,20(REGD)
ALPHA3	BXH	REG1,REG2,ZETA
BETA1	SLL	1,20(9)
BETA2	SLL	REG1,20
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–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	FIELD1(6),FIELD2(9)
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.

Extended Mnemonic Codes

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

The allowable extended mnemonic codes and their operand formats are shown in the following list, together with their machine instruction equivalents. Unless the instruction ends with an 'R', the extended mnemonics shown are for instructions in the RX format. Note that the only difference between the operand fields of the extended mnemonics and those of their machine instruction equivalents is the absence of the R1 field and the comma that separates it from the rest of the operand field. The extended mnemonic list, like the machine instruction list, shows explicit address formats only. Each address can be specified as an implicit address, as explained previously under "Effective Address — — — Explicit and Implied" and as summarized in Tables 3–1 and 3–2. Examples that illustrate instructions using extended mnemonic codes follow the list of extended mnemonics in Table 3–5.

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

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

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

Table 3–5. Extended Mnemonics

Extended Code		Meaning	Machine Instruction	
B	D2(X2,B2)	Branch Unconditional	BC	15,D2(X2,B2)
BR	R2	Branch Unconditional (RR Format)	BCR	15,R2
NOP	D2(X2,B2)	No Operation	BC	0,D2(X2,B2)
NOPR	R2	No Operation (RR Format)	BCR	0,R2

Table 3–5. Extended Mnemonics (Continued)

Extended Code		Meaning	Machine Instruction	
		Used After Compare Instructions		
BH	D2(X2,B2)	Branch on High	BC	2,D2(X2,B2)
BHR	R2	Branch on High (RR Format)	BCR	2,R2
BHE	D2(X2,B2)	Branch on a High or Equal	BC	10,D2(X2,B2)
BHER	R2	Branch on a High or Equal (RR Format)	BCR	10,R2
BL	D2(X2,B2)	Branch on Low	BC	4,D2(X2,B2)
BLR	R2	Branch on Low (RR Format)	BCR	4,R2
BLE	D2(X2,B2)	Branch on Low or Equal	BC	12,D2(X2,B2)
BLER	R2	Branch on Low or Equal (RR Format)	BCR	12,R2
BE	D2(X2,B2)	Branch on Equal	BC	8,D2(X2,B2)
BER	R2	Branch on Equal (RR Format)	BCR	8,R2
BNH	D2(X2,B2)	Branch on Not High	BC	13,D2(X2,B2)
BNHR	R2	Branch on Not High (RR Format)	BCR	13,R2
BNL	D2(X2,B2)	Branch on Not Low	BC	11,D2(X2,B2)
BNLR	R2	Branch on Not Low (RR Format)	BCR	11,R2
BNE	D2(X2,B2)	Branch on Not Equal	BC	7,D2(X2,B2)
BNER	R2	Branch on Not Equal (RR Format)	BCR	7,R2
		Used After Arithmetic Instructions		
BO	D2(X2,B2)	Branch on Overflow	BC	1,D2(X2,B2)
BOR	R2	Branch on Overflow (RR Format)	BCR	1,R2
BV	D2(X2,B2)	Branch on Overflow	BC	1,D2(X2,B2)
BVR	R2	Branch on Overflow (RR Format)	BCR	1,R2
BP	D2(X2,B2)	Branch on Plus	BC	2,D2(X2,B2)
BPR	R2	Branch on Plus (RR Format)	BCR	2,R2
BM	D2(X2,B2)	Branch on Minus	BC	4,D2(X2,B2)
BMR	R2	Branch on Minus (RR Format)	BCR	4,R2
BNM	D2(X2,B2)	Branch on Not Minus	BC	11,D2(X2,B2)
BNMR	R2	Branch on Not Minus (RR Format)	BCR	11,R2
BNP	D2(X2,B2)	Branch on Not Plus	BC	13,D2(X2,B2)
BNPR	R2	Branch on Not Plus (RR Format)	BCR	13,R2
BNZ	D2(X2,B2)	Branch on Not Zero	BC	7,D2(X2,B2)
BNZR	R2	Branch on Not Zero (RR Format)	BCR	7,R2
BZ	D2(X2,B2)	Branch on Zero	BC	8,D2(X2,B2)
BZR	R2	Branch on Zero (RR Format)	BCR	8,R2
		Used After Test Under Mask Instruction		
BALL	D2(X2,B2)	Branch if all Ones	BC	1,D2(X2,B2)
BALLR	R2	Branch if all Ones (RR Format)	BCR	1,R2
BSOM	D2(X2,B2)	Branch if some Ones	BC	4,D2(X2,B2)
BSOMR	R2	Branch if some Ones (RR Format)	BCR	4,R2
BNON	D2(X2,B2)	Branch if no Ones	BC	8,D2(X2,B2)
BNONR	R2	Branch if no Ones (RR Format)	BCR	8,R2
BNO	D2(X2,B2)	Branch if not all Ones	BC	14,D2(X2,B2)
BNOR	R2	Branch if not all Ones (RR Format)	BCR	14,R2

SECTION 4. ASSEMBLY INSTRUCTION STATEMENTS

Table 4–1 contains all of the assembly instructions, listed according to mnemonic operation code and name. They are fully described in this section and summarized in Table D–1 in Appendix D.

Table 4–1. Assembly Instructions

Symbol Definition Instructions

EQU	Equate Symbol
MAX	Equate Symbol (maximum value)
MIN	Equate Symbol (minimum value)
TEQU	Temporarily Equate a Symbol

Data Definition Instructions

DC	Define Constant
DS	Define Storage
CCW	Define Channel Command Word

Program Sectioning and Linking Instructions

START	Start Program
CSECT	Identify Control Section
DSECT	Identify Dummy Section
COM	Identify Common Control Section
ENTRY	Identify Entry-Point Symbol
EXTRN	Identify External Symbol
QUAL	Qualify Name

Base Register Instructions

USING	Use Base Address Register
DROP	Drop Base Address Register

Listing Control Instructions

TITLE	Identify Assembly Output
EJECT	Start New Page
SPACE	Space Listing
PRINT	Print Optional Data
NLIST	Suppress Listing
LIST	Resume Listing
DOUBL	Double Space Listing

Table 4–1. Assembly Instructions (Continued)

Program Control Instructions

ICTL	Input Control
ISEQ	Input Sequence Checking
ORG	Reset Location Counter
LTORG	Begin Literal Pool
CNOP	Conditional No Operation
END	End Program
SPEM	Suppress Possible Error Messages
RPEM	Restore Printing Error Messages
LIB	Library
SSEQ	Suppress Input Sequence Check
PSEG	Get Compool Segment

Debug Instructions

DUMP	Dump Definition
DUMPE	Emergency Dump Definition
DUMPC	Conditional Dump Definition
DUMPR	Register Dump Definition
TDMPL	Logical Record Tape Dump Definition
TDMPP	Physical Record Tape Dump Definition
TRACB	Branch Trace Definition
TRACE	Trace Definition

SYMBOL DEFINITION INSTRUCTIONS

EQU – Equate Symbol

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

Name	Operation	Operand
A symbol	EQU	An expression

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

The symbol in the name field is given the same attributes as the expression in the operand field. The length attribute (as well as the type and scale) of the symbol is that of the leftmost (or only) term of the expression. If the expression is an asterisk or a self-defining value, the implied length of the expression is one. The value attribute of the symbol is the value of the expression.

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

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

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

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

Name	Operation	Operand
FIELD	EQU	ALPHA–BETA+GAMMA

FIELD is defined as ALPHA–BETA+GAMMA and may be used in place of it. Note, however, that ALPHA, BETA, and GAMMA must all be previously defined.

Symbols can also be equated to other symbols to give the same attributes to different symbols used in different parts of the program.

MAX and MIN – Equate Symbol

The assembly instructions MAX and MIN perform a function similar to that performed by the EQU instruction but, in addition, they choose an expression from two or more expressions in the operand field. (A MAX or MIN statement with one expression in the operand field is permissible and is treated as an EQU statement.) The choice of expressions is either the highest value or location (MAX) or the lowest value or location (MIN). The format of a MAX or MIN statement is:

Name	Operation	Operand
A symbol	MAX or MIN	Expression 1, Expression 2, . . . , Expression n

The same checking that is performed for EQU statements is done for MAX and MIN statements and for each expression in the operand fields of these statements. In addition, all expressions must have the same relocatability attribute. If any of these conditions is not met, a message is printed; and the MAX or MIN statement is ignored.

If two or more expressions in the operand field of a single MAX or MIN statement have the same location or value attribute, then the leftmost, or first occurring, expression is the one chosen to supply the length, type, and scaling attributes for the defined symbol. The ordering of expression values is such that zero (if absolute) or the first byte of the control section or of the external item (if relocatable) is always a minimum value. The maximum value of an absolute expression is $2^{24}-1$; the maximum location for a relocatable expression is: beginning of control section + $2^{24}-1$. Thus, if A is the name of the first byte in a control section, then:

MAX	A–50,A–1,A,A+50	is A–1
MIN	A–53,A–1,A,A+50	is A

These unexpected results occur because of the uncertainties of relocation and the modulo arithmetic used is the fixed-point operations and in effective address calculation by the CPU. Meaningful results can be obtained by comparing absolute values or addresses within one control section only. As soon as a value drops below the value of the first byte of the control section, it is treated as a very large number.

TEQU – Temporarily Equate a Symbol

The assembly instruction TEQU is similar to an EQU instruction except that in a TEQU statement the symbol in the name field may be previously defined; whereas, in an EQU statement, the symbol in the name field must not be previously defined. The format of the TEQU statement is:

Name	Operation	Operand
ALPHA	TEQU	BETA

where ALPHA may or may not be previously defined and BETA must be previously defined. The attributes of BETA are assigned to ALPHA by the assembly program.

Any symbol may be temporarily equated. However, only another TEQU statement may be used to redefine that symbol. Therefore, to redefine ALPHA the following must be used:

```
ALPHA     TEQU     GAMMA
```

If a symbol is defined via anything but an EQU or TEQU and redefined by a TEQU, the assembled symbol will be given the value assigned by the TEQU. Equated symbols which are TEQUated will assume the value of the last TEQU in the assembly until redefinition by the EQU is encountered. The following example illustrates the assembly of TEQUated symbols:

```
Assembled
 Value          Code                          Comment
                 .
                 .
                 .
0000 0014        DC     A(SYMBOL)     Symbol not previously defined.
                                       Symbol receives value from the
                                       last TEQU in the assembly.
         SYMBOL  EQU    1
0000 0001        DC     A(SYMBOL)     Equate re-evaluated.
         SYMBOL  TEQU   2
0000 0002        DC     A(SYMBOL)
         SYMBOL  TEQU   20
0000 0014        DC     A(SYMBOL)
```

DATA DEFINITION INSTRUCTIONS

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

These statements are used to enter data constants into storage, to define and reserve areas of storage, and to specify the contents of channel command words. The statements may be named by symbols so that other program statements can refer to the fields generated from them.

DC – Define Constant

The DC instruction generates constant data in storage. A variety of constants may be specified: fixed-point, floating-point, decimal, hexadecimal, character, and storage addresses. (Data constants are generally called constants unless they are created from storage addresses, in which case they are called address constants.) The format of the DC statement is:

Name	Operation	Operand
A symbol or blank	DC	A single operand describing the constant, written in the format in the following text

An operand consists of four subfields; the first three subfields describe the constant (some or all may be omitted, depending on the constant), and the fourth subfield provides the constant or constants. Note that more than one constant may be specified in the fourth subfield for most types of constants. Each constant so specified must be of the same type; the descriptive subfields that precede the constants apply to all of them. No blanks may occur within any of the subfields (unless provided as characters in a character constant), nor may they occur between the subfields of an operand.

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

1	2	3	4
Duplication Factor	Type	Modifiers	Value List (Constants)

The symbol that names the DC instruction is the name of the constant (or first constant if the instruction specifies more than one). Relative addressing (e.g., SYMBOL+2) may be used to address the various constants if more than one has been specified, because the number of bytes allocated to each constant can be determined. This length consideration is discussed in various subsections of this publication and summarized in Table C–1 in Appendix C.

The value attribute (address) of the symbol that names the DC instruction is the address of the leftmost byte (after alignment) of the first, or only, constant. The length attribute depends on two things: the type of constant being defined and the presence of a length specification. Implied lengths are assumed for the various constant types in the absence of a length specification. If more than one constant is defined, the length attribute is the length (specified or implied) of the first constant.

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

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

LITERAL DEFINITIONS

Note that the discussion of literals as machine-instruction operands (in Section 2) refers to the description of the DC operand for the method of writing a literal operand. All subsequent operand specifications are applicable to writing literals, the only difference being that the literal is preceded by an equal sign. Examples of literals appear throughout the balance of the DC instruction text.

OPERAND SUBFIELD 1: DUPLICATION FACTOR

The duplication factor (multiplicity) may be omitted. If specified, it causes the constant(s) to be generated the number of times indicated by the factor. The factor may be specified either by an unsigned decimal value or by an absolute expression that is enclosed by parentheses. The value of the expression is interpreted as positive and modulo 2^{24}. All symbols in the expression must be previously defined. The multiplicity value may range from 0 to any value that does not cause the location counter to exceed $2^{24}-1$. If no multiplicity is specified, 1 is assumed. Multiplicity must not be greater than 1 for A-type or S-type address constants.

Thus, specifying the constant 30 with a duplication factor of 2 results in the generation of 3030. However, specifying the constants 30, 60, and 90 with a duplication

of 2 results in the generation of 306090306090. The duplication factor is always applied after the constant is fully assembled, i.e., after it has been developed into its proper format.

Note that a duplication factor of zero is permitted and will achieve the same result as it would in a DS instruction. See " Forcing Alignment " under " DS – Define Storage."

OPERAND SUBFIELD 2: TYPE

The type subfield defines the type of constant being specified. From the type specification, the assembly program determines how it is to interpret the constant and translate it into the appropriate machine format. The type is specified by a single-letter code as shown in Table 4–2.

Table 4–2. Type Subfield Codes

Code	Type of Constant	Machine Format
	Arithmetic	
F	Fixed-point	Signed, fixed-point binary format; normally a fullword.
H	Fixed-point	Signed, fixed-point binary format; normally a halfword.
E	Floating-point	Short floating-point format; normally a fullword.
D	Floating-point	Long floating-point format; normally a doubleword.
P	Decimal	Packed decimal format.
Z	Decimal	Zoned decimal format.
	Variable Field Length	
C	Character	Eight-bit code for each character.
X	Hexadecimal	Four-bit code for each hexadecimal digit.

Table 4–2. Type Subfield Codes (Continued)

Code	Type of Constant	Machine Format
	Address	
A	Address	Value of address; normally a fullword.
S	Address	S-address (base and displacement) constant; a halfword.

Note: The type subfield may be omitted where the constant value list is delimited by quote marks. In this case, type C is assumed. Further information about these constants is provided under " Operand Subfield 4: Value List."

OPERAND SUBFIELD 3: MODIFIERS

Modifiers are coded information about the explicit length (in bytes) desired for a constant and the scaling for the constant. If multiple modifiers are written, they must appear in this sequence: length and scale. Each modifier is written and used as described in the following text.

Length Modifier

The length modifier (L) specifies the amount of storage into which each data item is to be assembled. When a length modifier is given with a data component, no word boundary resolution is done. The length specification may take one of three forms:

1. Le. or Le where e is an unsigned integer or an expression contained within parentheses. (Any symbols in these expressions must have been defined by a previous statement in the source program.) The value of e, which is interpreted as positive and modulo 2^{24}, denotes the number of bytes of storage which will contain the assembled constant. Limits on the maximum value of e for each data type are shown in Table C–1 in Appendix C.

2. L.e where e is of the same form as specified in item 1. Here, the value of e gives the number of bits of storage which will contain the assembled constant. The value of e may exceed 8 and it will be understood to mean an integral number of bytes

plus so many bits. Thus, L.16 is equivalent to L2 and means 16 bits or 2 bytes. If a length modifier (other than type 1) is given, all succeeding items will be assembled in the next available bit location. If no bit length is given, proper boundary resolution (at least to the byte level) will occur.

3. Le1.e2 where e1 and e2 are of the same form as e defined in item 1. The value of e1 denotes the number of bytes, and e2 denotes the number of bits. After e2 is converted to an integral number of bytes plus so many bits as defined in item 2, the value of e1 is added to the number of bytes. The resulting value gives the amount of storage into which the assembled constant is placed. Thus, L1.18 is equivalent to L3.2 and means 3 bytes plus 2 bits.

When one or more bit lengths are given, which do not leave the location counter on a byte boundary, the remaining bits in the byte are set to zero; the location counter is set to the next full-byte boundary before the next statement is translated. This means that no name may have a bit address. The implied length of a data item which has a bit length is rounded up to the next integral byte. Table C–1 in Appendix C shows the maximum length values allowed for each data type. After a bit length has been rounded upward to the next byte length, the result must not exceed those values shown in Table C–1.

