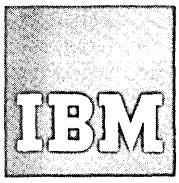
File Number S360-21
Order No. GC24-3414-7

DOS
TOS



Systems Reference Library

IBM System/360
Disk and Tape Operating Systems
Assembler Language

This reference publication contains specifications for the IBM System/360 Disk and Tape Operating Systems Assembler Language (including macro instructions and conditional assembly facilities).

The assembler language is a symbolic programming language used to write programs for the IBM System/360. The language provides a convenient means for representing the machine instructions and related data necessary to program the IBM System/360. The IBM System/360 Disk and Tape Operating Systems Assembler Programs process the language and provide auxiliary functions useful in the preparation and documentation of a program, and include facilities for processing macro instructions.

Part 1 of this publication is an introduction to the assembler language.

Part 2 describes the basic functions of the assembler language.

Part 3 describes the conditional assembly and macro facilities in the assembler language.

PREFACE

This publication is a reference manual for the programmer using the assembler language (including macro definitions and conditional assembly facilities). This publication also contains information peculiar to the IBM System/360 Disk and Tape Operating Systems for the D and F assembler.

Part 1 of this publication presents information common to all parts of the language. Part 2 contains specific information concerning the symbolic machine instruction codes and the assembler program functions provided for the programmer's use. Part 3 of this publication describes the conditional assembly and macro facilities in the assembler language.

Appendixes A through P follow Part 3. Appendixes A through F are associated with Parts 1 and 2 and present such items as a summary chart for constants, instruction listings, character set representations, and other aids to programming. Appendix G contains macro facility summary charts, and Appendix H discusses table capacities for various elements of the language. Appendix I is a sample program and assembler listing description. Appendix J is a features comparison chart of the IBM System/360 assemblers. Appendix K gives examples of the cards needed for assembler runs. Appendix L contains a description of how another version of the assembler can be included in the core-image library. Appendix M describes the output produced by the assembler. Appendix N explains the diagnostic error messages that can be issued by the assembler. Appendix O contains self-relocating program techniques. Appendix P contains sample macro definitions.

Prerequisite for a thorough understanding of this publication is a basic knowledge of IBM System/360 machine concepts. The publications most closely related to this are:

1. IBM System/360 Principles of Operation, Order No. GA22-6821.
2. IBM System/360 Disk Operating System: Data Management Concepts, Order No. GC24-3427, or IBM System/360 Tape Operating System: Data Management Concepts, Order No. GC24-3430.
3. IBM System/360 Disk Operating System: Supervisor and Input/Output Macros, Order No. GC24-5037 or IBM System/360 Tape Operating System: Supervisor and Input/Output Macros, Order No. GC24-5035.
4. IBM System/360 Disk Operating System: System Control and System Service Programs, Order No. GC24-5036 or IBM System/360 Tape Operating System: System Control and System Service Programs, Order No. GC24-5034.
5. IBM System/360 Disk Operating System: System Generation and Maintenance, Order No. GC24-5033 or IBM System/360 Tape Operating System: System Generation and Maintenance, Order No. GC24-5015.
6. IBM System/360 Disk and Tape Operating Systems Utility Macro Specifications, Order No. GC24-5042.

Titles and abstracts of other related publications are listed in the IBM System/360 Bibliography, Order No. GA22-6822.

Eighth Edition (January 1970)

This is a major revision of, and obsoletes, GC24-3414-6. All changes to the text, and small changes to illustrations, are indicated by a vertical line to the left of the change; changed or added illustrations are denoted by the symbol • to the left of the caption.

This edition applies to:

version 2, modification 5, of 360M - AS - 465 of the Tape Operating System, Assembler (D)
version 3, modification 5, of 360N - AS - 465 of the Disk Operating System, Assembler (D)
version 3, modification 7, of 360N - AS - 466 of the Disk Operating System, Assembler (F)

and to all subsequent modifications until otherwise indicated in new editions or Technical Newsletters. Changes are periodically made to the specifications herein; before using this publication in connection with the operation of IBM systems, consult the latest SRL Newsletter, Form N20-0360, for the editions that are applicable and current.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form is provided at the back of this publication for reader's comments. If the form has been removed, comments may be addressed to IBM Nordic Laboratory, Technical Communications, Box 962, S-18109 Lidingö 9, Sweden.

© Copyright International Business Machines Corporation 1968, 1969, 1970

IBM Technical Newsletter

File Number S360-21

Re: Order No. GC24-3414-7

This Newsletter No. GN33-8076

Date May 14, 1970 22/23

Previous Newsletter Nos. None

IBM SYSTEM/360 DISK AND TAPE OPERATING SYSTEMS
ASSEMBLER LANGUAGE

© IBM Corp. 1968, 1969, 1970

This Technical Newsletter, a part of release 22/23 of IBM System/360 Disk Operating System, provides replacement pages for IBM System/360 Disk and Tape Operating Systems Assembler Language, GC24-3414-7. These replacement pages remain in effect for subsequent releases unless specifically altered. Pages to be inserted and/or removed are listed below.

51, 52
61, 62
87, 88

A change to the text or a small change to an illustration is indicated by a vertical line to the left of the change; a changed or added illustration is denoted by the symbol • to the left of the caption.

Summary of Amendments

- When PRINT NOGEN is specified, generated MNOTE messages with an asterisk in the severity-code field will appear in the listing.
- One editorial change.

File this cover letter at the back of the manual to provide a record of changes.

IBM Nordic Laboratory, Technical Communications, Box 962, Lidingö 9, Sweden

PRINTED IN U.S.A.

CONTENTS

PART 1 -- INTRODUCTION TO THE ASSEMBLER LANGUAGE 7

SECTION 1: INTRODUCTION 7
Machine Features Required 7
Compatibility 8
The Assembler Language 8
Machine Operation Codes 9
Assembler Operation Codes 9
Macro Instructions 9
The Assembler Program 9
The Macro Generation and Conditional Assembly Section. . . 9
The Assembly Section 9
Programmer Aids 10
Assembler - DOS/TOS Relationships . . . 10

SECTION 2: GENERAL INFORMATION 11
Assembler Language Coding Conventions . 11
Coding Form 11
Continuation Lines 11
Statement Boundaries 11
Statement Format 13
Summary of Instruction Format . . 14
Comments Statements 14
Identification-Sequence Field . . 14
Character Set 14
Assembler Language Structure 15
Terms and Expressions. 15
Terms 15
Symbols 17
Self-Defining Terms 18
Location Counter Reference 19
Literals 20
Symbol Length Attribute Reference 21
Expressions 21
Evaluation of Expressions 22
Absolute and Relocatable Expressions 22

PART 2 -- BASIC FUNCTIONS OF THE ASSEMBLER LANGUAGE 24

SECTION 3: ADDRESSING -- PROGRAM SECTIONING AND LINKING 24
Addressing 24
Addresses -- Explicit and Implied . . 24
Base Register Instructions 24
Using -- Use Base Address Register 24
Drop -- Drop Base Register 25
Programming With the Using Instruction 26
Relative Addressing 26
Program Sectioning and Linking 27
Control Sections 27
Control Section Location Assignment 28
First Control Section 28
START -- Start Assembly 28
CSECT -- Identify Control Section 28
Unnamed Control Section 29
DSECT -- Identify Dummy Section. . 29
COM -- Define Blank Common Control Section 30
Symbolic Linkages 31
ENTRY -- Identify Entry-Point Symbol 31
EXTRN -- Identify External Symbol . . 31
Addressing External Control Sections 32
WXTRN -- Identify Weak External Symbol. 32