Scale Modifier

This modifier is written as Sn, where n is either a decimal value or an absolute expression enclosed by parentheses. Any symbol in the expression must be previously defined. The decimal value or the parenthesized expression may be preceded by a sign; if no sign is present, a plus sign is assumed. The maximum values for scale modifiers are summarized in Table C–1.

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

Fixed-Point Constant. The scale modifier specifies a number (2^n) by which the constant must be multiplied after it has been converted to its binary representation. Just as multiplication of a decimal number by 10^n causes the decimal point to move, multiplication of a binary number by 2^n causes the binary point to move. This multiplication has the effect of moving the binary point away from its assumed position in the binary field; the assumed position is to the right of the rightmost position.

Thus, the scale modifier indicates either of the following: (1) the number of binary positions to be occupied by the fractional portion of the binary number, or (2) the number of binary positions to be deleted from the integral portion of the binary number. A positive scale of x shifts the integral portion of the number x binary positions to the left, thereby reserving the rightmost x binary positions for the fractional portion. A negative scale shifts the integral portion of the number right, thereby deleting rightmost integral positions.

Notes: If a scale modifier does not accompany a fixed-point constant containing a fractional part, the fractional part is lost.

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

Floating-Point Constant. Only a positive scale modifier may be used with a floating-point constant. It indicates the number of hexadecimal positions that the fraction is to be shifted to the right. Note that this shift amount is in terms of hexadecimal positions, each of which is four binary positions. (A positive scaling actually indicates that the point is to be moved to the left. However, a floating-point constant is always converted to a fraction, which is hexadecimally normalized. The point is assumed to be at the left of the leftmost position in the field. Because the point cannot be moved left, the fraction is shifted right.)

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

OPERAND SUBFIELD 4: VALUE LIST

This subfield supplies the constant (or constants) described by the subfields that precede it. A data constant (all types except A and S) is enclosed by single quotation marks. An address constant (types A and S) is enclosed by parentheses. To specify two or more constants in the subfield, the constants must be separated by commas and

the entire sequence of constants must be enclosed by the appropriate delimiters (i.e., single quotation marks or parentheses). Thus, the format for specifying the constant(s) is one of the following:

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

All constant types except character (C) and hexadecimal (X) will be aligned on the proper word boundary, as shown in Table C–1 unless a length modifier is specified. In the the presence of a length modifier, no boundary alignment is performed. If an operand specifies more than one constant, any necessary alignment applies only to the first constant. Thus, for an operand that provides five fullword constants, the first constant would be aligned on a fullword boundary, and the remaining constants would automatically fall on fullword boundaries.

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

If an address constant contains a location counter reference, the location counter value that is used is the storage address of the first byte that the constant will occupy. Thus, if several address constants in the same instruction refer to the location counter, the value of the counter varies from constant to constant. However, if a single constant is specified (and it is a location counter reference) with a duplication factor, the constant is duplicated with the same location counter value.

The following text describes each of the constant types and provides examples. The description concludes with Table C–1, which summarizes the information on all aspects of constant definition, e.g., maximum lengths, exponent limits, boundary alignment, etc.

Character Constant – C

All letters, decimal digits, and special characters may be used in character constants. In addition, any of the remaining 256 punch combinations listed in Appendix A may be designated in a character constant. Only one character constant may be specified per operand. Since multiple constants within an operand are separated by commas, an attempt to specify two character constants would result in interpreting the comma that separates them as a character.

Special consideration must be given to representing the quotation mark as a character. Each single quotation mark desired as a character in the constant must be represented by a pair of single quotation marks. Only one single quotation mark will be generated for each pair.

The maximum length of a character constant is 255 bytes. No word boundary alignment is performed. Each character is translated into one byte. If no length modifier is given, the size in bytes of the character constant is equal to the number of characters in the constant. If a length modifier is provided, the result varies as follows:

1. If the number of characters in the constant exceeds the specified length, as many rightmost bytes as necessary are dropped.
2. If the number of characters is less than the specified length, the excess rightmost bytes are filled with blanks.

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

Name	Operation	Operand
FIELD	DC	C'TOTAL IS $10'

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

Name	Operation	Operand
FIELD	DC	CL15'TOTAL IS $10'

Note that in the next example, a length of 4 has been specified, but there are five characters in the constant:

Name	Operation	Operand
FIELD	DC	3CL4'ABCDE'

The generated constant would be: ABCDABCDABCD. On the other hand, if the length had been specified as 6 instead of 4, the generated constant would have been:

ABCDEbABCDEbABCDEb

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

*Not permitted for character and hexadecimal constants.

Name	Operation	Operand
	MVC	CODEAREA,=3CL4'ABCDE'

Hexadecimal Constant – X

A hexadecimal constant is comprised of one or more of the hexadecimal digits, which are 0–9 and A–F. Only one hexadecimal constant may be specified per operand. The maximum length of a hexadecimal constant is 255 bytes. No word boundary alignment is performed.

Constants that contain an even number of hexadecimal digits are translated as one byte per pair of digits. If an odd number of digits is specified, the leftmost four bits of the leftmost byte are filled with a hexadecimal zero, while the rightmost four bits contain the odd (first) digit.

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

1. If the number of hexadecimal digit pairs exceeds the specified length, the necessary leftmost bits (and/or bytes) are dropped.

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

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

Name	Operation	Operand
TEST	DC	X'FF00FF00'

The next example uses the same constant as a literal and sets the bit pattern of a register:

Name	Operation	Operand
	L	5,=X'FF00FF00'

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

Name	Operation	Operand
ALPHACON	DC	3XL2'A6F4E'

The resulting constant would be 6F4E, which would occupy the specified two bytes. It would then be duplicated three times, as requested by the duplication factor. If it had merely been specified as X'A6F4E', the resulting constant would have had a hexadecimal zero in the leftmost position: 0A6F4E.

Fixed-Point Constants – F and H

A fixed-point constant is written as a decimal number (not exceeding 23 digits) which may be followed by a decimal exponent. The number may be an integer, a fraction, or a mixed number (i.e., one with integral and fractional portions). The format of the constant is:

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

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

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

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

If the rightmost portion of the number must be dropped as a result of scale modifiers, rounding occurs. Any duplication factor that is present is applied after the constant is converted to its binary format and assembled into the proper number of bytes.

A field of three fullwords is generated by the following statement. The value attribute of CONWRD is the address of the leftmost byte of the first word; and the length attribute is 4, the implied length for a fullword fixed-point constant. The expression CONWRD+4 could be used to address the second constant (second word) in the field.

Name	Operation	Operand
CONWRD	DC	3F'658474'

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

Name	Operation	Operand
HALFCON	DC	HS6'–25.93'

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

Name	Operation	Operand
FULLCON	DC	HS8'3.50E–2'

The same constant could be specified as a literal:

Name	Operation	Operand
	AH	7,=HS8'3.50E–2'

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

Name	Operation	Operand
THREECON	DC	FS4'10.25.3.100'

Floating-Point Constants – E and D

A floating-point constant is written as a decimal number (not exceeding 23 digits), which may be followed by a decimal exponent, if desired. The number may be an integer, a fraction, or a mixed number (i.e., one with integral and fractional portions). The format of the constant is:

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

2. The exponent is optional. If specified, it is written immediately after the number as En, where n is an optionally signed decimal value that specifies the exponentof 10 (10^n). The exponent may be in the range –85 to +75. If an unsigned exponent is specified, a plus sign is assumed.

Machine format for a floating-point number is in two parts: the portion containing the exponent, which is called the characteristic, followed by the portion containing the fraction, which is called the mantissa. Therefore, the number specified as a floating-point constant must be converted to a fraction before it can be translated into the proper format. For example, the constant 27.35E2 represents the number 27.35 times 10^2. This constant represented as a fraction would be .2735 times 10^4, the exponent having been modified to reflect the shifting of the decimal point. Once the constant is converted into the proper exponent and fraction, each is translated into its binary equivalent and arranged in machine floating-point format.

The translated constant is placed in a fullword or a doubleword, depending on whether the type is specified as E or D. It is aligned at the proper word or doubleword boundary if a length is not specified. An implied length of four bytes is assumed for a fullword (E) and eight bytes is assumed for a doubleword (D). However, any length up to and including eight bytes may be specified for either type of constant by a length modifier, in which case no boundary alignment occurs.

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

Any of the following statements could be used to specity 46.415 as a positive, fullword, or floating-point constant.

Name	Operation	Operand
	DC	E'46.415'
	DC	E'46415E–3'
	DC	E'+464.15E–1'
	DC	E'.46415E+2'

Decimal Constants – P and Z

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

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

If zoned decimal format is specified (Z), each decimal digit is translated into one byte. The translation is done according to the character set shown in Appendix A. The rightmost byte contains the sign as well as the rightmost digit. For packed decimal format (P), each pair of decimal digits is translated into one byte. The rightmost digit and the sign are translated into the rightmost byte. The bit configuration for the digits is identical to the configurations for the hexadecimal digits 0–9 as shown in Section 2 under "Hexadecimal Delf-Defining Value." for both packed and zoned decimals, a plus sign is translated into the hexadecimal digit C, and a minus sign is translated into the hexadecimal digit D.

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

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

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

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

Examples of decimal constant definitions are:

Name	Operation	Operand
	DC	P'+1.25'
	DC	Z'–543'
	DC	Z'79.68'
	DC	PL3'79.68'

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

Name	Operation	Operand
	UNPK	OUTAREA+30,=PL8'+25'

Address Constants – A and S

An address constant is a storage address that is translated into a constant. Address constants are used for initializing base registers to facilitate the addressing of storage. However, storage addressing and control section communication are also dependent on the use of the USING assembly instruction and the loading of registers. Coding examples that illustrate these considerations are provided under "Base Register Instructions."

An address constant, unlike other types of constants, is enclosed in parentheses. If two or more address constants are specified in an operand, they are separated by commas, and the entire sequence is enclosed by parentheses. There are two types of address constants: A and S.

A-Type Address Constant

This constant is specified as an absolute or relocatable expression. The value of the expression is calculated as explained in Section 2, and is assembled as the constant. The implied length of an A–type constant is four bytes, and the value is placed in the rightmost portion. Alignment is to a fullword boundary, unless a length is specified. A length modifier may be used, in which case no alignment will occur. The length must be 1-4 bytes (without bit length specification).

In the following example, the field generated from the statement named ACON contains three constants, each of which occupies four bytes. The second statement shows the same set of constants specified as literals (i.e., address constant literals).

Name	Operation	Operand
ACON	DC LM	A(108,LOOP,END–START) 3,5,=A(108,LOOP,END–START)

S-Type Address Constant

The S-type address constant is used to store an address in base-displacement form. The 16-bit constant will be aligned on a hlaf-word boundary and given an implied length of 2 bytes. The S-type constants may be specified in two ways:

1. A relocatable or absolute expression enclosed in parenthesis, e.g., S(TABLE). The address value represented by the expression will be converted into the proper base register and displacement value by the assumbler. A general register must be available (through a USING statement) to permit the S-type address constant to be mapped into a base and displacement.
2. Two absolute expressions, the first of which represents the displacement value and the second, the base register, e.g., S[100(10)].

The leftmost four bits of the assembled constant is the base register designation, the remaining twelve bits is the displacement value.

Note: An explicit length or a duplication factor is not allowed; however, a list of constants may be specified as:

```
SADDR    DC    S(TABLE,SCON2,SCON3)
```

The S-type address constants should not be specified as literals.

DS – Define Storage

The DS instruction reserves areas of storage and assigns names to those areas. The use of this instruction is the preferred way of symbolically defining storage for work areas, input/output areas, etc. The size of a storage area that can be reserved by using the DS instruction is limited only by the maximum value of the location counter (however, the length cannot exceed the maximum for the type). The format of the DS statement is:

Name	Operation	Operand
A symbol or blank	DS	One or more operands, separated by commas, written in the format described in the following text

The format of the DS operand is identical to that of the DC operand; exactly the same subfields are employed and are written in exactly the same sequence as they are in the DC operand. There is one difference, that is, the value list (subfield 4) is optional in a DS operand, but it is mandatory in a DC operand.

If a DS operand specifies a constant in subfield 4, the assembly program determines the length of the data and reserves the appropriate amount of storage. It does not assemble the constant as it would for a DC operand. The ability to specify data and have the assembly program calculate the storage area that would be required for such data is a convenience to the programmer. If he knows the general format of the data that will be placed in the storage area during program execution, all he needs to do is show it as the fourth subfield in a DS operand. The assembly program then determines the correct amount of storage to be reserved, thus relieving the programmer of length considerations.

If the DS instruction is named by a symbol, its value attribute is the location of the leftmost byte of the reserved area. The length attribute of the symbol is the length (implied or explicit) of the type of data specified. If the DS has a series of operands, the length attribute for the symbol is developed from the first item in the first operand. Any positioning required for aligning the storage area to the proper type of boundary is done before the address value is determined.

Each field type (e.g., hexadecimal, character, and floating-point) is associated with certain characteristics (these are summarized in Table C–1). The associated characteristics will determine which field-type code the programmer selects for the DS operand and what other information he adds, notably a length specification or a duplication factor. For example, the E floating-point field and the F fixed-point field both have an implied length of four bytes. The leftmost byte is aligned to a fullword boundary. Thus, either code could be specified if it were desired to reserve four bytes of storage aligned to a fullword boundary. Thus, either code could be specified if it were desired to reserve four bytes of storage aligned to a fullword boundary. To obtain a length of eight bytes, one

could specify either the E or F field type with a length modifier of 8. However, a duplication factor would have to be used in order to obtain a larger field size, because the maximum length specification for either type is eight bytes. Note also that specifying length would cancel any special boundary alignment.

In contrast, packed and zoned decimal (P and Z), character (C), and hexadecimal (X) fields have an implied length of one byte. Any of these codes, if used, would have to be accompanied by a length modifier. unless just one byte is to be reserved. Unless a field of one byte is desired, either the length must be specified for the C or X field type, or else the data must be specified (as the fourth subfield), so that the assembly program can calculate the length.

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

Name	Operation	Operand
FIELD	DS	4CL10
AREA	DS	CL100

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

Additional examples of DS statements are:

ONE	DS	CL80	(one 80-byte field, length attribute of 80)
TWO	DS	80C	(80 one-byte fields, length attribute of 1)
THREE	DS	6F	(six fullwords, length attribute of 4)
FOUR	DS	D	(one doubleword, length attribute of 8)
FIVE	DS	4H	(four halfwords, length attribute of 2)
SIX	DS	20CL255	(20 areas of 255 bytes each or one area of 5100, length attribute of 255)

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

SPECIAL USES OF THE DUPLICATION FACTOR

Forcing Alignment

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

Name	Operation	Operand
	DS	OD
AREA	DS	CL128

Defining Fields of an Area

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

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

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

Figure 4–1 illustrates how DS instructions might be used to assign a name to the record area, then define the fields of the area and allocate the storage for them. Note that the first statement names the entire area by defining the symbol RDAREA; the statement gives RDAREA a length attribute of 80 bytes, but does not reserve any storage. Similarly, the fifth statement names a six-byte area by defining the symbol DATE; the three subsequent statements actually define the fields of DATE and allocate storage for them. The second, ninth, and last statements are used for spacing purposes and, therefore, are not named.

Name	Operation	Operand
RDAREA	DS	OCL80
	DS	CL4
PAYNO	DS	CL6
NAME	DS	CL20
DATE	DS	OCL6
DAY	DS	CL2
MONTH	DS	CL2
YEAR	DS	CL2
	DS	CL10
GROSS	DS	CL8
FEDTAX	DS	CL8
	DS	CL18

Figure 4–1. Use of DS Instruction

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 doubleword boundary. The internal machine format of a channel command word is shown in Table 4–3.

Table 4–3. Channel Command Word Format

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

The format of the CCW statement is:

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

The operands are written, from left to right, as follows:

1. An absolute expression that specifies the command code. The value of this expression is right-justified in byte 1.

2. A relocatable or absolute expression that specifies the data address. The value of this expression is right-justified in bytes 2–4.

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

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

The following is an example of a CCW statement:

Name	Operation	Operand
	CCW	X'02',READIN,X'48',80

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

32–35	36–39
0100	1000

An operand may be omitted to imply that the byte or bytes it specifies are to be set to zero. If an operand is omitted, the comma following it may be omitted only if no subsequent operands appear in the statement. For example: the statement:

```
CCW     X'07',INAREA,,160
```

will result in bytes 5 and 6 being assembled as zeros.

If there is a symbol in the name field of the CCW instruction, it is assigned the address value of the leftmost byte of the channel command word after any boundary alignment. The length attribute of the symbol is 8. A byte skipped because of alignment is set to zero.

PROGRAM SECTIONING AND LINKING

The START assembly instruction enables the programmer to identify an unsectioned program or the first section of a multisection program. It also may be used to specify a tentative starting location. The CSECT and DSECT assembly instructions enable the programmer to identify sections of a multisection program. The EXTRN and ENTRY assembly instructions facilitate symbolic linkages between independently assembled programs.

It is often convenient, or necessary, to write a large program in sections. The sections may be assembled separately, then combined subsequently into one object program. The assembly program provides facilities for creating multisectioned programs and symbolically linking separately assembled programs or program sections. However, sectioning a program is optional, and many programs can best be written without sectioning them.