SECTION 4: MACHINE INSTRUCTIONS 33
Machine Instruction Statements 33
Instruction Alignment and Checking 33
Operand Fields and Subfields 33
Lengths -- Explicit and Implied . . . 34
Machine Instruction Mnemonic Codes . . . 35
Machine Instruction Examples 35
RR Format 35
RX Format 36
RS Format 36
SI Format 36
SS Format 36
Extended Mnemonic Codes 36

SECTION 5: ASSEMBLER INSTRUCTION STATEMENTS 38
Symbol Definition Instruction 38
EQU --Equate Symbol 38
Data Definition Instructions 39
DC -- Define Constant 39
Operand Subfield 1: Duplication Factor 40
Operand Subfield 2: Type 40
Operand Subfield 3: Modifiers . . 40
Operand Subfield 4: Constant . . 42
DS -- DEFINE STORAGE 48
Special Uses of the Duplication Factor 49
CCW -- Define Channel Command Word . 50
Listing Control Instructions 51
TITLE -- Identify Assembly Output . . 51
EJECT -- Start New Page 51
SPACE -- Space Listing 52
PRINT -- Print Optional Data 52
Program Control Instructions 53
ICTL -- Input Format Control 53
ISEQ -- Input Sequence Checking . . . 53
PUNCH -- Punch a Card 54
REPRO -- Reproduce Following Card . . 54
ORG -- Set Location Counter 54
LTORG -- Begin Literal Pool 55
Special Addressing Consideration . 55
CNOP -- Conditional No Operation . . 55

COPY -- Copy Predefined Source Coding 57
END -- End Assembly 57

PART 3 -- CONDITIONAL ASSEMBLY AND MACRO FACILITIES IN THE ASSEMBLER LANGUAGE 58

SECTION 6: INTRODUCTION TO THE CONDITIONAL ASSEMBLY AND MACRO FACILITIES. 58
The Macro Instruction Statement 58
The Macro Definition 58
Source Statement Libraries 59
Varying the Generated Statements. . . . 59
Variable Symbols. 59
Types of Variable Symbols. . . 59
Assigning Values to Variable Symbols 59
Global SET Symbols 60
Organization of this Part of the Publication 60

SECTION 7: HOW TO PREPARE MACRO DEFINITIONS 61
MACRO -- Macro Definition Header 61
MEND -- Macro Defintion Trailer 61
Macro Instruction Prototype
Alternate Statement Form 62
Model Statements 62
Symbolic Parameters 63
Concatenating Symbolic Parameters with Other Characters or Other Symbolic Parameters. 64
Comments Statements 65
Copy Statements 65

SECTION 8: HOW TO WRITE MACRO INSTRUCTIONS 66
Macro Instruction Operands 66
Statement Form 67
Omitted Operands 67
Operand Sublists 67
Inner Macro Instructions 68
Levels of Macro Instructions 69

SECTION 9: HOW TO WRITE CONDITIONAL ASSEMBLY INSTRUCTIONS 70
SET Symbols 70
Defining SET Symbols 70
Using Variable Symbols 70
Attributes 71
Type Attribute (T'). 72
Length (L'), Scaling (S'), and Integer (I') Attributes 72
Count Attribute (K') 73
Number Attribute (N'). 73
Assigning Integer Attributes to Symbols 73
Sequence Symbols
LCLA,LCLB,LCLC -- Define SET Symbols . . 75
SETA -- Set Arithmetic 75
Evaluation of Arithmetic Expressions. 76
Using SETA Symbols 76
SETC -- Set Character. 77
Type Attribute 77
Character Expression. 78
Substring Notation. 78
Using SETC Symbols 80
SETB -- Set Binary 80
Evaluation of Logical Expressions 81
Using SETB Symbols 82
AIF -- Conditional Branch. 82
AGO -- Unconditional Branch. 83
ACTR -- Conditional Assembly Loop Counter 84
ANOP -- Assembly No Operation. 84
Conditional Assembly Elements. 85

SECTION 10: ADDITIONAL FEATURES. . . . 86
MEXIT -- Macro Definition Exit 86
MNOTE Statement 86
Global and Local Variable Symbols . . . 87
Defining Local and Global SET Symbols 88
Using Global and Local SET Symbols. 88
Subscripted SET Symbols 90
System Variable Symbols 91
&SYSNDX -- Macro Instruction Index 91
&SYSECT -- Current Control Section 92
&SYSLIST -- Macro Instruction Operand 93
Keyword Macro Definitions and Instructions 93.1
Keyword Prototype. 94
Keyword Macro Instruction. 94
Mixed-Mode Macro Definitions and Instructions 96
Mixed-Mode Prototype 96
Mixed-Mode Macro Instruction . . . 96
Conditional Assembly Compatibility . . . 97

APPENDIX A: EXTENDED BINARY CODED DECIMAL INTERCHANGE CODE (EBCDIC) . . . 98

APPENDIX B: HEXADECIMAL-DECIMAL NUMBER CONVERSION TABLE 101

APPENDIX C: MACHINE INSTRUCTION FORMAT.106

APPENDIX D: MACHINE INSTRUCTION MNEMONIC OPERATION CODES 108

APPENDIX E: ASSEMBLER INSTRUCTIONS. . .117

APPENDIX F: SUMMARY OF CONSTANTS . . .120

APPENDIX G: MACRO FACILITY SUMMARY. . 121

APPENDIX H: DICTIONARY AND SOURCE STATEMENT SIZES 126
Part 1: Dictionaries Used in Macro Generation 126
Part 2: Macro Mnemonic Table (D Assembler only) 128
Part 3: Source Statement Complexity-Conditional Assembly and Macro Generation 128
Part 4: Source Statement Complexity-Assembler Statements 129
Part 5: Print Control Statement Listing Restrictions 130

APPENDIX I: SAMPLE PROGRAM AND ASSEMBLER LISTING DESCRIPTION 131

APPENDIX J: ASSEMBLER LANGUAGES--FEATURES COMPARISON CHART 140

APPENDIX K: CARD INPUT FOR ASSEMBLY RUNS 143

APPENDIX L: REPLACING THE CURRENT ASSEMBLER 149

APPENDIX M: OBJECT DECK OUTPUT . . . 150

APPENDIX N: DIAGNOSTIC ERROR MESSAGES 152

APPENDIX O: SELF-RELOCATING PROGRAM TECHNIQUES 163

APPENDIX P: SAMPLE MACRO DEFINITIONS 165

INDEX: 167

ILLUSTRATIONS

Figures

2-1. Coding Form 12
2-2. Punched Card Form 12
2-3. Assembler Language Structure--Machine, Assembler, and Macro Instructions. 16
3-1. Multiple Base Register Assignment 26
4-1. Extended Mnemonic Codes 37
5-1. Type Codes for Constants. . . . 40.1
5-2. Bit-Length Specification (Single Constant) 40.1
5-3. Bit Length Specification (Multiple Constants). 40.1
5-4. Bit Length Specification (Multiple Operands) 41
5-5. CNOP Alignment. 56
D-1. List of Machine Instructions by Operation Code 109
D-2. Machine Instruction Summary . . 110
G-1. Macro Facility Elements 122
G-2. Expressions 123
G-3. Attributes. 124
G-4. Variable Symbols. 125

K-1. Card Input for an Assembly . . . 143
K-2. Device Assignments 144
K-3. Operating Considerations 144
K-4. Card Input for Assembly, Linkage Editing, and Execution 145
K-5. I/O Units Used by the Tape Assembler 146
K-6. I/O Units Used by the Disk Assembler 147
K-7. Card Input for Different Variations of Assembly, Linkage Editing, and Execution 148
L-1. Card Input for Selecting Different Assembler Variants 149
L-2. Assembler Variants 149
M-1. Assembler Output Deck 150
M-2. Format of ESD, TXT, RLD, and END Cards 151
M-3. Format of the SYM Card 151.1
M-4. Format of the REP Card 151.2
N-1. Assembler Diagnostic Error Messages 152

Tables

5-1. Channel Command Word. 51

SECTION 1: INTRODUCTION

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

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

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

A program written in the assembler language will normally consist of three types of instructions: machine instructions, assembler instructions, and macro instructions. They are all coded in a language that can be interpreted by the assembler processor program. Machine instructions are transformed into machine language instruction by instruction. This language can be directly interpreted by the machine. Their functions are not described in this manual. Refer to IBM System/360 Principles of Operation (GA22-6821).

Assembler instructions are used by the assembler during processing to manipulate the source program written in the assembler language. They are described in this manual.

IBM-supported macro instructions provide easy access to the control programs supplied by the system under which the installation is running. They are described in IBM System/360 Tape Operating System Supervisor and Input/Output Macros (GC24-5035) and in IBM System/360 Disk Operating System Supervisor and Input/Output Macros (GC24-5037). The user can also write his own macro definitions to obtain easy access to precoded sections of code. Writing macro definitions is covered in this manual.

Under DOS, two assemblers -- the D assembler and the F assembler are available. TOS offers only the D assembler. Two variants of the DOS and TOS D assemblers are available. Requirements of the different assemblers are discussed below.

MACHINE FEATURES REQUIRED

A minimum of 16,384, 24,576, or 65,536 bytes of main storage as detailed below:
16,384 (16K) bytes of main storage, of which at least 10,240 contiguous bytes must be available to the assembler. This is the core requirement for the 10K variant DOS and TOS D assemblers.
24,576 (24K) bytes of main storage, of which at least 14,336 contiguous bytes must be available to the assembler. This is the core requirement for the 14K variant DOS and TOS D assemblers.
65,536 (64K) bytes of main storage, of which at least 45,056 contiguous bytes must be available to the assembler. This is the core requirement for the 44K DOS F assembler.

NOTE: Additional storage, available to any of the assemblers, is used to expand assembler tables. For details on how to call a specific assembler see Appendix K and the DOS and TOS System Generation publications (listed in Preface).

Standard instruction set
One I/O Channel (either multiplexor or selector)
One Card Reader (1442N1, 2501, 2520B1, or 2540)[1]
One Card Punch (1442N1, 1442N2, 2520, or 2540)[1], if punched output is desired
One Printer (1403, 1404 - continuous forms only, or 1443)[1], if a printed listing is desired
One 1052 Printer-Keyboard
One 2311 or 2314 Disk Storage Drive. This has the DOS resident system pack.
or
One 2400-series Magnetic Tape Unit (either 7-track or 9-track). This has the TOS resident system.
Three work files. Under the DOS D Assembler, 10K Variant: either three disk storage extents or three magnetic tape units. The devices used must be all of the same type; i.e., three magnetic tape units, three 2311 disk storage extents, or three 2314 disk storage extents. Under the DOS D Assembler, 14K Variant, and the DOS F Assembler: any combination of disk storage extents and/or magnetic tape units. The disk storage devices used need not be of the same type as that of SYSRES.
Under the TOS D Assembler, 10K and 14K Variants: three magnetic tape units. The devices used must be of the same type as that of SYSRES.

[1] A 2400-series Magnetic Tape Unit may be substituted for this device. (It may be 7-track or 9-track. If 7-track is used the data conversion feature is required and the tape must be set converter on, translator off, odd parity.) The 1052 Printer-Keyboard must be operable if device assignment is tape.

The allowable disk storage devices are the 2311 and 2314 Disk Storage Drives. The allowable magnetic tape units are 2400-series Magnetic Tape Units (either 7-track or 9-track: if 7-track, the data conversion feature is required and the tape must be set converter on, translator off, odd parity).

Under the DOS D Assembler, 10K Variant, only the system source statement library is supported. Under the DOS D Assembler, 14K Variant, and the DOS F Assembler, a private source statement library is supported. Under the TOS D Assembler, 10K and 14K Variants, the standard private library is supported. The device used for the private library must be of the same type as that of SYSRES.

For the 10K DOS and the 14K TOS D assemblers, the assemble-and-execute option is an alternative to the DECK option; both are not supported for the same assembly. For the 14K DOS D assembler and for the F assembler, both options are supported in the same assembly. If the assemble-and-execute option is chosen, SYSLNK is a 2400-series Magnetic Tape Unit (9-track or 7-track with the data conversion feature) for the tape-resident system, or a 2311 or 2314 Disk Storage extent (which may be on the system resident device) for the disk-resident system.

If, under the F assembler and the 14K DOS D assembler LINK or CATAL options are chosen, the I/O requirements for both must be met.

COMPATIBILITY

Within the Disk and Tape Operating Systems the assemblers can be used on System/360 Models 30, 40, 50, 65, and 75, provided that main storage and input/output requirements are satisfied. The assemblers (disk and tape) will both accept the same source language input and produce identical object output.

The IBM System/360 Disk and Tape Operating Systems Assembler assembles source programs written in the IBM System/360 Basic Programming Support Basic Assembler Language, the Basic Programming Support Assembler (8K Tape) Language, the IBM 7090/7094 Support Package for IBM System/360 Assembler Language, and the IBM System/360 Basic Operating System (8K Disk) Language, with the following exceptions:

1. The XFR assembler instruction, which is considered an invalid mnemonic operation code in DOS/TOS Operating Systems is not allowed.
2. Additional cards may be required in macro definitions (if used by the source program) to satisfy DOS/TOS Operating Systems macro requirements.
3. System macro instructions are changed, where necessary, to conform with the proper DOS/TOS requirements.
4. An MNOTE assembler instruction whose operand entry consists solely of a message enclosed in apostrophes is given a severity code of one.
5. AIF operand entries must not contain explicit binary zeros or ones.

The DOS/TOS D assembler language is a subset of the Operating System assembler language. The DOS F assembler language implements the full Operating System assembler language with the exception of DXD and CXD statements and the Q-type (DC or DS) constant. Source programs written in DOS/TOS assembler language will be acceptable to the Operating System assemblers provided that system macro instructions are changed, where necessary, to conform with the proper Operating System requirements. Erroneous source input may be treated somewhat differently by D and by F.