The programmer who is writing an unsectioned program does not need to concern himself with the subsequent discussion of program sections, which are called control sections, nor does he have to employ the CSECT instruction, which is used to identify the control sections of a multisection program. Similarly, he does not need to concern himself with the discussion of symbolic linkages if his program neither requires a linkage to nor receives a linkage from another program. He may, nowever, wish to identify the program and/or specify a tentative starting location for it, both of which may be done by using the START instruction. He may also want to employ the dummy section feature obtained by using the DSECT instruction.

Note: The problem of program sectioning and linking is closely related to the specification of base registers for each control section. This is discussed under " Base Register Instructions " in the subsection " Programming with the USING Instruction." Several sectioning and linking examples are provided in the discussion.

First Control Section

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

1. It can begin without a CSECT definition. If its location counter has been incremented from 0 before the occurrence of a CSECT, it can be resumed with " blank CSECT ". Otherwise, it cannot be resumed.

2. Its tentative loading location may be specified as an absolute value (using the START card).

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

Symbolic Linkages

Symbols may be defined in one program and referred to in another, thus effecting symbolic linkages between independently assembled programs. The linkages can be effected only if the assembly program is able to provide information about the linkage symbols to the loader, which resolves these linkage references at load time. The assembly program places the necessary information in the control dictionary on the basis of the linkage symbols identified by the ENTRY and EXTRN instructions.

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

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

START – START Program

The START instruction may be used to give a name to the program and to indicate the beginning of an assembly. It may be used to specify a tentative starting location for the program. It may also be used to identify an overlay program and to specify a compool to be used in the assembly. The format of the START statement is:

Name	Operation	Operand
Symbol or blank	START	(self-defining value) (,V) (,POOL'poolname)

All or any combination of the fields shown under operand may be used, but they must appear in the order shown.

The symbol in the name field becomes the name of the program. The symbol is assigned the value of the self-defining value in the operand field. The 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 assembly program will assign a name of .NONAME.

The self-defining value in the operand field specifies the initial setting of the location counter. If the value of the operand is not a multiple of 8, the location counter will be set at the next doubleword 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 tentative load location. The loader will only relocate the program if separately assembled programs or Common have conflicting load locations (or if directed to relocate). This enables the programmer to match the locations shown in the listing produced by the assembly program with the locations in storage dump listings.

The field containing ",V", if present, indicates that this is to be an overlay deck.

The field containing ", POOL 'poolname ", if present, indicates that a compool is to be used in the assembly. The " poolname " must be a valid compool name, 2–8 alphanumeric characters long with the 1st character alpha. The terminator may be either a blank, or a quote matching that following " POOL ". Further information on the use of this option may be found in the section on " Assembling with a Compool ".

A START statement may be preceded in the source program only by the ICTL or TITLE statement. If it appears anywhere else or does not appear in the program, the assembly program will set the location counter to zero and name the program .NONAME. Any invalid occurrences of a START statement will not be used.

Either of the following START statements 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'
```

CSECT – Identify Control Section

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

Name	Operation	Operand
A symbol or blank	CSECT	Ignored; should be blank

If a symbol names the CSECT instruction, the symbol is established as the name of the control section. All statements following the CSECT are assembled as part of that control section until a statement which identifies a different control section is encountered (i.e., another CSECT, COM or a DSECT instruction).

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

A special case of the CSECT instruction is ".PREV CSECT". This statement is valid only in a DSECT, and restores the program to the last CSECT which was in use (blank CSECT if no other was declared). Any number of DSECT's may have intervened. This instruction is generated internally during compool processing, but is available to the user. If it appears in a CSECT, it will be flagged as an error.

Another special use of the CSECT instruction is in producing a compool segment overlay deck. This is coded ".OVLY CSECT", and its use is described under the section "Assembling With A Compool".

CONTROL SECTION LOCATION ASSIGNMENT

Control section contents can be intermixed because the assembly program provides a location counter for each control section, as explained in Section 2. Locations are assigned to control sections in such a way that the sections are placed in storage consecutively, in the same order as they first occur in the program. Each control section subsequent to the first begins at the next available doubleword boundary.

The highest location in a control section is the rightmost byte in the highest doubleword assigned during assembly by the control section's location counter. The highest location need not be the last location assigned if there are ORG statements within the control section. The size of the control section is the difference between the highest and initial locations plus one (rounded to a multiple of eight).

DSECT – Identify Dummy Section

A DSECT assembly instruction provides complete definition of a section of code and names associated with that section, without reserving any storage. The format of the DSECT statement is:

Name	Operation	Operand
A symbol	DSECT	Ignored; should be blank

Program statements belonging to dummy sections may be interspersed throughout the program or may be written as a unit. In either case, the appropriate DSECT instruction should precede each set of statements. When multiple DSECT instructions with the same name are encountered, regardless of qualification, the first instruction is considered to initiate the dummy section and the remaining instructions to continue it.

A name on a DSECT may not be repeated, with the same qualification, as a name on any other statement type. If it is so used, a message will appear in the listing and the second usage is ignored. If the name is used first on a statement other than a DSECT and then appears in the name field of a DSECT, that DSECT will not have a name. Once that dummy section is ended, it may not be resumed. An exception to this is that a name which is an external symbol may also be used to define a DSECT. The symbol may also be used to define a DSECT provided that the external symbol definition precedes the DSECT. In this case the ANALYZ listing shows the name on the DSECT statement as undefined if the qualifiers are not the same, or multi-defined if the qualifiers are identical. If the DSECT preceded the external Symbol definition, a serious error is generated for a multi-defined symbol.

Names of statements occurring in dummy sections may appear in USING statements and may, therefore, be used in the operand field of instructions as storage addresses. They should not appear in A-type address constants. Two dummy control sections are not considered to have a constant relativity to one another; i.e.,

1. A USING statement which denotes a register pointing to one dummy section does not make another dummy section addressable.

2. An expression that contains two names, each from a different dummy section, is invalid.

DUMMY SECTION LOCATION ASSIGNMENT

A location counter determines the relative locations of named program elements in a dummy section. The location

counter is always set to zero at the beginning of the dummy section, and the location values assigned to symbols that name statements in the dummy section are relative to the initial statement in the section.

COM – Identify Common Control Section

The COM assembly instruction defines a particular type of control section known as Common. The format of the COM statement is:

Name	Operation	Operand
Must be blank	COM	Ignored; should be blank

The name field must be blank, and the operand field is not used. The location counter is set to zero. If more than one COM statement occurs in one assembly, second and succeeding statements cause resumption of the same control section. Names may be attached to statements within Common, and these names may be qualified. The loader will independently relocate the Common control section so that Common of every separately assembled program, loaded together, has the same starting location. In this way, separately assembled programs may share variables and data in Common.

ENTRY – Identify Entry-Point Symbol

The ENTRY instruction identifies linkage symbols that are defined in this program but may be used by some other program. The format of the Entry statement is:

Name	Operation	Operand
Must be blank	ENTRY	One or more relocatable symbols, separated by commas, that also appear as statement names

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

Name	Operation	Operand
	ENTRY	SINE, COSINE

EXTRN – Identify External Symbol

The EXTRN instruction identifies linkage symbols that are used by this program but defined in some other program. Each external symbol must be identified; this includes symbols that name control sections.

The format of the EXTRN statement is:

Name	Operation	Operand
Ignored, should be blank	EXTRN	One or more relocatable symbols separated by commas

The symbols in the operand field may not appear as names of statements in this program (except to define a DSECT; see section on DSECT). The following example identifies three external symbols that have been used as operands in this program but are defined in some other program.

Name	Operation	Operand
	EXTRN	RATEBL, PAYCALC, WITHCALC

An example that employs the EXTRN instruction appears subsequently under "Programming with the USING Instruction."

QUAL – Qualify Names

The primary function of the QUAL assembly instruction is to distinguish between two occurrences of the same symbol in different sections of a program. The format of the QUAL statement is:

Name	Operation	Operand
Blank	QUAL	c or blank

Operand c is an alphabetic character or decimal digit which is to be used as the qualifier for the section of code immediately following the QUAL statement. A qualified section is begun by a QUAL statement and is terminated by the next QUAL statement which has a different character in the operand field. A blank operand field in a QUAL statement causes the section following the QUAL to be unqualified. The same character which appears in the operand fields of two QUAL statements causes a resumption of the first section to use that qualifier.

Two facts about the function of QUAL should be noted:

1. QUAL is an instruction to the assembly program. It qualifies symbolic references for the assembly process only, and no qualification will ever appear on symbols which go to the loader via the ESD (External Symbol Dictionary).

2. Every symbol in the coding, whether it appears in the name field or the operand field, immediately following a QUAL statement is qualified (except for a CSECT name which remains unqualified).

Any symbol in a qualified section that is external in nature is entered into the ESD in unqualified form. At the same time, for internal use, the same symbol is entered into the program's symbol table with the current qualifier. When such a symbol is referenced subsequently, it must be qualified if the reference is strictly internal, and it must be unqualified if the reference causes a search of the ESD. References which appear in an EXTRN statement should be unqualified, because such references are external in nature. All other symbolic operands must refer to symbols in the same qualified section, or they must be coded with the correct qualifier by the programmer. A symbol is coded with a qualifier by appending a period and the qualifier. The following example illustrates internal cross references between qualified sections:

```
                QUAL    X
        FIRST   LA      4,2(6)
                .
                .
                QUAL    3
1.      TRY     BC      8,FIRST.X
2.              BC      7,TRY
```

Statement 1 references a symbol which is defined in a differently qualified section and, therefore, correctly includes that qualifier. If FIRST had been written without the qualifier, the assembly program would have added the qualifier 3, creating an undefined symbol FIRST.3. Statement 2 references a symbol within its own qualified section, and, therefore, does not include the qualifier character 3, which would otherwise be necessary to reference TRY. An operand within a qualified section may refer to an unqualified name by writing the name followed by a period and no qualifier.

QUALIFIERS ON SYSTEM SYMBOLS

If a system symbol is qualified and the system symbol plus qualifier is not defined internally in the program, the qualification will be ignored.

BASE REGISTER INSTRUCTIONS

The USING and DROP assembly instructions enable programmers to use expressions that represent implicit addresses as operands of machine-instruction statements, leaving the assignment of base registers and the calculation of displacements to the assembly program.

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

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

USING – Use Base Address Register

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

Name	Operation	Operand
Ignored; should be blank	USING	From 2–17 expressions of the form v, r1, r2, r3, . . . , r16

Operand v is an absolute or simply relocatable expression. It specifies a value that the assembly program can use as a base address. Other operands must be absolute expersions. Operand r1 specifies the general register that can be assumed to contain the base address represented by operand v. Operands r2, r3, r4, . . . , specify registers that can be assumed to contain v+4096, v+8192, v+12288, . . . , respectively. The values of the operands r1, r2, r3, . . . , r16 must be between 0 and 15. For example, the statement:

Name	Operation	Operand
	USING	*, 12, 13

tells the assembly program it may assume that the current value of the location counter will be in general register 12 at object time, and that the current value of the location counter, incremented by 4096, will be in general register 13 at object time.

If the programmer changes the value in a base register currently being used, the assembly program must be told the new value by means of another USING statement. In the following sequence, the assembly program first assumes that the value of ALPHA is in register 9. The second statement then causes the assembly program to assume that ALPHA+1000 is the value in register 9. This means that any displacement between the two statements is calculated with the assumption that ALPHA is in register 9. After the second statement, any displacement is calculated assuming that register 9 contains ALPHA+1000.

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

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

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

WARNING

If a USING specifies an absolute value less than 4096, the assembler may consider a constant explicit displacement as covered by the base, leading to a serious error.

DROP – Drop Base Address Register

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

Name	Operation	Operand
Must be blank	DROP	Up to 16 absolute expressions of the form r1, r2, R3, . . . , r16; or ".ALL"

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

Name	Operation	Operand
	DROP	7, 11

Use of ".ALL" will make all registers unavailable. There is no test to determine if a given register appeared previously in a USING statement.

It is not necessary to use a DROP statement when the base address in a register is changed by a USING statement; nor are DROP statements needed at the end of the source program. A register made unavailable by a DROP instruction can be made available again by a subsequent USING instruction.

Programming With the Using Instruction

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

If an instruction has both implied and explicit base registers, the assembler will use the implied base, and give a diagnostic for a serious error.

The instruction sequence in Figures 4–2 and 4–3 illustrate the assignment of base registers and show several methods for loading base registers. In Figure 4–2, the

BALR instruction loads register 2 with the address of the first storage location immediately following the instruction; in this case, it is the location named FIRST. The USING instruction indicates to the assembly program that register 2 contains this location. When employing this method, the USING instruction must immediately follow the BALR instruction. *No other USING or load instructions are required if the location named LAST is within 4095 bytes of FIRST* (and there are no external symbols).

In Figure 4–3, the BALR and LM instructions load registers 2–5. The USING instructions indicate to the assembly program that these registers are available as base registers for addressing a maximum of 16,384 consecutive bytes of storage, beginning with the location named HERE. The number of addressable bytes may be increased or decreased by altering the number of registers designated by the USING and LM instructions and the number of address constants specified in the DC instruction.

ADDRESSING MULTIPLE CONTROL SECTIONS

Special care must be exercised in a program with multiple Control Sections (CSECTs). Current implementation of the BAL Assembler effectively concatenates all CSECTs into one in the final object module. As a result, a USING set for n(<4096) bytes from the end of a CSECT will cover the first 4069–n bytes of the next CSECT. This can result in program errors from unexpected register assignments.

However, it is not safe to assume that a register may be used to cover multiple CSECTs on a single USING. Considerations of compatibility with other assemblers, and possible implementation of "scatter-load" capability in the future, may result in assembler changes which would prohibit USINGs from crossing CSECT boundaries.

Programs should, therefore, be designed with separate USINGs for each CSECT, and carefully checked to ensure against overlap.

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

Figure 4–2. Base Register Assignment

ADDRESSING DUMMY SECTIONS

The programmer may wish to describe the format of an area whose storage location will not be determined until the program is executed. He can describe the format of the area in a dummy section, and he can use symbols defined in the dummy section as the operands of machine instructions. To effect references to the storage area, he does the following:

1. Provides a USING statement which specifies both a general register that the assembly program can assign to the machine instructions as a base register and a value for the dummy section that the assembly program may assume the register contains.
2. Loads the same register with the symbolic address of the storage area.

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

Each DSECT is a totally separate entity, and is not contiguous to any CSECT or other DSECT. Therefore a USING on a symbol within a DSECT is good only for that DSECT. Overlap into another control section cannot occur in this case.

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

Figure 4–3. Base Register Assignment

An example of this use is shown in Figure 4–4. An area has been defined by the dummy section RECORD. The second USING statement specifies that general register 3 is available for use as a base register, and that it contains the value Record. However, the program loads the value stored at ADDR into register 3. This value is the initial address of the area to which the programmer wishes to refer at execution time. Thus, all machine instructions that employ symbols defined in the dummy section will refer to storage locations relative to the address in register 3.

```
Name        Operation   Operand
MAINPROG    CSECT
BEGIN       BALR        2,0
            USING       *,2
            .
            .
            L           3,ADDR
            USING       RECORD,3
            CLI         RCDCODE,'A'
            BE          ATYPE
            .
            .
ATYPE       MVC         FIELD1,RCDFLD1
            MVC         FIELD2,RCDFLD2
            .
            .
ADDR        DS          F
FIELD1      DS          CL20
FIELD2      DS          CL18
            .
            .
RECORD      DSECT
RCDCODE     DS          CL1
RCDFLD1     DS          CL20
RCDFLD2     DS          CL18
            .
            .
            END         BEGIN
```

Figure 4–4. Addressing Dummy Sections

ADDRESSING EXTERNAL PROGRAMS

An external symbol that names data may be referred to as follows:

1. Identify the external symbol with the EXTRN instruction, and create an address constant from the symbol.
2. Load the constant into a general register, and use the register for base addressing.

For example, to use an area named RATETBL, which is in another program, the coding in Figure 4–5 might be used.

LISTING CONTROL INSTRUCTIONS

The listing control instructions identify an assembly listing and assembly output cards, provide blank lines in an assembly listing, and designate how much detail is to be included in an assembly listing. In no case are instructions or constants generated in the object program.

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

Figure 4–5. Addressing External Programs

TITLE – Identify Assembly Output

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

Name	Operation	Operand
Symbol or blank	TITLE	A sequence of characters

The name field may contain a symbol of from one to four alphameric characters, the first of which must be alphabetic. The contents of the name field are punched into columns 73–76 of all the output cards for the program, if the TITLE is before the START card.

The operand field extends to column 71 (inclusive) of the card. The contents of the operand field are printed at the top of each page of the assembly listing.

A program may contain more than one TITLE statement. Each TITLE statement provides the heading for pages in the assembly listing that follow it, until another TITLE statement is encountered. Additionally, the first TITLE statement in a program provides the heading for pages of the assembly listing that precede it if it is put before the START card. Each TITLE statement that is encountered causes the listing to be advanced to a new page (before the heading is printed).

For example, if the following statement is the first TITLE statement to appear in a program (and is before the START card):

Name	Operation	Operand
PGM1 PROG1	TITLE START	THIS IS THE FIRST HEADING 0

then PGM1 is punched into all of the output cards (columns 73–76) and this heading appears at the top of each page: THIS IS THE FIRST HEADING.

If the following statement occurs later in the same program:

Name	Operation	Operand
	TITLE	THIS IS A NEW HEADING

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

If a new symbol was included in the name field, any data remaining to be punched would be punched immediately with the old name in columns 73–76, and cards that are punched after the appearance of the TITLE card would have the new name punched.

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 statement is:

Name	Operation	Operand
Must be blank	EJECT	Not used; should be blank

If the next line of the listing normally appears at the top of a new page, the EJECT statement has no effect. The EJECT statement will not be printed.

SPACE – Space Listing

The SPACE instruction inserts one or more blank lines in the listing. The format of the SPACE statement is:

Name	Operation	Operand
Must be blank	SPACE	A decimal value or blank

A decimal value specifies the number of blank lines to be inserted in the assembly listing. If this value exceeds the number of lines that remain on the listing page, the statement will have the same effect as an EJECT statement. The value will be ignored if it is greater than 56. A blank operand field will cause one line to be skipped. If the operand field is invalid, the statement will be ignored. Anything in the name field will not be used. The statement will not be printed.

DOUBL – Double Space

The DOUBL instruction is used to have a single space between each line in the listing. It may appear anywhere in the program. The output listing will be entirely double spaced, regardless of where the DOUBL instruction appears. The format of the DOUBL statement is:

Name	Operation	Operand
Must be blank	DOUBL	Ignored; should be be blank

PRINT – Print Optional Data

The PRINT instruction designates the amount of detail that is to be included in an assembly listing. The format of the PRINT statement is:

Name	Operation	Operand
Must be blank	PRINT	a,b or ,b or a

Where " a " can be NODATA or DATA, " b " in the first format can be NOLIT or LIT, and " b " in the second format can be NOLIT or LIT with DATA assumed. The third format can be NODATA or DATA with LIT assumed.

The operands have the following meanings to the assembly program:

DATA	lists all bytes assembled for each DC statement.
NODATA	lists only the first line assembled for each DC statement.
LIT	lists the contents of the literal pool.
NOLIT	does not list the contents of the literal pool.

A program may contain any number of PRINT statements. A PRINT statement controls the printing of the assembly listing until the occurrence of another PRINT statement. If no PRINT statement appears in a program or until the occurrence of the first PRINT statement, DATA and LIT are assumed.

NLIST – Suppress Listing

This instruction will suppress the printing of the listing,

The format of the statement is:

Name	Operation	Operand
Must be blank	NLIST	Should be blank

Note: If BAL errors are detected while an NLIST is in effect, the statement plus errors will be printed. Warning errors suppressed by SPEM statements will not force printing of an NLISTed statement.

LIST – Resume Listing

This instruction resumes the printing of the listing once a NLIST has been given. The LIST is not itself printed in the listing.

The format of the statement is:

Name	Operation	Operand
Must be blank	LIST	Should be blank

PROGRAM CONTROL INSTRUCTIONS

The program control instructions specify the end of an assembly, set the location counter to a value or word boundary, specify the placement of literals in storage, check the sequence of input cards, and indicate the statement format.

ICTL – Input Format Control

The ICTL instruction allows the programmer to alter the normal format of his source program statements. The format of the ICTL statement is:

Name	Operation	Operand
Must be blank	ICTL	d

Operand d is a self-defining value that specifies the begin column for the name field. The value must be between 1 and 66. If the ICTL statement is not used, the begin column is assumed to be column 1. When the ICTL is used, it must be the first statement of the program.

ISEQ – Input Sequence Checking

The ISEQ instruction may be used to cause the assembly program to check the sequence of input cards. The columns checked will always be 73–80. The format of the ISEQ statement is:

Name	Operation	Operand
Must be blank	ISEQ	Ignored; should be blank

Sequence checking begins with the first card following the ISEQ statement. Comparison of adjacent cards makes use of the eight-bit internal collating sequence. If the sequence number of a card is equal to or less than the sequence number of the preceding card, a warning flag will occur in the listing. This flag will not alter the processing of the statement(s).

SSEQ – Suppress Sequence Checking

The SSEQ instruction causes the assembly program to cease checking the sequence of input cards (in columns 73–80). The format of the SSEQ statement is:

Name	Operation	Operand
Must be blank	SSEQ	Ignored; should be blank

ORG – Reset Location Counter

The ORG instruction alters the setting of the location counter for the current control section. An ORG statement may be used anywhere in a program, as often as desired. An ORG statement may appear in a CSECT, DSECT, or COM control section. The format of the ORG statement is:

Name	Operation	Operand
Symbol or blank	ORG	A relocatable expression

Any symbols in the expression must have been previously defined in the same control section in which the ORG statement appears. The value of the expression sets the location counter of the current control section.

The statement:

```
ORG         *+500
```

increases the location counter by 500 bytes above its current setting. Nothing is assembled for the 500 bytes skipped; i.e., the bytes are not cleared by the assembly program.

An ORG instruction that resets the location counter below its initial value in the control section or references another control section will not be used; it will be printed only in the listing. If the operand field is blank or invalid, the ORG instruction will not be used. If a name is specified, it will receive the value of the expression.

The ORG instruction provides an alternative way of reserving storage areas; the preferred way is with the DS (Define Storage) assembly instruction.

However, when a storage area cannot be conveniently defined by 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
```

LTORG – Begin Literal Pool

The LTORG instruction may be used to specify the place of the literal pool into which all literals thus far encountered in a program are to be assembled. A LTORG statement may appear at any point in a program. If the LTORG appears in a DSECT, however, the programmer must make certain that only the DSECT literals are present in the literal pool. The format of the LTORG statement is:

Name	Operation	Operand
Symbol or blank	LTORG	Ignored; should be blank

The effect of the LTORG statement is to position all the literals encountered up to the LTORG statement (either from the beginning of the program or from a previous LTORG statement) at appropriate boundaries, starting at the first doubleword boundary that follows the LTORG statement.

Any literals used after the last LTORG statement in a program are placed at the end of the first control section. If there is no LTORG statement in a program, all literals used in the program are placed at the end of the first control section. It is the responsibility of the programmer to ensure that a base register has been loaded and a proper USING instruction has been given so that the literals can be addressed.

The assembly program first stores those literals which are 8 bytes, or a multiple of 8 bytes in length; then it stores literals which are 4 bytes, or an odd multiple of 4 bytes in length. Literals of 2 bytes or an odd multiple of 2 bytes are stored next, and finally all literals which consist of an odd number of bytes. Within each group the literals are stored in the exact order in which they occur in the source program (or since the last previous LTORG statement).

As each literal is handled, the contents and actual byte construction of the source component that specifies it are checked against the existing literal collection. If all bits of the new specification match those of a literal previously stored, the duplicate is not stored, and its reference points to the literal it duplicates. However, if the specification components differ in any respect, even though the resultant literals may be identical, both literals are stored.

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

X'F0' C'0'	**Both are stored**
C'A' 'A'	**Both are stored**
X'FFFF' X'FFFF'	**Identical; only the first is stored**
X'00' X'0'	**Both are stored**
XL3'0' HL3'0'	**Both are stored**

It is the responsibility of the programmer to be certain that he has specified a literal of the correct length so that if, for example, the following code is encountered by the assembly program:

```
L 4,=X'4'
```

it assumes a length of 1 byte for the literal and the instruction may be erroneously written. If the programmer states:

```
L 4,=F'4'
```

the literal is considered a fullword and is stored on a word boundary, making the instruction valid.

CNOP – Conditional No Operation

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

The CNOP instruction ensures the alignment of the location counter setting to a halfword, word, or doubleword boundary. If the location counter is already properly aligned, the CNOP instruction has no effect. If the specified alignment requires the location counter to be incremented, one to three no operation instructions are generated, each of which uses two bytes. If an odd number of bytes is skipped, the first byte will be set to zero. The format of the CNOP statement is:

Name	Operation	Operand
Symbol or blank	CNOP	Two decimal values of the form: b, w

Operand b specifies at which byte in a word or doubleword the location counter is to be set; b can be 0, 2, 4, or 6. Operand w specifies whether byte b is in a word (w=4) or doubleword (w=8). If the operand field is blank or invalid, the CNOP instruction will not be used. The following pairs of b and w are valid:

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

Figure 4–6 shows the position in a doubleword that each of these pairs specifies. Note that both 0,4 and 2,4 specify two locations in a doubleword.

Assume that the location counter is currently aligned at a doubleword boundary. Then the CNOP instruction in this sequence:

Name	Operation	Operand
	CNOP BALR	0,8 2,14

Doubleword							
Word				Word			
Halfword		Halfword		Halfword		Halfword	
Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte
0, 4 0, 8		2, 4 2, 8		0, 4 4, 8		2, 4 6, 8	

Figure 4–6. CNOP Position Specifications

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

Name	Operation	Operand
	CNOP BALR	6,8 2,14

causes the object code for two branch on condition zero operations to be generated, thus aligning the BALR instruction at the last halfword in a doubleword as:

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

After the BALR instructions is generated, the location counter is at a doubleword boundary, thereby ensuring an unbroken instruction flow if a constant definition for a doubleword follows. Without the CNOP, doubleword alignment would be necessary.

END – End Program

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

The format of the END statement is:

Name	Operation	Operand
Must be blank	END	A relocatable expression or blank

The expression, if present, specifies the point to which control is transferred when loading is complete. The point to which the programmer usually wants to transfer control is the first machine instruction in the program, as shown in the following sequence:

Name	Operation	Operand
AREA BEGIN	START DS BALR USING . . . END	2000 50F 2,0 *,2 BEGIN

SPEM – Suppress Possible Error Messages

If the programmer desires to have the assembly program suppress the listing of certain or all *possible* error messages, he must include a SPEM statement in his program. The format of the statement is:

Name	Operation	Operand
Must be blank	SPEM	Blank or a number

If the operand field is blank, all possible error messages (PEM 's) are suppressed. If the operand field contains a number from 1 to 7, the following applies:

Number	Suppresses
1	Privileged operation PEM
2	Void expression PEM
3	Privileged operation and void expression PEM 's
4	All PEM 's except privileged operation and void expression
5	All PEM 's except void expression
6	All PEM 's except privileged operation
7	All PEM 's (same as blank)

The SPEM instruction does not cause the list suppression of *serious* errors; nor does it negate the cumulative count of *possible* error messages.

RPEM – Resume Possible Error Messages

The RPEM assembly instruction causes the assembly program to resume the listing of possible errors. The format of the statement is:

Name	Operation	Operand
Must be blank	RPEM	Must be blank

The SPEM and RPEM assembly instructions can be used at more than one point in a program to cause alternate suppression and resumption of the listing of possible errors.

LIB – Library Update

The LIB assembly instruction produces LIB cards for updating the JOVIAL library. The operand field contains information that is to appear in columns 19–42 of the LIB cards used as input to the library update program. Further information on the data required in columns 19–42 is available in LIBEDT-02, p. 5. The assembly program does not check the validity of the information in the operand field. It merely produces a 12–2–9 LIB card with the program name in columns 11–18 and columns 19–42 identical to the input operand field.

The format of the LIB statement is:

Name	Operation	Operand
Must be blank	LIB	Library Update information

Assembling with a Compool

The BAL programmer can make use of a standard JOVIAL-type Compool in his assemblies. The data obtained this way is essentially the same as that available to the JOVIAL programmer using DIRECT code. The data is (generally) in segments, and each segment will appear in the program as:

```
            EXTRN   ZXsegnam
ZXsegnam    DSECT
ZUsegnam    DS      0F
ZXdata1     DS      xF
ZXdata2     EQU     ZXdata1+y
ZXdata3     EQU     X'nnnnnn'
            .
            .
            .
.PREV       CSECT
```

where

segnam	is Segment name
data1	is a JOVIAL TABLE
data2	is a JOVIAL ITEM
data3	is a JOVIAL PARAMETER ITEM

The programmer can selectively include desired segments, or include the entire compool.

PSEG – Select Compool Segment

The PSEG Instruction is the means by which the programmer can select compool segments. In order to use this option, the POOL option on the start card must be used. Otherwise all PSEG cards will be flagged as errors.

The format of the PSEG statement is:

Name	Operation	Operand
Ignored; should be blank	PSEG	Segment name Segment name, options Options

The use of " PSEG segnam " will cause a request for a segment to be placed in a table for inclusion in the listing at a later point. When "PSEG segnam,.USE" or "PSEG ,.USE" are specified, the table is referenced, and all requested segments will appear in the listing following the PSEG statement.

If " PSEG .ALL ", " PSEG ", or " PSEG ," is specified, the entire compool will appear in the listing. Unless some segments have been previously included, all will immediately follow the PSEG statement.

If the program has " POOL " specified on the Start card, but has no PSEG statements, the entire compool will appear just prior to the "END " statement.

If no PSEG statements have ".USE " or " ALL ", specified segments are placed just prior to the END statement. This case is similar to the complete omission of PSEG 's, but does allow selection of segments.

SEGMENT ASSEMBLIES

If assemblies of compool segments are required, either for loading or as object decks, the user may specify PUNCHC or LISTP on his $BAL card, or

```
PSEG        .PUNC
or
PSEG        .PUNC, .LIST
```

in the program.

Either will cause assemblies of all compool segments appearing in the program. Use of ",.LIST " or LISTP will cause the assemblies to use the " LIST, ANALYZ " option. " PUNCH " will or will not be used, according to whether it was requested for the original program.

WARNING

This option requires the availability of the WORK1 tape. If it is not available, the option will be ignored, but assembly will continue.

COMPOOL TAPE

The Assembler will accept either a Compool or an MLC tape. A Compool tape, if used, must be mounted on .COMP. An MLC may be mounted on either .LIB or .COMP.

LISTING OF SEGMENTS

Normally the Compool DSECT's will not appear on the listing. If a listing of the DSECT's is desired, the user may specify "LISTD" on the $BAL card. If a selective listing of certain segments is desired, it can be obtained by grouping those segments to be listed with a "PSEG ,.USE ,.LIST" card, and the other segments with a separate "PSEG ,.USE" card.

COMPOOL SEGMENT OVERLAY ASSEMBLIES

A special technique may be used to obtain an overlay deck for a compool segment. The technique involves the use of the compool segment as a CSECT and thus permits referencing the compool to obtain the relation displacements required to preset compool items. To excercise the option, the program name on the START card for the overlay deck must be identical to the compool segment being overlaid. Both the ",V" and ",POOL 'name" options must also be furnished. A typical deck set-up is as follows:

```
ZXsegnam    START     O,V,POOL' COMPOOL
.OVLY       CSECT
            PSEG      segnam    NOTE: No
                                ZX prefix
            .
            .
            .
```

[NOTE: Additional PSEGS for other required segments may follow and will appear normally as DSECTS. Only one segment may be overlaid.]

```
            PSEG      ,.USE
            ORG       ZXname1
            DC                  } Preset Data
            ORG       ZXname2
            DC
            .
            .
            .
            END
```

The resulting assembly will be as follows:

```
ZXsegnam    START     O,V,POOL' COMPOOL
.OVLY       CSECT
            PSEG      segnam
```

[NOTE: Alternate other required segments.]

```
            PSEG      ,.USE
Z$segnam    CSECT
```

[NOTE: Compool segment 'segnam' equates and DS's.]

```
            ORG       ZXname1
            DC
            .
            .
            .
            END
```

The ".OVLY CSECT" must precede the "PSEG ,.USE". The segment to be overlaid will be listed in this case, but normal rules apply to other segments (if any). To suppress the listing of the segment, precede the "PSEG ,.USE" with "NLIST" and follow it with "LIST".

ERRORS

In addition to errors detailed above, use of a PSEG card with an invalid segment name or a segment name not on the compool is a serious error. Duplication of segment names will cause a warning to be issued.

RECOMMENDED PROCEDURES

For maximum efficiency of assembly, only one PSEG statement with the " .USE " option should appear. Since it is often desired to use Compool-defined Symbols in EQU statements, the Compool PSECT's should appear as early in the program as possible.