V-type address constants which reference external data may be invalid under the Operating System Assembler if the program is segmented into an overlay structure.

Note: The assignment, size, and ordering of literal pools may differ among the assemblers.

Differences in conditional assembly instructions for IBM System/360 assemblers are described in Section 10 of this publication.

THE ASSEMBLER LANGUAGE

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

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

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

Machine Operation Codes

The assembler language provides mnemonic machine instruction operation codes for all machine instructions in the IBM System/360 Universal Instruction Set, and extended mnemonic operation codes for the conditional branch instruction.

Assembler Operation Codes

The assembler language also contains mnemonic assembler instruction operation codes, used to specify auxiliary functions to be performed by the assembler program. These are instructions to the assembler program itself and, with a few exceptions, do not result in the generation of any machine language code by the assembler program. Certain assembler instructions, i.e., conditional assembly instructions, affect the order of source statement assembly and macro generation or the content of generated instructions.

Macro Instructions

The assembler language enables the programmer to define and use macro instructions. Macro instructions are represented by an operation code which, in turn, actually stands for a sequence of machine and/or assembler instructions that accomplish the desired function.

Macro instructions used in preparing an assembler language source program fall into two categories: system macro instructions, provided by IBM, which relate the object program to components of the Basic Operating System, and macro instructions created by the programmer specifically for use in the program at hand, or for incorporation in a library, available for future use.

Programmer-created macro instructions are used to simplify the writing of a program and/or to ensure that a standard sequence of instructions is used to accomplish a desired function.

For instance, the logic of a program may require the same instruction sequence to be executed again and again. Rather than code this entire sequence each time it is needed, the programmer creates a macro instruction to represent the sequence, and then each time the sequence is needed, the programmer simply codes the macro instruction statement. During assembly, the sequence of instructions represented by the macro instruction is inserted in the object program.

Part 3 of this publication discusses the conditional assembly and macro facilities.

THE ASSEMBLER PROGRAM

The assembler program, also referred to as the "assembler," processes source statements written in the assembler language. The assembler is separated into an assembly section and a conditional assembly and macro generation section.

The Macro Generation and Conditional Assembly Section

Before source statements can be translated into actual machine language, macro instructions and conditional assembly statements within the source program must be processed. The source program is read. Any programmer macro definitions which appear before the main portion of the program are stored for use when the macro is referenced. (System macro definitions are retrieved from the source statement library and handled in the same way.)

The main portion of the program is then processed. Whenever macro generation or conditional assembly is required, the generated or conditionally assembled text is inserted in the original source program. The resultant augmented source program is ready for input to the assembly section.

The Assembly Section

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

The amount of main and secondary storage allocated to the assembler program for use during processing determines the maximum

number of certain language elements that may be present in the source program. For a discussion of these dependencies, see Appendix H.

PROGRAMMER AIDS

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

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

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

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

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

The linking facilities of the assembler language and program allow symbols to be defined in one assembly and referred to in another, thus effecting a link between separately assembled programs. This permits reference to data and/or transfer of control between programs. A discussion of sectioning and linking is in Section 3 under Program Sectioning and Linking.

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

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

ASSEMBLER - DOS/TOS RELATIONSHIPS

The assembler program is a component of IBM disk and tape operating systems and functions under their control. DOS/TOS provides the assembler with input/output, library, and other services needed in assembling a source program. In a like manner, the object program produced by the assembler will normally operate under control of DOS/TOS and depend on it for input/output and other services. In writing the source program, the programmer must include statements requesting the desired functions from DOS/TOS. (See the Supervisor and Input/Output Macros publications listed in the Preface.)

This section presents information about assembler language coding conventions, assembler source statement structure, addressing, and the sectioning and linking of programs.

ASSEMBLER LANGUAGE CODING CONVENTIONS

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

Coding Form

A source program is a sequence of source statements that are punched into cards. A standard assembler card is shown in Figure 2-2. These statements may be written on the standard coding form, X28-6509 (Figure 2-1), provided by IBM. One line of coding on the form is punched into one card. The vertical columns on the form correspond to card columns.

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

The body of the form (Figure 2-1) is composed of two fields: the statement field, columns 1-71, and the identification-sequence field, columns 73-80. The identification-sequence field is not part of a statement and is discussed following the subsection Statement Format.

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

Continuation Lines

When it is necessary to continue a statement on another line the following rules apply.

1. Enter any nonblank character in the continuation column (end column plus one) of the statement line.
2. Continue the statement on the next line, starting in the continue column. Columns to the left of the continue column must be blank.

One continuation line is allowed for the D assembler and two continuation lines are allowed for the F assembler, except for source macro-instructions and macro prototype statements, which may have more than one continuation line (see Part 3).