DEBUG Instructions

UNCONDITIONAL DUMP

The unconditional dump card is used to get a dump of the specified portion of storage each time the specified instruction address is reached during program execution. The format of the unconditional dump card is:

Name	Operation	Operand
Symbol	DUMP	Mnemonic format, identifying label, " from " address, " to " address

The following example of an unconditional dump card will cause the area of storage from BEGIN to MOVE to be dumped every time the location PACK is reached. The dump will have the identifying label UCDUMP and a hexadecimal with mnemonics format.

```
PACK  DUMP  HEXI,UCDUMP,BEGIN,MOVE
```

CONDITIONAL DUMP

The conditional dump card gets a dump of the specified portion of storage each time the specified instruction address is reached and the specified conditions are met. The format of the conditional dump card is:

Name	Operation	Operand
Symbol	DUMPC	Mnemonic format, identifying label, " from " address, " to " address, after nnn time instruction is reached – start dumping, after nnn time instruction is reached – halt dumping, dump after every nnn time instruction is reached

The following example of a conditional dump card will cause the area of storage from START to TRANS to be dumped the third time location ADD is reached and every second time the location is reached thereafter, until the fifteenth time. The dump will have the identifying label CONDMP and a regular hexadecimal format.

```
ADD  DUMPC  HEX,CONDMP,START,TRANS,003,015,
                                        002
```

REGISTER-STORAGE CONDITIONAL DUMP

The register-storage conditional dump card compares the contents of a general register with the contents of a location in storage each time the specified instruction address is reached. The specified portion of storage is dumped when the condition of comparison is met. The format of the register-storage conditional dump card is:

Name	Operation	Operand
Symbol	DUMPR	Mnemonic format, identifying label, "from" address, "to" address, condition of comparison,, general register number, storage location (omitted if condition Z is specified)

The condition of the fullword comparison between the register and the storage location may be one of the following:

Condition	Meaning
E	Equal
N	Not equal
L	Register is less than storage
G	Register is greater than storage
Z	Register is zero

The register number must be in decimal format and may not specify a floating-point register. The storage location need not be on a fullword boundary.

The following example of a register-storage conditional dump card will cause the storage area from FLIGHTNO to ASSIGN to be dumped every time location SWITCH is reached and there is an equal compare between general register 8 and storage location LOOPBSY. The dump will have the identifying label RSDUMP and a doubleword floating-point format.

```
SWITCH  DUMPR  DPFL,RSDUMP,FLIGHTNO,ASSIGN,
               E,8,LOOPBSY
```

UNCONDITIONAL TRACE

The unconditional trace card obtains trace information after the execution of each instruction within the specified area. The format of the unconditional trace card is:

Name	Operation	Operand
Blank	TRACE	Identifying label, "start" address, "end" address

The following example of an unconditional trace card will cause trace information to be issued after the execution of each instruction from LOOP to ENDLOOP. The trace will have the identifying label TRACE1.

```
TRACE  TRACE1,LOOP,ENDLOOP
```

BRANCH CONDITIONAL TRACE

The branch conditional trace card obtains trace information after the execution of each successful branch within the specified area. The format of the branch conditional trace card is:

Name	Operation	Operand
Blank	TRACE	Identifying label, "start" address, "end" address

The following example of a branch conditional trace card will cause trace information to be issued after the execution of each successful branch in the trace area from COMPUTE to MOVEOUT. The trace will have the identifying label TRACE2.

```
TRACB  TRACE2,COMPUTE,MOVEOUT
```

PHYSICAL RECORD TAPE DUMP

The physical record tape dump card gets a dump of the physical tape records at successful or unsuccessful end of job. The format of the physical record tape dump card is:

Name	Operation	Operand
Blank	TDMPP	Mnemonic format, identifying label, logical tape drive (0–99), "from" file number (1–99), "from" record within above file (1–99999), "to" file number (1–99), "to" record within above file (1–99999), number of bytes to be dumped (1–999) for each physical record

If the entire physical record is to be dumped, the number of bytes may be left blank.

The following example of a physical record tape dump card will cause all bytes of information from every physical record, from file 2 record 129 to file 4 record 5 on logical tape drive 17, to be dumped after program execution. The dump will have the identifying label TDUMP1 and an alphameric format.

```
TDMPP  ALPH,TDUMP1,17,2,129,4,5
```

LOGICAL RECORD TAPE DUMP

The logical record tape dump card gets a dump of the logical tape records at successful or unsuccessful end of job. The format of the logical record tape dump card is:

Name	Operation	Operand
Blank	TDMPL	Mnemonic format, identifying label, logical tape drive (0–99), "from" file number (1–99), "from" record within above file (1–99999), "to" file number (1–99), "to" record within above file (1–99999), number of bytes to be dumped (1–999) for each physical record

If the entire logical record is to be dumped, the number of bytes may be left blank.

The following example of a logical record tape dump card will cause 100 bytes of information from each logical record from file 1 record 1 to file 1 record 1200 on logical tape drive 23 to be dumped after program execution. The dump will have the identifying label TDUMP2 and a regular hexadecimal format.

```
TDMPL  HEX,TDUMP2,23,1,1,1,1200,100
```

EMERGENCY DUMP

The emergency dump card gets a dump of the specified portion of storage if the program is unable to continue. The format of the emergency dump card is:

Name	Operation	Operand
Blank	DUMPE	Mnemonic format, identifying label, "from" address, "to" address

The following example of an emergency dump card will cause the storage area from BEGIN to END to be dumped if the program cannot be completed. The dump will have the identifying label EMERG1 and a hexadecimal with mnemonics format.

```
DUMPE  HEXI,EMERG1,BEGIN,END
```

Appendix A. CHARACTER CODES

Table A–1. Character Codes

8-BIT BCD CODE	CHARACTER SET PUNCH COMBINATION	PRINTER GRAPHICS	DECIMAL	HEXA-DECIMAL
00000000	12,0,9,8,1		0	00
00000001	12,9,1		1	01
00000010	12,9,2		2	02
00000011	12,9,3		3	03
00000100	12,9,4		4	04
00000101	12,9,5		5	05
00000110	12,9,6		6	06
00000111	12,9,7		7	07
00001000	12,9,8		8	08
00001001	12,9,8,1		9	09
00001010	12,9,8,2		10	0A
00001011	12,9,8,3		11	0B
00001100	12,9,8,4		12	0C
00001101	12,9,8,5		13	0D
00001110	12,9,8,6		14	0E
00001111	12,9,8,7		15	0F
00010000	12,11,9,8,1		16	10
00010001	11,9,1		17	11
00010010	11,9,2		18	12
00010011	11,9,3		19	13
00010100	11,9,4		20	14
00010101	11,9,5		21	15
00010110	11,9,6		22	16
00010111	11,9,7		23	17
00011000	11,9,8		24	18
00011001	11,9,8,1		25	19
00011010	11,9,8,2		26	1A
00011011	11,9,8,3		27	1B
00011100	11,9,8,4		28	1C
00011101	11,9,8,5		29	1D
00011110	11,9,8,6		30	1E
00011111	11,9,8,7		31	1F
00100000	11,0,9,8,1		32	20
00100001	0,9,1		33	21
00100010	0,9,2		34	22
00100011	0,9,3		35	23
00100100	0,9,4		36	24
00100101	0,9,5		37	25
00100110	0,9,6		38	26
00100111	0,9,7		39	27
00101000	0,9,8		40	28
00101001	0,9,8,1		41	29
00101010	0,9,8,2		42	2A
00101011	0,9,8,3		43	2B
00101100	0,9,8,4		44	2C
00101101	0,9,8,5		45	2D
00101110	0,9,8,6		46	2E
00101111	0,9,8,7		47	2F
00110000	12,11,0,9,8,1		48	30
00110001	9,1		49	31
00110010	9,2		50	32
00110011	9,3		51	33
00110100	9,4		52	34
00110101	9,5		53	35
00110110	9,6		54	36
00110111	9,7		55	37
00111000	9,8		56	38
00111001	9,8,1		57	39
00111010	9,8,2		58	3A
00111011	9,8,3		59	3B
00111100	9,8,4		60	3C

Table A–1. Character Codes (Continued)

8-BIT BCD CODE	CHARACTER SET PUNCH COMBINATION	PRINTER GRAPHICS	DECIMAL	HEXA-DECIMAL
00111101	9,8,5		61	3D
00111110	9,8,6		62	3E
00111111	9,8,7		63	3F
01000000		blank	64	40
01000001	12,0,9,1		65	41
01000010	12,0,9,2		66	42
01000011	12,0,9,3		67	43
01000100	12,0,9,4		68	44
01000101	12,0,9,5		69	45
01000110	12,0,9,6		70	46
01000111	12,0,9,7		71	47
01001000	12,0,9,8		72	48
01001001	12,8,1		73	49
01001010	12,8,2		74	4A
01001011	12,8,3	. (period)	75	4B
01001100	12,8,4	<-	76	4C
01001101	12,8,5	(	77	4D
01001110	12,8,6	+	78	4E
01001111	12,8,7		79	4F
01010000	12	&	80	50
01010001	12,11,9,1		81	51
01010010	12,11,9,2		82	52
01010011	12,11,9,3		83	53
01010100	12,11,9,4		84	54
01010101	12,11,9,5		85	55
01010110	12,11,9,6		86	56
01010111	12,11,9,7		87	57
01011000	12,11,9,8		88	58
01011001	11,8,1		89	59
01011010	11,8,2		90	5A
01011011	11,8,3	$	91	5B
01011100	11,8,4	*	92	5C
01011101	11,8,5	)	93	5D
01011110	11,8,6		94	5E
01011111	11,8,7		95	5F
01100000	11	-	96	60
01100001	0,1	/	97	61
01100010	11,0,9,2		98	62
01100011	11,0,9,3		99	63
01100100	11,0,9,4		100	64
01100101	11,0,9,5		101	65
01100110	11,0,9,6		102	66
01100111	11,0,9,7		103	67
01101000	11,0,9,8		104	68
01101001	0,8,1		105	69
01101010	12,11		106	6A
01101011	0,8,3	,	107	6B
01101100	0,8,4	%	108	6C
01101101	0,8,5		109	6D
01101110	0,8,6		110	6E
01101111	0,8,7		111	6F
01110000	12,11,0		112	70
01110001	12,11,0,9,1		113	71
01110010	12,11,0,9,2		114	72
01110011	12,11,0,9,3		115	73
01110100	12,11,0,9,4		116	74
01110101	12,11,0,9,5		117	75
01110110	12,11,0,9,6		118	76
01110111	12,11,0,9,7		119	77
01111000	12,11,0,9,8		120	78
01111001	8,1		121	79
01111010	8,2		122	7A
01111011	8,3	#	123	7B
01111100	8,4	@	124	7C

Table A–1. Character Codes (Continued)

8-BIT BCD CODE	CHARACTER SET PUNCH COMBINATION	PRINTER GRAPHICS	DECIMAL	HEXA-DECIMAL
01111101	8,5	' (quote)	125	7D
01111110	8,6	=	126	7E
01111111	8,7		127	7F
10000000	12,0,8,1		128	80
10000001	12,0,1		129	81
10000010	12,0,2		130	82
10000011	12,0,3		131	83
10000100	12,0,4		132	84
10000101	12,0,5		133	85
10000110	12,0,6		134	86
10000111	12,0,7		135	87
10001000	12,0,8		136	88
10001001	12,0,9		137	89
10001010	12,0,8,2		138	8A
10001011	12,0,8,3		139	8B
10001100	12,0,8,4		140	8C
10001101	12,0,8,5		141	8D
10001110	12,0,8,6		142	8E
10001111	12,0,8,7		143	8F
10010000	12,11,8,1		144	90
10010001	12,11,1		145	91
10010010	12,11,2		146	92
10010011	12,11,3		147	93
10010100	12,11,4		148	94
10010101	12,11,5		149	95
10010110	12,11,6		150	96
10010111	12,11,7		151	97
10011000	12,11,8		152	98
10011001	12,11,9		153	99
10011010	12,11,8,2		154	9A
10011011	12,11,8,3		155	9B
10011100	12,11,8,4		156	9C
10011101	12,11,8,5		157	9D
10011110	12,11,8,6		158	9E
10011111	12,11,8,7		159	9F
10100000	11,0,8,1		160	A0
10100001	11,0,1		161	A1
10100010	11,0,2		162	A2
10100011	11,0,3		163	A3
10100100	11,0,4		164	A4
10100101	11,0,5		165	A5
10100110	11,0,6		166	A6
10100111	11,0,7		167	A7
10101000	11,0,8		168	A8
10101001	11,0,9		169	A9
10101010	11,0,8,2		170	AA
10101011	11,0,8,3		171	AB
10101100	11,0,8,4		172	AC
10101101	11,0,8,5		173	AD
10101110	11,0,8,6		174	AE
10101111	11,0,8,7		175	AF
10110000	12,11,0,8,1		176	B0
10110001	12,11,0,1		177	B1
10110010	12,11,0,2		178	B2
10110011	12,11,0,3		179	B3
10110100	12,11,0,4		180	B4
10110101	12,11,0,5		181	B5
10110110	12,11,0,6		182	B6
10110111	12,11,0,7		183	B7
10111000	12,11,0,8		184	B8
10111001	12,11,0,9		185	B9
10111010	12,11,0,8,2		186	BA
10111011	12,11,0,8,3		187	BB
10111100	12,11,0,8,4		188	BC

Table A–1. Character Codes (Continued)

8-BIT BCD CODE	CHARACTER SET PUNCH COMBINATION	PRINTER GRAPHICS	DECIMAL	HEXA-DECIMAL
10111101	12,11,0,8,5		189	BD
10111110	12,11,0,8,6		190	BE
10111111	12,11,0,8,7		191	BF
11000000	12,0		192	C0
11000001	12,1	A	193	C1
11000010	12,2	B	194	C2
11000011	12,3	C	195	C3
11000100	12,4	D	196	C4
11000101	12,5	E	197	C5
11000110	12,6	F	198	C6
11000111	12,7	G	199	C7
11001000	12,8	H	200	C8
11001001	12,9	I	201	C9
11001010	12,0,9,8,2		202	CA
11001011	12,0,9,8,3		203	CB
11001100	12,0,9,8,4		204	CC
11001101	12,0,9,8,5		205	CD
11001110	12,0,9,8,6		206	CE
11001111	12,0,9,8,7		207	CF
11010000	11,0		208	D0
11010001	11,1	J	209	D1
11010010	11,2	K	210	D2
11010011	11,3	L	211	D3
11010100	11,4	M	212	D4
11010101	11,5	N	213	D5
11010110	11,6	O	214	D6
11010111	11,7	P	215	D7
11011000	11,8	Q	216	D8
11011001	11,9	R	217	D9
11011010	12,11,9,8,2		218	DA
11011011	12,11,9,8,3		219	DB
11011100	12,11,9,8,4		220	DC
11011101	12,11,9,8,5		221	DD
11011110	12,11,9,8,6		222	DE
11011111	12,11,9,8,7		223	DF
11100000	0,8,2		224	E0
11100001	11,0,9,1		225	E1
11100010	0,2	S	226	E2
11100011	0,3	T	227	E3
11100100	0,4	U	228	E4
11100101	0,5	V	229	E5
11100110	0,6	W	230	E6
11100111	0,7	X	231	E7
11101000	0,8	Y	232	E8
11101001	0,9	Z	233	E9
11101010	11,0,9,8,2		234	EA
11101011	11,0,9,8,3		235	EB
11101100	11,0,9,8,4		236	EC
11101101	11,0,9,8,5		237	ED
11101110	11,0,9,6,6		238	EE
11101111	11,0,9,8,7		239	EF
11110000	0	0	240	F0
11110001	1	1	241	F1
11110010	2	2	242	F2
11110011	3	3	243	F3
11110100	4	4	244	F4
11110101	5	5	245	F5
11110110	6	6	246	F6
11110111	7	7	247	F7
11111000	8	8	248	F8
11111001	9	9	249	F9
11111010	12,11,0,9,8,2		250	FA
11111011	12,11,0,9,8,3		251	FB

Table A–1. Character Codes (Continued)

8-BIT BCD CODE	CHARACTER SET PUNCH COMBINATION	PRINTER GRAPHICS	DECIMAL	HEXA-DECIMAL
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

Table B–1 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 tables figures:

HEXADECIMAL	DECIMAL
1000	4096
2000	8091
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

Table B–1. Hexadecimal-Decimal Number Conversion

	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

Table B–1. Hexadecimal-Decimal Number Conversion (Continued)

	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

Table B–1. Hexadecimal-Decimal Number Conversion (Continued)

	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

Table B–1. Hexadecimal-Decimal Number Conversion (Continued)

	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

Table B–1. Hexadecimal-Decimal Number Conversion (Continued)

	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	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F50	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F60	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F70	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F80	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F90	3984	3985	3986	3987	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

Table C–1. Summary Information For Defining Constants

Type	Implied Length In Bytes	Alignment	L(1) Modifier Range	Specification of Constant	List	Exponent Range	S Modifier Range	Truncation/ Padding Side	Code
C	as needed	byte	.1 to 256.0	characters	No			right	1
X	as needed	byte	.1 to 256.0	hexadecimal digits	No			left	0
F	4	word	.1 to 8.0	decimal value, exponent	Yes	–85 to +75	–96 to +159	left	5
H	2	half-word	.1 to 8.0	decimal value, exponent	Yes	–85 to +75	–96 to +159	left	4
E	4	word	.1 to 8.0	decimal value, exponent	Yes	–85 to +75	0 to 2L–2 (2)	right	6
D	8	double-word	.1 to 8.0	decimal value, exponent	Yes	–85 to +75	0 to 2L–2 (2)	right	7
P	as needed	byte	.1 to 16.0	decimal value	No			left	2
Z	as needed	byte	.1 to 16.0	decimal value	No			left	3
A		word	1.0 to 4.0	an expression	Yes			left	9
S	2	half-word	not allowed	an expression	Yes				10

1. The fractional portion of the length modifier indicates the number of bits; the integral portion indicates the number of bytes. Bit-length specification is not allowed for A-type constants.

2. L is length of constant (implied or specified). Note that negative scaling is not permitted.

Table D–1. Reference Summary For Assembly Instructions

Mnemonic	Name Field	Operand Field
CCW	An optional symbol	Four operands, separated by commas
CNOP	An optional symbol	Two decimal values, separated by a comma
COM	Must be blank	Ignored; should be blank
CSECT	An optional symbol or special code	Ignored; should be blank
DC	An optional symbol	One or more operands, separated by commas
DOUBL	Must be blank	Ignored; should be blank
DROP	Must be blank	One to 16 absolute expressions, separated by commas; or ".ALL"
DS	An optional symbol	One or more operands, separated by commas
DUMP	Symbol at which dump requested	Four operands, separated by commas
DUMPC	Symbol at which dump requested	Seven operands, separated by commas
DUMPE	Must be blank	Four operands, separated by commas
DUMPR	Symbol at which dump requested	Six or seven operands, separated by commas
EJECT	Must be blank	Not used; should be blank
END	Must be blank	A relocatable expression or blank
ENTRY	Must be blank	One or more relocatable symbols, separated by commas
EQU	A required symbol	An absolute or relocatable expression
EXTRN	Must be blank	One or more relocatable symbols, separated by commas
ICTL	Must be blank	A self-defining value of from 1–66, inclusive
ISEQ	Must be blank	Ignored; should be blank
LIB	Must be blank	Library update information
LIST	Must be blank	Should be blank
LTORG	An optional symbol	Ignored; should be blank
MAX	A required symbol	One or more absolute or relocatable expressions, separated by commas

Table D–1. Reference Summary For Assembly Instructions (Continued)

Mnemonic	Name Field	Operand Field
NLIST	Must be blank	Should be blank
ORG	An optional symbol	A relocatable expression
PRINT	Must be blank	One or two operands separated by a comma
PSEG	Ignored; should be blank	Segment name and/or one or more special codes
QUAL	Must be blank	An alphabetic character, or decimal digit, or a blank
RPEM	Must be blank	Must be blank
SPACE	Must be blank	A decimal value or blank
SPEM	Must be blank	Must be blank
SSEQ	Must be blank	Should be blank
START	An optional symbol	A self-defining value
TDMPL	Must be blank	Seven or eight operands, separated by commas
TDMPP	Must be blank	Seven or eight operands, separated by commas
TEQU	A required symbol	An absolute or relocatable expression
TITLE	An optional symbol	A sequence of characters
TRACB	Must be blank	Three operands, separated by commas
TRACE	Must be blank	Three operands, separated by commas
USING	Ignored; should be blank	An absolute or simply relocatable expression followed by 1–16 absolute expressions separated by commas

Appendix E. ASSEMBLY DEFINITIONS OF SYSTEM SYMBOLS

Table E–1. System Symbols Defined By Assembly

Symbol	Meaning	Symbol	Meaning
SYSPIR	For system communication	SYSTWR	Write to the typewriter
SYSTIM	Initiate timer and its interrupt	SYSTRE	Read from the typewriter
SYSIO	For system communication	SYSCOM	Overlay loading and communication
SYSDMP	Emergency dump after end of job	SYSWRS	General output (write)
SYSRET	Return to instruction of last PSW in stack	SYSRDS	General input (read)
SYSRTA	Return to specified instruction	SYSPRS	General print or punch
SYSRSL	For system communication	SYSPUN	General punch
SYSEOJ	End of job (normal)	SYSBRA	Monitor-assisted branch
SYSDEB	For system communication	SYSCTL	Control Device
SYSTRC	For system communication	SYSSTR	Set returns for devices
SYSWAT	Type PROGRAM WAITING and read CONT	SYSCLK	Initiate timer and interrupts
SYSIOI	For system communication and USERIO	SYSDTH	Terminate processing
SYSIOO	For system communication and USERIO	SYSCDP	Request emergency dump
SYSPKY	Set protection key	SYSTIN	Service console typewriter interrupt
SYSSSK	Set storage key request	SYSTCE	Record I/O instruction
SYSKMC	For system communication	SYSWRM	Count abnormal condition
SYSDTF	Define file format (system)	SYSLGD	I/O device logout
SYSRAS	For system communication	SYSLGC	I/O channel logout
SYSMOP	For system communication	SYSIPR	Initialize IOCE processor interrupt
SYSPIN	User definition for program interrupts		

Appendix F. BAL PROCEDURES

The IBM 9020 Assembly Program (assembler) accepts Basic Assembly Language (BAL) output of the JOVIAL compiler or BAL programs originated by the programmer. The assembler is called by the monitor (Figure F–1) when the compiler has completed processing or when a BAL program is detected on system input. The assembler then translates the BAL input into a form acceptable to the loader, assigns tentative storage addresses, provides for program relocation, and furnishes linkages for interprogram listing on an output unit. If execution is requested, the assembler also places the object program on the auxiliary tape for submission to the loader. When it has completed processing the BAL source program, the assembler returns control to the monitor.

No programmer instructions are necessary at assembly time for the translation of the JOVIAL compiler output. For the assembly of original BAL programs, the programmer must prepare appropriate control cards and must include these in his input deck. The $BAL control card is described in this appendix.

The symbolic analyzer, a portion of the assembler, can be requested by the programmer. It prints a list of all symbols in the processed program. The list is indexed to show the line number (in the program listing) of each symbol definition and the line numbers of all statements referring to the symbol.

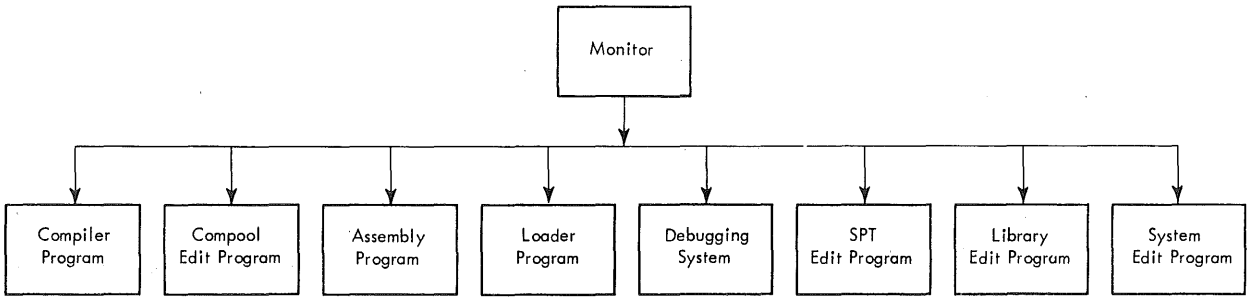

Figure F–1. Utility Programming System for the IBM 9020 Data Processing System

ASSEMBLER INPUT

Figure F–2 shows the flow of input to the assembler and the types of output produced by the assembler. The monitor, directed by control cards that accompany the input, controls the processing of all input through the 9020 Utility Programming System. When a source program is preceded by a $BAL control card, the monitor relinquishes control to the assembler for processing. Similarly, if a source program is preceded by a $JOV control card, the monitor relinquishes control to the compiler to translate the program into BAL; the program is then submitted to the assembler. Therefore, all input to the assembler is in BAL.

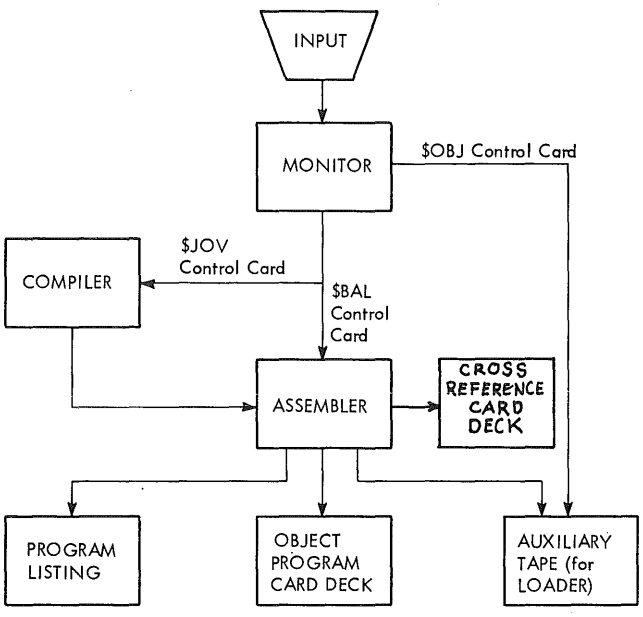

Figure F–2. Flow of Input to the Assembler

Each program must have a START card at the beginning and an END card at the end. When the assembler detects the END statement in a BAL source program, it produces a loader END card. This card will contain the address of the first executable instruction, if specified in the END statement.

A TITLE card can be inserted immediately before the START card to identify program listings and punched cards.

In addition to the $BAL control card, four assembler cards (ICTL, TITLE, START, and END) supply the assembler with information about the BAL program. The $BAL control card is described in the following text.

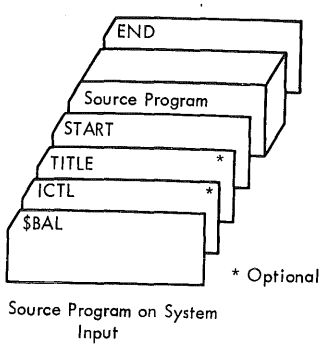

Figure F–3. Sample BAL Source Deck Accepted by the Assembler

INPUT DECK STRUCTURE

Figure F–3 shows a typical source program deck accepted by the assembler. The first control card of a source program deck must be a $BAL control card, which directs the monitor to call in the assembler.

In Figure F–3, the Input Control (ICTL) card, used to specify the card column in which the BAL statements begin, is optional. If used, it must precede the START card of the BAL source program; if not used, the assembler assumes that statements begin in column 1.

$BAL Control Card

The $BAL control card directs the monitor to transfer control to the assembler.

The format of the $BAL control card is:

Col.	1
1	6
$BAL	**LIST,PUNCH,ANALYZ,XREF,PUNCHC, LISTP,LISTD,PUNCHS,INDEX,LOAD**

Any or all of the options in the operand field may be used; if more than one option is used, the options may be listed in any sequence and must be separated by commas, with the $BAL starting in column 1 and the first option starting in column 16. LIST causes the assembler to produce a program listing, PUNCH causes an object deck to be punched, and ANALYZ causes a cross-reference listing of all symbols to be produced by the assembler's Symbolic Analyzer. If the ANALYZ option is requested, LIST is assumed. The XREF option causes the assembler to punch an XRF deck if no serious errors were encountered in the assembly. If the XREF option is selected, the LIST and ANALYZ options are assumed. If listing of the program is suppressed by an NLIST card, XRF cards will not be punched for any symbols referenced in the area that the NLIST covers. LISTD, PUNCHC, and LISTP are valid only for assemblies using a compool. LISTD forces the compool DSECT's to be listed. If LISTD is requested, LIST is assumed. PUNCHC causes referenced compool segments to be assembled, one assembly per segment, but the assemblies are not listed. PUNCHC does not cause compool segment object decks to be punched. PUNCH must be specified, in addition to PUNCHC, to obtain the punched object decks. LISTP is the same as PUNCHC but the compool segment assemblies are listed. PUNCHS and INDEX are JOVIAL options which are recognized and ignored. The LOAD option is utilized to force output of an object deck with serious errors to AUXIL for input to the LOADER or UNTE. If an unrecognizable field is encountered, LIST, PUNCH, ANALYZ is assumed.

The following is an example of a $BAL control card:

Col. 1	Col. 16
$BAL	LIST,ANALYZ

When the monitor encounters this card, it passes control to the assembler which, after translating the source program and assigning tentative storage locations, produces a program listing and a cross-reference list of symbols.

FUNCTION OF THE ASSEMBLER

The assembler translates BAL source programs and BAL output of the JOVIAL compiler into a form acceptable to the loader, producing object programs. In accomplishing this, the assembler also performs other functions, such as: program linking, error checking, assigning tentative addresses in storage for the program, establishing common storage, and producing, if requested, object decks, program listings, and cross-reference listings of all symbols.

These operations are performed in three phases: Pass I, Interlude, and Pass II. In Pass I, tentative storage addresses are assigned to each control section (starting at location 0), error checking is performed, and internal tables, such as the symbol table and the table of literals, are built. The Interlude phase adjusts the addresses by assigning consecutive storage locations to all control sections, adjusts the address values of the symbols entered in the symbol table, and produces External Symbol Dictionary (ESD) cards. In Pass II, the translation of the source program is completed, and an object deck and program listing are produced, if required. If a cross-reference listing is requested, an additional pass is required.

Program Linking

Separately assembled programs that are loaded and executed together can refer to instructions and data within one another. The assembler makes this possible by producing Relocation List Dictionary (RLD) cards and ESD cards.

The ESD cards indicate external symbols, entry points, program name, and common to the loader. (An external symbol appears as an address constant in the present program, but is defined in another program. The external symbol is thus an entry point or the program name in the program in which it is defined.) The RLD card supplies the address of the address constant containing the external symbol.

Program linkage is completed by the loader. The loader assigns to the external symbol the absolute address of its corresponding entry point or program name. The loader then uses the RLD information to place the external symbol address value in the proper address constant location.

Error Checking

The assembler examines the BAL source programs for possible errors resulting from the incorrect use of BAL. If an error is detected, the assembler prints a diagnostic message in the program listing. These messages are described in the subsection "Assembler Output."

The assembler recognizes two types of errors: possible and serious. Possible errors are those which do not prevent the assembled program from being loaded and executed. If desired, error messages for possible errors can be suppressed from the program listing. Serious error messages are always printed in the listing and prevent execution of the program.

If the LIST option was not present on the $BAL card and serious errors, or warning errors which are not suppressed, are detected, the statement plus all unsuppressed error messages will be printed. The NLISTed statements with unsuppressed errors will also be printed together with the diagnostics.

Storage Assignment

The assembler assigns tentative storage addresses to all program instructions and data, and assigns a relocation

identification number (ID) to each control section in a program. (A control section may be an entire program or a logical subdivision of a program, as designated by the programmer.) The control section ID is also assigned to all symbols within that control section. The program itself receives an ID of 01. All control sections within the program receive an ID between 02 and 254. In Pass I, each control section has addresses assigned to it starting at location 0. During the Interlude, these addresses are adjusted to give consecutive addresses to consective control sections. To do this, the length (in bytes) of the first control section (after being rounded to a multiple of eight) is added to the address of each symbol defined in the second control section. The sum of the lengths of the first and second control sections (after being rounded to a multiple of eight) is then added to the address of each symbol in the third control section, etc. This operation continued until all control sections (except dummy control sections) have been assigned consecutive storage addresses. The assembler then assigns an ID of 01 to all control sections and symbols whose addresses have been adjusted; therefore, the control sections are not individually relocatable. Dummy control sections retain their unique IDs, between 02 and 254. Common storage, a unique section defined to the assembler by a COM statement, always receives an ID of 255.

The assembler also assigns base registers to the program, and computes displacements for machine instructions that require them. The assignment of base registers is dependent upon the information contained in USING and DROP instructions in the source program. These instructions indicate the contents and numbers of the general registers that are available for use as base registers.

The tenative addresses assigned by the assembler to an object program may serve as actual storage addresses, provided those addresses are available at load time. If not, the loader can assign new addresses without altering the arrangement or referencing of the program, because the difference between the original assembler-assigned addresses and the loader-assigned addresses will remain constant.

Common Storage

The BAL source program can request the use of common storage with the COM statement. Common storage is a special and independently relocatable type of section. This enables programs in a job to define a common section and manipulate data in it. (A job is a program or group of programs that form an executable unit.) The loader assigns these common sections to the same start location. Therefore, each program may share the same data area. At load time, only one common storage area is alloted to the job. The assembler assigns every common section to location zero, and assigns an ID of 255 to each common section and to the symbols defined within that section. The loader determines the size of common storage by making it equal to the largest area required by any common section in the job.

Symbol Table Generation

The assembler generates a symbol table consisting of entries for the program name, symbols defined in the source program, and external symbols. When the assembler encounters a symbol in the name field of a source program statement, it enters the symbol in the symbol table and assigns an address value and a length attribute; where applicable, a data-type code, a scale modifier, and a qualification factor are assigned.

The address value is the tentative storage address of the leftmost byte of the field allocated to the statement, and is determined by the value of the location counter when the symbol occurs.

The length attribute is the byte size of the field named by the symbol. The data-type code is a single hexadecimal character that describes the type of data in the field named by the symbol:

Type of Data	Code
Hexadecimal (X)	**0**
EBCDIC (C)	**1**
Packed decimal (P)	**2**
Zoned decimal (Z)	**3**
Fixed-point halfword (H)	**4**
Fixed-point fullword (F)	**5**
Floating-point fullword (E)	**6**
Floating-point doubleword (D)	**7**
Instruction (I)	**8**
Address constant (A)	**9**
Base displacement (S)	**A**

The scale modifier is used with fixed-point and floating-point constants to specify the amount of internal scaling that is desired.