Statement Boundaries

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

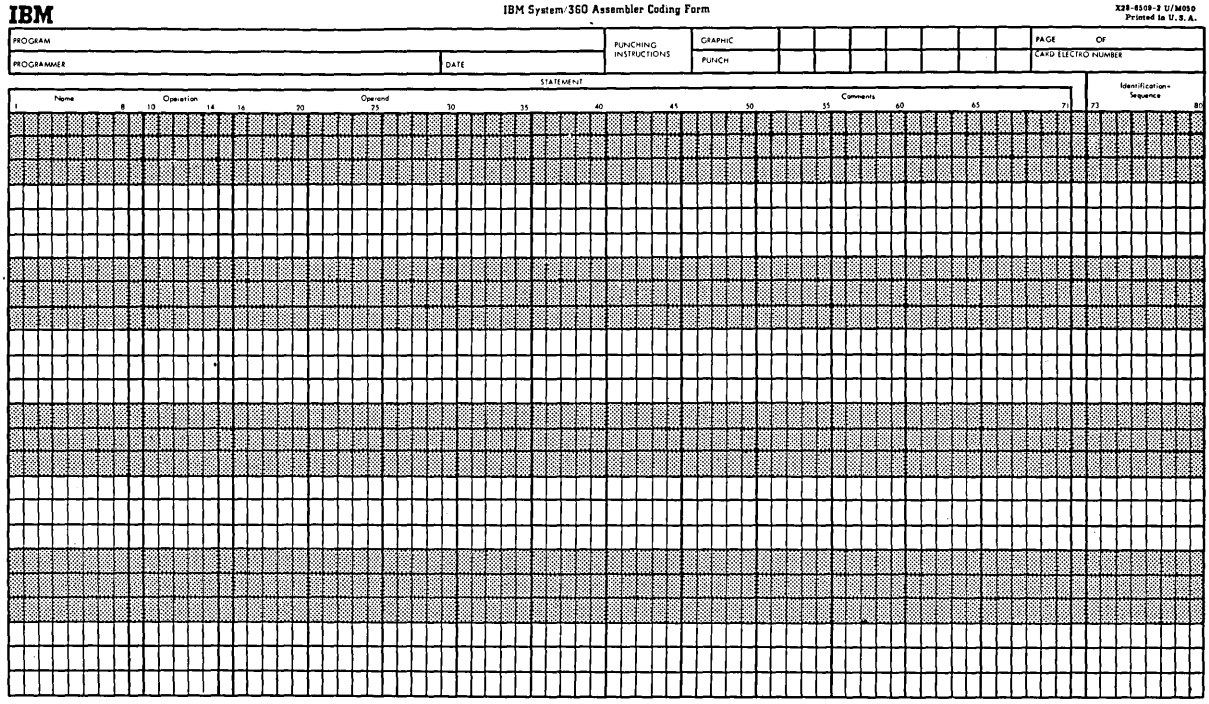


Figure 2-1. Coding Form

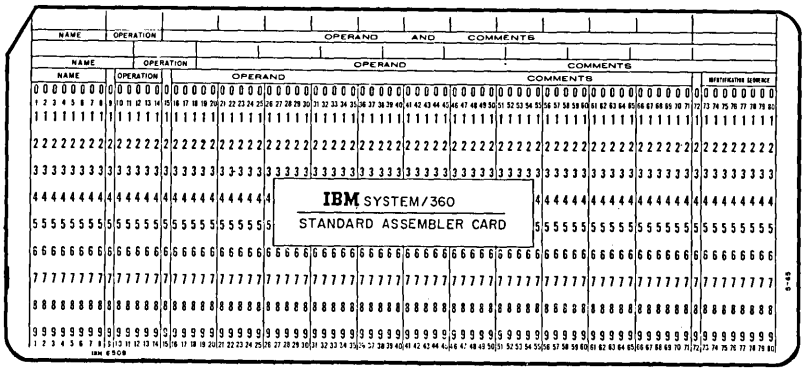


Figure 2-2. Punched Card Form

Statement Format

There are two types of statements--instructions and comments.

Instructions may consist of one to four entries in the statement field. They are, from left to right: a name entry, an operation entry, an operand entry, and a comments entry. These entries must be separated by one or more blanks, and must be written in the order stated. Total statement size is limited to 187 characters. If this limit is exceeded, the assembly listing may be incorrect for that statement.

The coding form (Figure 2-1) is ruled to provide an eight-character name field, a five-character operation field, and a 56-character operand and/or comments field.

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

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

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

Name Entries: The name entry is a symbol, eight characters or fewer, created by the programmer to identify a statement. A name entry is usually optional, but, if present, must be entered with the first (or only) character appearing in the begin column. If the begin column is blank, the assembler program assumes no name has been entered. Blanks must not appear within a name entry, whether the symbol was introduced directly by the programmer or indirectly by conditional assembly or macro generation.

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

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

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

Operands must be separated by commas. Blanks must not intervene between operands and the commas that separate them.

The operands may not contain embedded blanks except as follows:

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

Comments Entries: Comments are descriptive items of information about the program that are to be inserted in the program listing. All 256 valid characters, including blanks, may be used in writing a comment. The entry cannot extend beyond the end column (normally column 71), and a blank must separate it from the operand.

In instructions where an operand entry is optional but not present and a comments entry is desired, the absence of the operand entry must be indicated by a comma preceded and followed by one or more blanks, as follows:

```
┌--------┬----------┬----------------------┐
|Name    |Operation |Operand               |
├--------┼----------┼----------------------┤
|        |CSECT     |,  COMMENT            |
|        |  .       |                      |
|        |  .       |                      |
|        |  .       |                      |
|        |END       |,  COMMENT            |
└--------┴----------┴----------------------┘
```

Instruction Example: The following example illustrates the use of name, operation, operand, and comments entries. A compare

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

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

Summary of Instruction Format

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

Every statement requires an operation entry. Name and comment entries are optional. Operand entries are required for all machine instructions and most assembler instructions.

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

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

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

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

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

Comments Statements

Comments statements are used to include a programmer's notes on an assembly listing. (These notes can be helpful during debugging and maintenance of a program.) Comments statements have no effect in the assembled program; they are only printed in the assembly listing and, therefore, may appear at any point. Extensive notes, or comments, may be written by using a series of comments statements.

There are two types of comments statements. One type, written with an asterisk (*) in the begin column, is used for comments on the source program. The other type, written with a period in the begin column and followed by an asterisk, is used for comments on a macro-definition. This type is further described in Section 7.

An example of the comments statement is:

Name	Operation	Operand	
*THIS COMMENT IS CONTINUED ON ANOTHER LINE.			X

Identification-Sequence Field

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

To aid in keeping source statements in order, the programmer may code an ascending sequence of characters in this field or a portion of it. These characters are punched into their respective cards, and, during assembly, the programmer may request the assembler to verify this sequence by use of the Input Sequence Checking (ISEQ) assembler instruction. This instruction is discussed in Section 5 under Program Control Instructions.

Character Set

Source statements are written using the following characters:

Letters A through Z, and $, #, @

Digits 0 through 9

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

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

may appear between paired apostrophes, in comments, and in macro instruction operands.

ASSEMBLER LANGUAGE STRUCTURE

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

A source statement is composed of:

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

A name entry is:

- A symbol.

An operation entry is:

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

An operand entry is:

- One or more operands composed of one or more expressions. An expression is composed of a term or an arithmetic combination of terms. In general, an operand entry should contain 50 or fewer terms (see Appendix H).

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

Figure 2-3 depicts this structure. Terms shown in Figure 2-3 are classed as absolute or relocatable. Terms are absolute or relocatable due to the effect of program relocation upon them. (Program relocation is the loading of the object program into storage locations other than those originally assigned by the assembler program.) A term is absolute if its value does not change upon relocation. A term is relocatable if its value changes upon relocation.

The following subsection, Terms and Expressions, discusses these items as outlined in Figure 2-3.