The qualification factor is used to distinguish between two occurrences of the same symbol with different meanings.

As the assembler constructs the symbol table, it checks that each symbol appears only once in the 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. Subsequent usages are ignored; however, an error message is printed in the program listing. An error message is also printed if a symbol referred to by a program is not entered in the symbol table.

As the assembler evaluates each expression in the program, it replaces the symbols that occur in the operand field with the tentative addresses given for them in the symbol table.

ASSEMBLER OUTPUT

The assembler assembles the BAL source program and produces diagnostic messages reflecting conditions occurring during assembly. In addition, the programmer may request several optional forms of assembler output, by specifying the desired options on the $BAL control card. This control card may be used to request an object deck, a program listing, and/or a cross-reference listing of the symbols used in the program.

All requested assembler output, including diagnostic messages, is recorded on the system output unit (either tape or printer/punch). If the system output unit is magnetic tape, the tape may be processed as a peripheral operation. If loading and execution of the program are to take place immediately, the object program is recorded on the auxiliary tape.

Auxiliary Tape

If a BAL program is to be loaded and executed immediately after assembly, the assembler places the object program on the auxiliary tape. (The auxiliary tape is used as input to the loader.) The object program is composed of ESD, TXT, RLD, END, DBG, and LIB card images, which are described in the subsection "Object Deck."

Assembler-Produced Cards

If the PUNCH option is used on the $BAL control card, an object deck is produced by the assembler. The object deck begins with a $OBJ control card and is followed by the other cards described in the subsection "Object Deck."

Object Deck

The assembler may produce seven types of cards in the object deck from the BAL source program: Text (TXT), External Symbol Dictionary (ESD), Relocation List Dictionary (RLD), End (END), Debug (DBG), Library (LIB), and Object ($OBJ). The purpose of these cards is described in the following text; the formats are described in the publications *IBM 9020 Data Processing System: Loader Program (LOADER–01) and Debugging System (DEBUGG–01) Manuals and IBM 9020 Data Processing System: Library Edit Manual (LIBEDT–01).*

TXT CARD

The TXT cards contain the text of the program in a form acceptable to the loader. Each card may contain up to 56 bytes of information. The number of bytes of information on the card and the address at which the first byte of information is to be loaded are also specified on the card.

ESD CARD

The ESD cards contain the values assigned by the assembler to the program name and to all symbols declared by EXTRN and ENTRY statements in the BAL source program. ESD cards indicate every entry point and external symbol used, thereby making it possible for programs to refer to one another. An ESD card is also produced for Common storage (an area defined to the assembler by a COM statement in the BAL source program).

RLD CARD

The RLD cards contain an entry for each address constant that contains a relocatable expression. These cards indicate to the loader those address constants that must be changed if the program is loaded at a location other than the one assigned by the assembler.

END CARD

The assembler produces the END card when it encounters the END statement in the BAL source program. This card signals the conclusion of the program to the loader. It also specifies the address of the first executable instruction in the program, if the symbolic name of the first executable instruction was originally given in the END statement. If an external symbol was used, the symbol will appear on the END card produced by the assembler.

DBG CARD

The DBG card requests execution-time debugging, and is produced from the programmer's symbolic (BAL) debugging request. As prepared for inclusion at assembly time, the debugging requests are coded in BAL (described

in the publication *IBM 9020 Data Processing System: Debugging System Manual (DEBUGG–01)* and are translated by the assembler into loader language.

LIB CARD

LIB cards are used to place compiled JOVIAL programs and/or routines on the library tape. If the compiler output contains a statement which specifies that the program or routine is to be placed on the library tape, the assembler produces the LIB card. The LIB card is further described in the publication *IBM 9020 Data Processing System: Library Edit Manual (LIBEDT–01).*

$OBJ CONTROL CARD

The $OBJ control card is produced by the assembler and placed at the beginning of the object deck. Any object deck used as input to the loader must be preceded by this card.

XREF Deck

Three types of cards may be produced by the assembler in the XREF punched deck: an XRF header card (.XRF3), an XRF symbol card (.XRF4), and an XRF trailer card (.XRF7). The purpose of these cards is described in the following text; the formats are described in the *Subprogram Design Specification for the Compool Reference Matrix Subprogram (XREF).*

.XRF3 CARD

The XRF header card contains the name of the program for which the XRF deck was punched. It informs the XREF subprogram that an XRF deck follows.

.XRF4 CARD

The XRF symbol card contains the program name and up to eight Compool data names and/or library routine names referenced by the assembled program.

.XRF7 CARD

The XRF trailer card contains the program name and the count of the number of Compool data names and library routine names referenced by the assembled program. This card informs the XREF Subprogram that the XRF deck has been completed.

Program Listing

If the LIST option is specified on the $BAL control card, a program listing is produced during Pass II of the assembly. Every statement in the program is printed as a separate line, unless the programmer makes use of the suppress option. The programmer may suppress the listing by omitting the LIST option from the $BAL control card, or part of the listing may be suppressed by using the PRINT, SPEM (suppresses printing of possible error messages), or NLIST instructions in the BAL source program.

Appendix G contains a sample of a program listing. Page 01 of the listing contains the information given in the TITLE card and any comment cards following the TITLE card. Page 02 of the listing is an external symbol listing, containing program name, type, ESD–ID number, location in storage, length (in hexadecimal), and card identification. The actual program listing begins on page 03. Each line of the program listing contains the following fields:

Code	Field
F	Flag
LOC	Location
OP	Assembled output
RR	
B–DIS	
B–DIS	
ADDR1	Effective address of operands 1 and 2
ADDR2	
LINE	Line number
SYMBOL	Symbol (BAL)
OP	Operation code (BAL)
OPERAND–COMMENTS	Operand (BAL) and comments
IDENT	Identification sequence

The relocation dictionary follows the program listing, and contains the ESD–ID of the section where the address constant was defined (ID–LOC); the ESD–ID of the defined address constant (ID–DEF); the number of bytes of the address constant (LENGTH); the sign of the address constant (SIGN): and card identification (CARD IDENT). A cross-reference listing follows the relocation dictionary listing, and is, in turn, followed by a list of undefined symbols (if any) and a summary of errors.

The fields of the program listing are described in the following text.

F FIELD

The F field is usually blank, but may contain one of the following alphabetic characters if the specified

condition exists (these flags apply to the program listing and do not affect assembly of the program):

Flag	Conditions
A	Indicates an error in sequence numbers (sequencing is checked only if an ISEQ request is made in the BAL source program).
B	Indicates an excessively long operand field (will not fit on the line).
C	Indicates that both a sequencing error and an excessively long operand field occur in the same statement.
D	Indicates a constant that has been generated by a literal (printing of literals may be suppressed by the PRINT instruction in the BAL source program).
F	Indicates that both an excessively long operand field and a constant generated by a literal occur in the same statement.
M	Indicates that the analyzer found a multi-defined symbol.
S	Indicates that the analyzer found a system symbol which was redefined in the problem program. This flag is for information only; it does not signal an error.
R	Indicates that the analyzer found a system symbol which was multi-redefined.

Whenever an entry appears in the flag field, the assembler automatically prints a legend for the flags at the end of the listing. See page 11 of the listing in Appendix G.

LOC FIELD

The LOC field contains a six-character hexadecimal representation of the address assigned to the first byte of the object code produced for the statement.

ASSEMBLED OUTPUT

The assembled output contains the object code produced for the statement. This field is divided into four subfields:

Subfield	Contents
OP	A two-character OP code in hexadecimal.
RR	A two-character subfield containing register and length information.
B–DIS	The first base register and displacement (blank if none).
B–DIS	The second base register and displacement (blank if none).

OPERAND 1 and 2 ADDRESSES

Subfield	Contents
ADDR1	Effective address of the first operand (blank if none).
ADDR2	Effective address of the second operand (blank if none).

For constants, 16 bytes are printed on each line. If the constant requires more than 16 bytes, additional lines are used to print it (unless a PRINT statement in the BAL source program has suppressed the additional lines). Constants that have a multiplicity greater than one require multiple lines for printing.

LINE FIELD

The LINE field contains a number assigned to each input statement by the assembler. This line number is used by the symbolic analyzer for identifying the location of symbol definitions and references.

SYMBOL FIELD

The SYMBOL field contains the symbol appearing in the name field of the BAL source statement. If there is none, the field remains blank.

OP FIELD

The OP field contains the operation code appearing in the operation field of the BAL source statement.

OPERAND–COMMENTS FIELD

The OPERAND–COMMENTS field contains the operand and contents of the comments field specified in the BAL source statement.

IDENT FIELD

The IDENT field contains the contents of columns 73 – 80 of the BAL source statement, which are used by the SPT edit program or for card identification and sequencing.

Cross-Reference Listing Of Symbols

When the ANALYZ option on the $BAL control card is specified, the symbolic analyzer is called to produce a cross-reference listing of all symbols used in a program. This option is valid only when a program listing is also requested. The NLISTed statements are not processed by the BAL analyzer. Consequently, any multidefined symbols, system symbol redefinitions, etc. which occur in NLISTed code will not be detected by the analyzer and the appropriate flags will not appear in the listing. The cross-reference listing, entitled "Symbolic References," appears after the relocation dictionary listing in Appendix G.

The symbolic analyzer is a part of the assembler, and the use of this option requires at least one additional pass to produce the cross-reference listing. The listing gives the line number of each symbol definition and the line numbers of all statements that refer to the symbol. Multidefined symbols and redefined system symbols are flagged. A separate listing of any undefined symbols is printed after the symbol listing. The symbolic analyzer can handle any number of references, but if more than 255 undefined symbols occur, only the first 255 are printed and a diagnostic message, stating that there are unlisted and undefined symbols, will be printed.

Diagnostics

The assembler prints a diagnostic message in the program listing for errors discovered during processing of a BAL source program. The diagnostic message(s) is printed immediately after the erroneous statement. If no LIST option was present on the $BAL card, or if an NLIST is in effect for the erroneous statement, the assembler will force printing of the statement and all unsuppressed diagnostics. The asterisks that precede each message are for ease of identification in the program listing.

Execution Permitted	Message
No	***** FIELD n HAS INVALID PUNCTUATION
No	***** FIELD n HAS FOUND INVALID CHARACTER
No	***** FIELD n HAS A SYMBOL OR NUMBER WHICH IS TOO LONG
No	***** FIELD n HAS AN EXPRESSION WHICH IS LONG OR COMPLEX
No	***** (symbol) IS AN UNDEFINED SYMBOL
No	***** FIELD n HAS AN INVALID USE OF *
No	***** FIELD n IS INVALIDLY COMPLEX RELOCATABLE
Yes	***** FIELD n HAS A VOID EXPRESSION – POSSIBLE ERROR
Yes	***** FIELD n HAS BEEN TRUNCATED – POSSIBLE ERROR
No	***** FIELD n HAS A RELOCATABLE SYMBOL WHICH IS MULTIPLIED OR DIVIDED
No	***** FIELD n HAS TOO MANY ELEMENTS IN AN EXPRESSION
No	***** (symbol) IS A MULTI-DEFINED SYMBOL
Yes	***** USE OF A PRIVILEGED OP CODE – POSSIBLE ERROR
No	***** FIELD n HAS AN EXPRESSION INVALIDLY TERMINATED
Yes	***** PSEUDO-OP IS MISPLACED – POSSIBLE ERROR
Yes	***** HALF WORD ALIGNMENT HAS OCCURRED – POSSIBLE ERROR
No	***** FIELD n HAS A RELOCATABLE IN PLACE OF ABSOLUTE
No	***** FIELD n HAS AN ERROR IN LITERAL DEFINITION
Yes	***** DC SPECIFIED BUT NO VALUE LIST – POSSIBLE ERROR
Yes	***** FIELD n HAS UNUSED REGISTER SPECIFIED FOR DROP – POSSIBLE ERROR

Execution Permitted	Message
No	***** FIELD n HAS A REGISTER EXPRESSION RELOCATABLE OR GREATER THAN 15
No	***** ADDRESS ON END CARD IN ERROR
No	***** FIELD n HAS AN INVALID EXPRESSION VALUE
Yes	***** NAME FIELD ON TITLE CARD INVALIDLY SPECIFIED – POSSIBLE ERROR
No	***** ADDRESS IN USING IS INVALID
No	***** FIELD n HAS AN INVALID REGISTER FOR USING
No	***** INVALID OP CODE – NOP GENERATED
Yes	***** FIELD n HAS A NON-FLOATING POINT REGISTER SPECIFIED – POSSIBLE ERROR
Yes	***** FIELD n HAS A NON-EVEN REGISTER SPECIFIED – POSSIBLE ERROR
No	**** FIELD n HAS ADDRESS WHICH IS NOT COVERED BY A USING
Yes	***** FIELD n HAS ADDRESS WHICH MAY BE ERRONEOUSLY ALIGNED – POSSIBLE ERROR
No	***** FIELD n HAS ADDRESS FOR WHICH BOTH AN IMPLIED AND SPECIFIED REGISTER APPLY
No	***** FIELD n HAS SYMBOL WHOSE IMPLIED LENGTH IS TOO LARGE
No	***** (symbol) SYMBOL CAUSED SYMBOL TABLE TO OVERFLOW
No	***** FIELD n HAS DIVISION WHICH RESULTED IN ZERO QUOTIENT
No	***** RLD TABLE OVERFLOWED
No	***** (symbol) HAS NOT BEEN PREVIOUSLY DEFINED
No	***** ERROR IN MODIFIER (S)

Execution Permitted	Message
No	***** ERROR IN VALUE LIST
Yes	***** FIELD n USING REGISTER 0 AS A BASE – POSSIBLE ERROR
No	***** FIELD n ATTEMPT TO USE NON-ZERO VALUE FOR REGISTER ZERO
No	***** FIELD n HAS A VALUE WHICH IS TOO LARGE
Yes	***** TRUNCATION OF CONSTANT – POSSIBLE ERROR
Yes	***** INCOMPATIBLE SCALING – POSSIBLE ERROR
Yes	***** FRACTION HAS BEEN OMITTED IN FLOAT. PT. NUMBER – POSSIBLE ERROR
Yes	***** FLOATING POINT CONSTANT IS TOO LARGE – POSSIBLE ERROR
Yes	***** CONSTANT HAS BEEN ROUNDED AND TRUNCATED – POSSIBLE ERROR
No	***** EXPONENT IS INVALID
No	***** FIELD n HAS ENTRY WHICH IS NOT IN PROGRAM OR COMMON
No	***** FIELD n HAS EXPRESSION WITH INVALID RELOCATABILITY
No	***** (symbol) SYMBOL SHOULD NOT APPEAR IN NAME FIELD
Yes	***** (symbol) IS A DUPLICATE DEFINED ENTRY POINT – POSSIBLE ERROR
Yes	***** (symbol) IS A DUPLICATE DEFINED EXTRN – POSSIBLE ERROR
Yes	***** START CARD MISSING – POSSIBLE ERROR
Yes	***** DOUBLE WORD ALIGNMENT HAS OCCURRED – POSSIBLE ERROR
No	***** CSECT TABLE HAS OVERFLOWED

Execution Permitted	Message
No	*****LITERAL TABLE HAS OVERFLOWED
No	***** ENTRY TABLE HAS OVERFLOWED
No	***** LOCATION COUNTER HAS EXCEEDED MAXIMUM
No	***** (symbol) IS NOT DEFINED, PERHAPS BECAUSE OF SYMBOL TABLE OVERFLOW
No	***** LITERAL CANNOT BE REFERENCED BECAUSE OF SYMBOL TABLE OVERFLOW
Yes	***** FIELD n RESULTED IN A CONSTANT WHICH WAS TOO LARGE – POSSIBLE ERROR
Yes	***** ATTEMPT TO DEFINE A NEW CONTROL SECTION PREVIOUSLY DEFINED – POSSIBLE ERROR
Yes	***** NO DBG CARD GENERATED – POSSIBLE ERROR (This will follow one of the following 6 messages.)
Yes	***** FIELD n HAS INVALID FORMAT SPECIFICATION – POSSIBLE ERROR
Yes	***** FIELD n HAS AN INVALID LABEL – POSSIBLE ERROR
Yes	***** FIELD n HAS AN INVALID INTEGER – POSSIBLE ERROR
Yes	***** FIELD n HAS AN INVALID ADDRESS – POSSIBLE ERROR
Yes	***** FIELD n HAS AN INVALID CONDITION SPECIFICATION – POSSIBLE ERROR
Yes	***** FIELD n HAS AN ERROR IN REGISTER SPECIFICATION – POSSIBLE ERROR
Yes	***** FRACTION PART LOST – POSSIBLE ERROR
Yes	***** VALUE SPECIFICATION MISSING – POSSIBLE ERROR
Yes	***** FLOATING POINT EXPONENT UNDERFLOW – POSSIBLE ERROR
Yes	***** (symbol) NOT USED
Yes	***** SYSTEM ERROR
Yes	ERROR IN DATA ITEM
No	FIELD n HAS TWO CONSECUTIVE QUOTES AFTER SYMBOL X.
Yes	FIELD n VALUE EXCEEDS 24 BITS. RESULT WILL BE TRUNCATED – POSSIBLE ERROR.

Error Messages

The following messages are issued by the assembler:

CONTROL CARD INVALID OR MISSING.

ASSEMBLY SKIPPED.

END CARD SUPPLIED BY ASSEMBLER ********.**

SYMBOL TABLE OVERFLOW – ANALYZER SKIPPED.

JOVIAL INPUT RECORDS MISSING – POSSIBLE TAPE ERROR.

Due to a probable hardware error, all of the BAL records output by JOVIAL were not received by BAL. A message will be typed to the operator requesting the job be rerun.

UNEXPECTED EOF OR ERROR READING COMPOOL.

Due to a hardware error, the compool tape or MLC has been mispositioned prior to performing the PUNCHC for compool segments. Rerun the job and the problem should not reoccur.

Appendix G. SAMPLE PROGRAM LISTING

```
MAIN PROGRAM FOR JOB                                                    VERSION 08/01/64   DATE 10/28/64   PAGE 01

                                        ASSEMBLY CONTROL CARDS

                                MAIN    TITLE MAIN PROGRAM FOR JOB                                          SAMPL010
                                *               FIRST PROGRAM BEGINS AT  LOCATION 65536                     SAMPL020
```

```
MAIN PROGRAM FOR JOB                                    EXTERNAL DICTIONARY   VERSION 08/01/64   DATE 10/28/64   PAGE 02

          NAME                TYPE              ESD-ID          LOCATION          LENGTH          CARD IDENT

        MAINPR                PROGRAM             01              010000          0001C0          MAIN0001
                              COMMON              FF              000000          000348          MAIN0002
        SR1                   EXTRN               02              000000                          MAIN0003
        SR2                   EXTRN               03              000000                          MAIN0003
        ROUT                  EXTRN               05              000000                          MAIN0004
        LINE                  ENTRY               01              0100D0                          MAIN0005
        CSCT1                 ENTRY               01              010000                          MAIN0005
        MAIN                  ENTRY               01              010198                          MAIN0005
```

```
MAIN PROGRAM FOR JOB                                                    VERSION 08/01/64   DATE 10/28/64   PAGE 03

F   LOC   OP RR B-DIS B-DIS   ADDR1  ADDR2   LINE SYMBOL    OP   OPERAND-COMMENTS                              IDENT

                              010000          00001 MAINPR   START X'10000'                                  SAMPL030
                                              00002          ENTRY LINE,CSCT1,MAIN.X                         SAMPL040
*****CSCT1  IS A MULTI-DEFINED SYMBOL

                                              00003          EXTRN SR1,SR2                                   SAMPL050
                                              00004 *                MAIN  PROGRAM                          SAMPL060
                                              00005          ISEQ                                            SAMPL070
  010000 05 E0                               00006 CSCT1    BALR  14,0                                      SAMPL080
                              010002          00007          USING *,14                                      SAMPL090
  010002 58 F0 E 176          010178          00008          L     15,=A(COMB)                               SAMPL100
  010006 41 10 E 0CE          0100D0          00009          LA    1,LINE                                    SAMPL110
  01000A D2 87 E 0CE E 03E    0100D0 010040   00010          MVC   LINE(136),BLANKS                          SAMPL120
  010010 D2 0B E 0D8 0 000    0100DA          00011          MVC   LINE+10(12),PNAME                         SAMPL140
*****PNAME  IS AN UNDEFINED SYMBOL

  010016 92 F1 E 0CE          0100D0          00012          MVI   LINE,C'1'                                 SAMPL150
  01001A 58 A0 E 17A          01017C          00013          L     10,=A(SR1)                                SAMPL170
  01001E 05 DA                                00014          BALR  13,10                                     SAMPL180
  010020 98 AC E 17E          010180          00015          LM    10,12,=A(5,COMB+L'TABLE*NENTRIES,SR2)     SAMPL190
  010024 05 DC                                00016          BALR  13,12                                     SAMPL200
  010026 D2 87 E 0CE E 03E    0100D0 010040   00017          MVC   LINE(136),BLANKS                          SAMPL210
  01002C D2 02 E 0D8 E 18E    0100DA 010190   00018          MVC   LINE+10(3),=C'END'                        SAMPL220
A 010032 0A 1B                              00019 EOJ      SVC   SYSEOJ                                    SAMPLIO/
A 010034 07 00                              00020 RERET    CNOP  2,4
A 010036 58 E0 E 03A          01003C        00021          L     14,SYSCT
A 01003A 05 DE                              00022          BALR  13,14
A 01003C 0001017C                           00023 SYSCT    DC    A(=A(SR1))
  010040                                    00024          CNOP  0,4                                       SAMPL260
A                                           00025          PRINT NODATA,
  010040 4040404040404040404040404040404040 00026 BLANKS   DC    9CL16' '                                  SAMPL270
                                            00027          PRINT DATA,
  0100D0                                    00028          CNOP  0,8                                       SAMPL280
  0100D0                                    00029 LINE     DS    150C                                      SAMPL290
  010166 D4C1C9D540D7D9D6C7D9C1D4           00030 PNAME    DC    C'MAIN PROGRAM'                           SAMPL300
A                                           00031          TITLE DEFINE COMMON                             SAMP
```

```
DEFINE COMMON                                                           VERSION 08/01/64   DATE 10/28/64   PAGE 04

F   LOC   OP RR B-DIS B-DIS   ADDR1  ADDR2   LINE SYMBOL    OP   OPERAND-COMMENTS                              IDENT

  000000                                    00032          COM                                             SAMPL310
                              000000        00033 COMB     EQU   *                                         SAMPL320
A                                           00034 *                SAMPLE TABLE ENTRY CHANGES WITH EACH ASSEMBLY
  000000                                    00035 TABLE    DS    CL12                                      SAMPL350
A                                           00036 *                VARIABLE NUMBER OF ENTRIES              SAMPL360
                              000014        00037 NENTRIES EQU   20                                        SAMPL360
                                            00038          TITLE SECOND CSECT                              SAMPL370
```

```
SECOND CSECT                                                    VERSION 08/01/64   DATE 10/28/64  PAGE 05

F   LOC   OP RR B-DIS B-DIS   ADDR1  ADDR2   LINE SYMBOL    OP   OPERAND-COMMENTS                        IDENT


                                            00039          QUAL  X                                      SAMPL380
  010198                                    00040 CSCT2    CSECT                                        SAMPL390
                                            00041          ORG   MAINPR.+X'2000'                        SAMPL400
*****FIELD 1 HAS EXPRESSION WITH INVALID RELOCATABILITY

  010198 05 C0                              00042 MAIN     BALR  12,0                                   SAMPL410
                        01019A              00043          USING *,12                                   SAMPL420
  01019A 47 00 0 000                        00044 SWITCH   NOP   EXIT                                   SAMPL430
*****EXIT  IS AN UNDEFINED SYMBOL

  01019E 90 2B 0 000                        00045          STM   2,11,TEMP                              SAMPL440
*****TEMP  IS AN UNDEFINED SYMBOL

                                            00046          EXTRN ROUT                                   SAMPL450
  0101A2 58 D0 E 18A        01018C          00047          L     13,=A(ROUT)                            SAMPL460
  0101A6 05 ED                              00048          BALR  14,13                                  SAMPL470
  0101A8 0000                               00049          DC    AL2(TABLE.)                            SAMPL480
  0101AA 000008                             00050          DC    AL3(SR1.+8)                            SAMPL490
  0101AD 0000002A                           00051          DC    FL4'42'                                SAMPL500
  0101B1 00
*****HALF WORD ALIGNMENT HAS OCCURRED - POSSIBLE ERROR

  0101B2 58 D0 0 000                        00052          L     13,=(ROUT)                             SAMPL510
*****FIELD 2 HAS AN ERROR IN LITERAL DEFINITION

  0101B6 05 ED                              00053          BALR  14,13                                  SAMPL520
  0101B8 0A 19                              00054 EOPG     SVC   SYSRET.                                SAMPL530
                                            00055          SSEQ                                         SAMP
                                            00056          TITLE ADD TO COMMON DECLARATION              SAMPL

ADD TO COMMON DECLARATION                                       VERSION 08/01/64   DATE 10/28/64  PAGE 06

F   LOC   OP RR B-DIS B-DIS   ADDR1  ADDR2   LINE SYMBOL    OP   OPERAND-COMMENTS                        IDENT


  C0000C                                    00057          COM                                          SAMPL
  C0000C                                    00058          DS    200F                                   SAMP
  00032C                                    00059 ENDTB    DS    F                                      SAMPL
  000330 40                                 00060 RECORD   DC    20C' '                                 SAMPL
  000331 40
  000332 40
  000333 40
  000334 40
  000335 40
  000336 40
  000337 40
  000338 40
  000339 40
  00033A 40
  00033B 40
  00033C 40
  00033D 40
  00033E 40
  00033F 40
  000340 40
  000341 40
  000342 40
  000343 40
                                            00061          TITLE END OF FIRST CSECT                     SAMPL

END OF FIRST CSECT                                              VERSION 08/01/64   DATE 10/28/64  PAGE 07

F   LOC   OP RR B-DIS B-DIS   ADDR1  ADDR2   LINE SYMBOL    OP   OPERAND-COMMENTS                        IDENT


                                            00062          QUAL                                         SAMPL
  010172                                    00063 CSCT1    CSECT                                        SAMP
*****CSCT1  IS A MULTI-DEFINED SYMBOL

                                            00064 *              DEFINE CONSTANTS                       SAMP
  010172                                    00065          LTORG                                        SAMP
  010172 0000C0000000
D 010178 00000000                                          DC    A(COMB)
D 01017C 00000C00                                          DC    A(SR1)
D 010180 00000005C00000F000000000                          DC    A(5,COMB+L'TABLE*NENTRIES,SR2)
D 01018C 00000C00                                          DC    A(ROUT)
*****ROUT  IS AN UNDEFINED SYMBOL

D 010190 C5D5C4                                            DC    C'END'


            ON CARD-ID MAIN0013             00066 EOJ      DUMP  ALPH,COMMON,COMB,TABLE+L'TABLE*NENTRIES  SAMP
                                            00067 SWITCH   DUMPC HEX,CONDMP,X'10000,X'15000',5,10 '      SAMP
*****FIELD 3 HAS INVALID PUNCTUATION
*****FIELD 3 HAS AN INVALID ADDRESS - POSSIBLE ERROR
*****NO DBG CARD GENERATED - POSSIBLE ERROR

            ON CARD-ID MAIN0014             00068          TRACE LOOPTR,MAIN.X,EOPG.X                   SAMP
                                            00069          DUMPE HEXI,EMERG,MAINPP,EOPG                 SAMP
*****EOPG  IS AN UNDEFINED SYMBOL
*****FIELD 4 HAS AN INVALID ADDRESS - POSSIBLE ERROR.
*****NO DBG CARD GENERATED - POSSIBLE ERROR



  010193                    010000          00070          END   MAINPR                                 SAMP
```

```
END OF FIRST CSECT                                  RELOCATION DICTIONARY   VERSION 08/01/64   DATE 10/28/64   PAGE 08

        LOCATION    ID-LOC    ID-DEF    LENGTH    SIGN      CARD IDENT,

          01003C      01        01         4         +       MAIN0016
          0101A8      01        FF         2         +       MAIN0016
          0101AA      01        02         3         +       MAIN0016
          010178      01        FF         4         +       MAIN0016
          01017C      01        02         4         +       MAIN0016
          010184      01        FF         4         +       MAIN0016
          010188      01        03         4         +       MAIN0017
```

```
END OF FIRST CSECT                                  SYMBOLIC REFERENCES     VERSION 08/01/64   DATE 10/28/64   PAGE 09

  FLAG   DEFINED     SYMBOL      QUAL     REFERENCES

          00026      BLANKS               00010     00017
          00033      COMB                 00008     00015     00066
    M     00006      CSCT1                00002     00063
          00040      CSCT2        X
          00059      ENDTB        X
          00019      EOJ                  00066
          00054      EOPG         X       00068
          00029      LINE                 00002     00009     00010     00011     00012     00017     00018
          00042      MAIN         X       00002     00068
          00001      MAINPR               00041     00069     00070
          00037      NENTRIES             00015     00066
          00030      PNAME
          00060      RECORD       X
          00020      RERET
          00046      ROUT         X       00047     00052
          00003      SR1                  00013     00023     00050
          00003      SR2                  00015
          00044      SWITCH       X
    S     00023      SYSCT                00021
                     SYSEOJ               00019
                     SYSRET               00054
          00035      TABLE                00015     00049     00066     00066
```

```
END OF FIRST CSECT                                  UNDEFINED SYMBOLS       VERSION 08/01/64   DATE 10/28/64   PAGE 10

                   SYMBOL      QUAL     REFERENCES

                   EOPG                 00069
                   EXIT          X      00044
                   PNAME                00011
                   ROUT
                   SWITCH               00067
                   TEMP          X      00045
```

```
END OF FIRST CSECT                                                  VERSION 08/01/64   DATE 10/28/64   PAGE 11


LEGEND FOR FLAG FIELD - F   MEANING OF FLAG

                        A   SEQUENCE NO. OF STATEMENT IS EQUAL TO OR SMALLER THAN PREVIOUS AND ISEQ WAS REQUESTED.
                        B   STATEMENT TRUNCATED ON LISTING BECAUSE THE ENTIRE OPERAND-COMMENT FIELD COULD NOT FIT ON A LINE.
                        C   BOTH A SEQUENCE NO. ERROR AND STATEMENT TRUNCATION OCCURRED ON THE LINE.
                        D   A DATA CONSTANT WAS GENERATED AS A RESULT OF A LITERAL SPECIFICATION.
                        F   BOTH STATEMENT TRUNCATION AND GENERATED OC FOR LITERAL HAVE OCCURRED.

                        M   ANALYZER HAS FOUND SYMBOL TO BE MULTI-DEFINED
                        S   ANALYZER HAS FOUND A SYSTEM SYMBOL WHICH WAS RE-DEFINED IN THE PROGRAM
                        R   ANALYZER HAS FOUND A SYSTEM SYMBOL WHICH IS MULTI-REDEFINED




                  00005 POSSIBLE ERRORS   -   00010 SERIOUS ERRORS

                                              PROGRAM CANNOT BE EXECUTED

ASSEMBLY COMPLETED. - UNSUCCESSFUL

LOADING WILL BE SUPPRESSED.
```

Table H–1. I/O Requirements

Unit	R/O*	Type	Notes
SYSTEM	R	T	
AUXIL	R	T	
SYSIN	R	T/C	Tape or Card
SYSOUT	R	T/PP	Tape or Printer/Punch
WORK 2	O	T	Requirement depends on core size vs program size
LIB	O	T	If Compool tape/MLC used, one of these is required
COMP	O	T	
WORK 1	O	T	May be used as input from JOV, SYMCOR. If Compool was used and .PUNC or PUNCHC requested, WORK 1 is required.

* R = Required, O = Optional

Table I–1. Assembler SE's vs Variables (.WORK2 Available)

Variables	Number of SE's							
	1	2	3	4	5	6	7	8
Maximum number of symbols	1024	4096	8192	8192	16384	16384	16384	20480
Maximum number of address constants	512	2048	4096	4096	8192	8192	8192	8192
Maximum number of literals	128	1024	2048	2048	4096	4096	4096	4096

Table I–2. Assembler SE' vs Variables (.WORK2 Not Available)

Variables	Number of SE's							
	1	2	3	4	5	6	7	8
Maximum number of symbols	144	719	1296	1872	2248	3023	3600	4176
Maximum number of address constants	72	359	646	934	1221	1509	1796	2048
Maximum number of literals	72	359	646	934	1221	1509	1796	2048
Approximate number of cards	478	2778	5078	7378	9678	11978	14278	16578

DISTRIBUTION LIST

Enroute Support Documentation (Non SPO-MD Documents)

NAFEC	Copies
AAT-540	5
MITRE NAFEC	2
ANA-64.A	50
ARD-140	2
AAF-360	1
ARD-140 (UK)	1

Regions	Copies
ACE-400	1
ACE-550	1
AEA-400	1
AEA-550	1
AGL-400	1
AGL-550	1
ANE-400	1
ANE-550	1
ANW-400	1
ANW-550	1
ARM-400	1
ARM-550	1
ASO-400	1
ASO-550	1
ASW-400	1
ASW-550	1
AWE-400	1
AWE-550	1

Washington	Copies
AAT-300	1
AAT-501	1
MITRE	3
AAF-40	1
ARD-160	1
ARD-160.A	1
AAF-350	1

OTHER	Copies
AAC-940F	6
U. K. London	2
U. K. Middlesex	2
ARD-57	1

ARTCC	Copies
Albuquerque	3
Atlanta	3
Boston	3
Chicago	3
Cleveland	3
Denver	3
Fort Worth	3
Houston	3
Indianapolis	3
Jacksonville	3
Kansas City	3
Los Angeles	3
Memphis	3
Miami	3
Minneapolis	3
New York	3
Oakland	3
Salt Lake City	3
Seattle	3
Washington	3

Attention: Automation Field Office