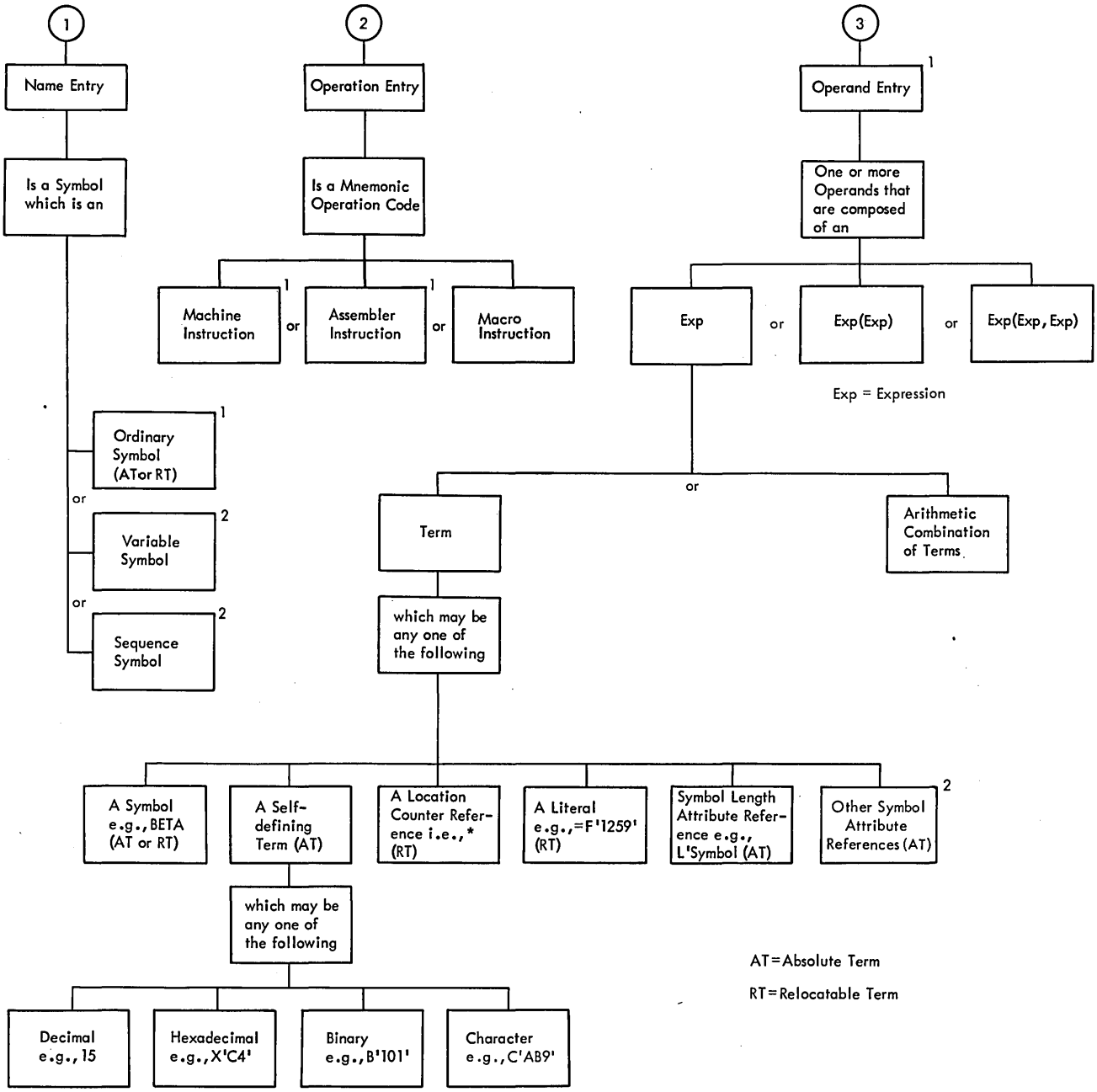
TERMS AND EXPRESSIONS

TERMS

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

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

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



[1] May be generated by combination of variable symbols and assembler language characters. (Conditional assembly only)

[2] Conditional assembly only.

Figure 2-3. Assembler Language Structure--Machine, Assembler, and Macro Instructions

Symbols

A symbol is a character or combination of characters used to represent locations or arbitrary values. Symbols, through their use in name fields and in operands, provide the programmer with an efficient way to name and reference a program element. There are three types of symbols:

1. Ordinary symbols.
2. Variable symbols.
3. Sequence symbols.

Ordinary symbols consist of one to eight letters and/or numbers, the first of which must be a letter. Such symbols are used to identify machine locations or arbitrary values. In the following sections, the occurrence of symbol refers to this type of term. Absolute symbols are ordinary symbols whose values do not change upon program relocation. Relocatable symbols are ordinary symbols whose values change upon relocation.

The following are valid ordinary symbols:

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

It is advisable to avoid using symbols beginning with IJ; they may conflict with IOCS symbols (which begin with IJ).

It is also advisable to avoid using symbols which are identical to a file name (name field) in a DTF statement with a single character suffix. For example, for the file name RECIN, IOCS generates the symbols: RECIN1, RECIN2, RECIN3, etc.

The following ordinary symbols are invalid, for the reasons noted:

256B	First character is not alphabetic.
RECORDAREA2	More than eight characters.
BCD*34	Contains a special character - an asterisk.
IN AREA	Contains a blank.

Variable symbols must begin with an ampersand (&) followed by one to seven letters and/or numbers, the first of which must be a letter. Variable symbols are used within the source program or macro definition to allow different values to be assigned to one symbol. A complete discussion of variable symbols appears in Part 3.

Sequence symbols consist of a period (.) followed by one to seven letters and/or numbers, the first of which must be a letter. Sequence symbols are used to indicate the position of statements within the source program or macro definition. Through their use the programmer can vary the sequence in which statements are processed by the assembler program. (See the complete discussion in Part 3).

NOTE: Sequence symbols and variable symbols are used only for the macro language and for conditional assembly. Programmers who do not use these features need not be concerned with these symbols.

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

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

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

A symbol is said to be defined when it appears as the name of a source statement. (A special case of symbol definition is discussed in Section 3, under "Program Sectioning and Linking").

Symbol definition also involves the assignment of a length attribute to the symbol. (The assembler maintains an internal table - the symbol table - in which the values and attributes of symbols are kept. When the assembler encounters a symbol in an operand, it refers to the table for the values associated with the symbol.) The length attribute of a symbol is the length, in bytes, of the storage field whose address is represented by the symbol. For example, a symbol naming an instruction that occupies four bytes of storage has a length attribute of 4. Note that there are exceptions to this rule; for example, in the case where symbol has been defined by an equate to location counter value (EQU *)

or to a self-defining term, the length attribute of the symbol is 1. These and other exceptions are noted under the instructions involved. The length attribute is never affected by a duplication factor.

PREVIOUSLY DEFINED SYMBOLS: The assembler language requires that symbols appearing in the operand entry of some instructions be previously defined. This simply means that the symbols, before their use in an operand, must have appeared as the name entry of a prior statement. For example:

```
          .
          .
          .
SYM1   MVC   A,B
SYM2   EQU   SYM1
          .
          .
          .
```

would be a valid sequence of coding. The same two instructions in reverse order would be invalid.

GENERAL RESTRICTIONS ON SYMBOLS: A symbol may be defined only once in an assembly. While the same symbol may appear as the name of two or more statements before macro generation and conditional assembly, only one such statement should be generated. In addition, a symbol may be used in the name field more than once as a control section name (i.e., defined in the START, CSECT, or DSECT assembler statements described in Section 3) because the coding of a control section may be suspended and then resumed at any subsequent point. The CSECT or DSECT statement that resumes the section must be named by the same symbol that initially named the section; thus, the symbol that names the section must be repeated. Such usage is not considered to be duplication of a symbol definition.

Self-Defining Terms

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

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

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

USING SELF-DEFINING TERMS: Self-defining terms are the means of specifying machine values or bit configurations without equating the values to symbols and using the symbols. Self-defining terms may be used to specify such program elements as immediate data, masks, registers, addresses, and address increments.

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

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

Hexadecimal Self-Defining Term: A hexadecimal self-defining term is a sequence of one to six hexadecimal digits. The digits must be enclosed in single apostrophes and preceded by the letter X: X'C49'.

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

The hexadecimal digits and their bit patterns are as follows:

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

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

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

represented. Padding with binary zeros is on the left.

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

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

```
|Name   |Operation |Operand                |
|-------|----------|-----------------------|
|ALPHA  |TM        |GAMMA,B'10101101'      |
```

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

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

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

For each apostrophe or ampersand desired in a character term, two apostrophes or ampersands must be written. For example, the character value A'# would be written as C'A''#', while an apostrophe followed by a blank and another apostrophe would be written as C''' '''.

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

Location Counter Reference

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

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

The Location Counter setting can be controlled by using the START and ORG assembler instructions, which are described in Sections 3 and 5, respectively. The counter affected by either of these assembler instructions is the counter for the control section in which they appear. The maximum value for the Location Counter is $2^{24}-1$.

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

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

predefined symbol, with the exception of the EQU and ORG assembler instructions.

Literals

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

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

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

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

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

A literal may not be specified in an address constant (see Section 5, DC--Define Constant). A literal may not be specified in a shift instruction or an I/O instruction (HIO,TIO,SIO).

A literal may not have an explicit base or an explicit index when specified in an instruction.

The instruction coded below shows one use of a literal.

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

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

NOTE: If the literal operand is a self-defining term (X,C,B, or D), and the = is omitted the statement may assemble without error (see Using Self-Defining Terms).

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

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

The method of describing and specifying a constant as a literal is nearly identical to the method of specifying it in the operand of a DC assembler instruction. The major difference is that the literal must start with an equal sign (=), which indicates to the assembler that a literal follows. See the discussion of the DC assembler instruction operand format (Section 5) for the means of specifying a literal. The type of literal designated in an instruction is not checked for correspondence with the operation code of the instruction.

Some examples of literals are:

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

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

The programmer may also specify that multiple literal pools be created. However, the sequence in which literals are ordered within the pool is controlled by the assembler. Further information on positioning the literal pool(s) is in Section 5 under LTORG--BEGIN LITERAL POOL.

Duplicate Literals: If duplicate literals occur within one literal pool, only one literal is stored. Literals are considered duplicates only if their specifications are identical. A literal will be stored, even if it appears to duplicate another literal, if it is an A-type address constant containing any reference to the Location Counter.

The following examples illustrate the foregoing rules:

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

XL3'0' }
       } Both are stored
HL3'0' }

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

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

Symbol Length Attribute Reference

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

```
L'BETA
```

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

For ease in following the example, the length attributes of A1 and B2 are mentioned.

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

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

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

Note: The length attribute of * is equal to the length of the instruction in which it appears, except in an EQU to * instruction where the length attribute is 1.

EXPRESSIONS

Expressions, which are used in coding operand entries for assembler language statements, are composed of either a single term or an arithmetic combination of terms (see Figure G-2). Arithmetically combined terms, enclosed in parentheses, may be used in combination with terms outside the parentheses. For example:

```
14+BETA-(GAMMA-LAMBDA)
```

When terms in parentheses occur in combination with other terms, like (GAMMA-LAMBDA) in the example, the parenthesized terms are reduced first to a single value. This value may be absolute or relocatable, depending on the combination of terms. This value then is used in reducing the rest of the combination to another single value.

Parenthesized terms may be included within another set of terms in parentheses. For example:

```
A+B-(C+D-(E+F)+10)
```

This expression has two levels of parentheses. A level of parentheses is a left parenthesis and its matching right parenthesis. One level of parentheses surrounds E+F. The next higher level of parentheses surrounds C+D-(E+F)+10. The innermost set of terms in parentheses (the lowest level) is evaluated first.

The following are examples of valid expressions:

```
*                       BETA*10
AREA1+X'2D'             B'101'
*+32                    C'ABC'
N-25                    29
FIELD+332               L'FIELD
FIELD                   LAMBDA+GAMMA
(EXIT-ENTRY+1)+GO       TEN/TWO
=F'1234'
ALPHA-BETA/(10+AREA*L'FIELD)-100
A*(A*(A*(A+1)+3*(B-3)))
```

The rules for coding expressions are:

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

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

3. An expression may not consist of more than 16 terms.

4. An expression may not have more than five levels of parentheses.

5. A multi-term expression may not contain a literal.

Evaluation of Expressions

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

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

1. Each term is given its value.

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

3. Every expression is computed to 32 bits, and then truncated to the rightmost 24 bits.

4. Division always yields an integer result; any fractional portion of the result is dropped. E.g., 1/2*10 yields a zero result, whereas 10*1/2 yields 5.

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

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

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

NOTE: In A-type address constants, the full 32 bit final result is truncated on the left to fit the specified or implied length of the constant.

Absolute and Relocatable Expressions

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

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

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

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

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

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

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

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

The pairing of relocatable terms (with opposite signs and the same relocatability attribute) cancels the effect of relocation. Therefore the value represented by the paired terms remains constant, regardless of program relocation. For example, in the absolute expression A-Y+X, A is an

absolute term, and X and Y are relocatable terms with the same relocatability attribute. If A equals 50, Y equals 25, and X equals 10, the value of the expression would be 35. If X and Y are relocated by a factor of 100 their values would then be 125 and 110. However, the expression would still evaluate as 35 (50-125+110=35).

An absolute expression reduces to a single absolute value.

The following examples illustrate absolute expressions. A is an absolute term; X and Y are relocatable terms with the same relocatability attribute.

```
A-Y+X
A
A*A
X-Y+A
*-Y   (a reference to the Location Counter
       must be paired with another relocata-
       ble term from the same control sec-
       tion, i.e., with the same relocatabil-
       ity attribute)
```

Relocatable Expressions: A relocatable expression is one whose value would change by n if the program in which it appears is relocated n bytes away from its originally assigned area of storage.

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

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

2. All the relocatable terms but one must be paired. Pairing is described in Absolute Expression.

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

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

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

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

The following examples illustrate relocatable expressions. A is an absolute term, W and X are relocatable terms with the same relocatability attribute, Y is a relocatable term with a different relocatability attribute.

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

SECTION 3: ADDRESSING -- PROGRAM SECTIONING AND LINKING

ADDRESSING

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

ADDRESSES -- EXPLICIT AND IMPLIED

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

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

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

BASE REGISTER INSTRUCTIONS

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

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

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

USING -- Use Base Address Register

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

Name	Operation	Operand
A sequence symbol or not used	USING	From 2-17 expressions of the form v,r1, r2,r3,...,r16

Operand v must be an absolute or relocatable expression with a value ranging from -2^{24} to $+2^{24}-1$. No literals are permitted. Operand v specifies a value that the assembler can use as a base address. The other operands must be absolute expressions. The operand r1 specifies the general register that can be assumed to contain the base address represented by operand v. Operands r2, r3, r4, . . . 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 assembler 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, and wishes the assembler to compute displacement from this value, the assembler must be told the new value by means of another USING statement. In the following sequence the assembler first assumes that the value of ALPHA is in register 9. The second statement then causes the assembler to assume that ALPHA+1000 is the value in register 9.

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

If the programmer has to refer to the first 4096 bytes of storage, he can use general register 0 as a base register subject to the following conditions:

1. The value of operand v must be either an absolute or relocatable zero or simply relocatable.
2. Register 0 must be specified as operand r1.

The assembler assumes that register 0 contains zero. Therefore, regardless of the value of operand v, it calculates displacements as if operand v were absolute or relocatable zero. The assembler also assumes that subsequent registers specified in the same USING statement contain 4096, 8192, etc.

NOTE: If register 0 is used as a base register, the program is not relocatable, despite the fact that operand v may be relocatable. The program can be made relocatable by:

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

DROP -- Drop Base Register

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

Name	Operation	Operand
A sequence symbol or not used	DROP	Up to 16 absolute expressions of the form r1,r2, r3,...,r16

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

Name	Operation	Operand
	DROP	7,11

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

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

PROGRAMMING WITH THE USING INSTRUCTION

The USING (and DROP) instructions may be used anywhere in a program, as often as needed, to indicate the general registers that are available for use as base registers and the base address values the assembler may assume each contains at execution time. Whenever an address is specified in a machine instruction statement, the assembler determines whether there is an available register containing a suitable base address. The USING instruction establishes addressability at assembly time, assuming that the registers assigned as base registers have been loaded with correct base addresses. Any reference to relocatable or absolute terms, which are to be assembled into the base-displacement form, such as names in the operand of a machine instruction or S-type address constant, must come after the pertinent USING instruction that makes the terms addressable. References to terms relocatable or otherwise in the operand of an A-type or Y-type address constant do not have to be preceded by a USING statement. A register is considered available for a relocatable address if it was assigned a relocatable value that is in the same control section as the address. A register assigned an absolute value is available for addressing absolute locations only. In either case the base address is considered suitable only if it is less than or equal to the address of the item to which the reference is made. The difference between the two addresses may not exceed 4095 bytes. In calculating the base register to be used, the assembler always uses the available register giving the smallest displacement. If there are two registers with the same value, the highest numbered register is used.

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

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

In Figure 3-1, the BALR and LM instructions load registers 2-5. The USING instruction indicates to the assembler that these registers are available as base registers for addressing a maximum of 16,384 consecutive bytes of storage, beginning with the location named HERE. The number of addressable bytes may be increased or decreased by altering the number of registers designated by the USING and LM instructions and the number of address constants specified in the DC instruction.

Note: Care must be taken when assigning base registers to avoid using, except under special circumstances:

1. General registers 0, 1, 13, 14 and 15, as they are used by the system.
2. Any register used explicitly or implicitly by a machine instruction.

RELATIVE ADDRESSING

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

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

Figure 3-1. Multiple Base Register Assignment

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

PROGRAM SECTIONING AND LINKING

It is often convenient, or necessary, to write a large program in sections. The sections may be assembled separately, then combined subsequently into one program. The assembler provides facilities for creating multisectioned programs and symbolically linking separately assembled programs or program sections.

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

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

CONTROL SECTIONS

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

To the assembler, there is no such thing as a program; instead, there is a source module, which consists of one or more control sections. (However, the terms source module and program are often used interchangeably.) An unsectioned program is treated as a single control section. To the linkage editor, there are no programs, only control sections that must be fashioned into one or more phases.

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

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

There is a limit to the number of external symbol dictionary entries. The total number of control sections (named, unnamed, and common control sections), dummy sections, unique symbols in EXTRN and WXTRN statements, and V-type address constants may not exceed 255. (The V-type address constant is described in Section 5 under DC -- Define Constant; the other external symbols are described in this section.) If the same symbol appears both in V-type address constant and in the name field of a START, CSECT, or DSECT statement, it is counted as two symbols.

Control Section Location Assignment

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

FIRST CONTROL SECTION

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

1. The initial value of its location counter may be specified as an absolute value.
2. It normally contains the literals requested in the source module, although their positioning can be altered. This is further explained under the discussion of the LTORG assembler instruction.

START -- Start Assembly

The START instruction may be used to give a name to the first (or only) control section of a source module. It may also be used to specify the initial value of the location counter for the first control section of the module. The typical form of the START instruction statement is as follows:

Name	Operation	Operand
Any symbol or not used	START	A self-defining term or not used

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

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

The assembler uses the self-defining term specified by the operand as the initial value of the location counter of the program. This value should be divisible by eight. For example, either of the following statements:

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

could be used to assign the name PROG2 to the first control section and to set the initial value of the location counter to 2040. If the operand is omitted, the assembler sets the initial value to zero. The location counter is set at the next doubleword boundary when the value of the START operand is not divisible by 8.

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

CSECT -- Identify Control Section

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

Name	Operation	Operand
Any symbol or not used	CSECT	Not used; should not be present

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

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

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

Under the Tape Operating System (TOS) a completely empty control section (CSECT) is flagged in error.

Unnamed Control Section

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

DSECT -- Identify Dummy Section

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

Name	Operation	Operand
An ordinary symbol or a variable symbol	DSECT	Not used; should not be present

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

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

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

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

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

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

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

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

The values assigned to symbols defined in a dummy section are relative to the initial statement of the section.

Thus, all machine-instructions which refer to names defined in the dummy section will, at execution time, refer to storage locations relative to the address loaded into the register.

An example is shown in the following coding. Assume that two independent modules (assembly 1 and assembly 2) have been loaded and are to be executed as a single overall program. Assembly 1 is an input routine that places a unit record in a specified area of storage, and places the address of that area in register 3. The input area is aligned on a fullword boundary. Then assembly 1 branches to assembly 2. Assembly 2 processes the record, which has the following format:

Columns	Content
1	INCODE
2	blank
3 and 4	INPUTA
5 through 8	INPUTB

The coding shown in the example is from assembly 2.

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

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

Name	Operation	Operand
ASMBLY2	CSECT	
BEGIN	BALR	2,0
	USING	*,2
	.	
	USING	INAREA,3
	CLI	INCODE,C'A'
	BE	ATYPE
	.	
ATYPE	MVC	WORKA,INPUTA
	MVC	WORKB,INPUTB
	.	
	CNOP	2,4
WORKA	DS	H
WORKB	DS	F
	.	
INAREA	DSECT	
	CNOP	0,4
INCODE	DS	CL1
	DS	CL1
INPUTA	DS	H
INPUTB	DS	F
	.	
	END	

The programmer must ensure that a section of code in his program is actually described by the dummy section which references it i.e., that data is properly aligned in both places. The DSECT named INAREA in the previous example adequately describes the section of code introduced into assembly 1, as it was aligned on a full word boundary. Further, WORKA and WORKB will be aligned and contiguous to each other in the same way as INPUTA and INPUTB are.

COM -- DEFINE BLANK COMMON CONTROL SECTION

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

Only one blank common control section may be designated in a source module. However, more than one COM statement may appear within a module. The first identifies the beginning of the blank common control section; the rest identify the resumption of the section.

When several modules are loaded, each designating a common control section, the amount of storage reserved is equal to the longest common control section. The form is:

Name	Operation	Operand
A sequence symbol or not used	COM	Not used; should not be present

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

It is necessary to establish addressability relative to a named statement within COM since the COM statement itself cannot have a name. In the following example, addressability to the common area of storage is established relative to the named statement XYZ.

Name	Operation	Operand
	.	
	L	8,=A(XYZ)
	USING	XYZ,8
	MVC	PDQ(16),=4C'ABCD'
	.	
	COM	
XYZ	DS	16F
PDQ	DS	16C
	.	

No instructions or constants appearing in a common control section are assembled. Data can only be placed in a common control section through execution of the program.

If the assignment of common storage is done in the same manner by each independent assembly, reference to a location in common by any assembly results in the same location being referenced. When the blank common control section is assembled the initial value of the location counter is set to zero.

SYMBOLIC LINKAGES

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

In the module where the linkage symbol is defined (i.e., used as a name), it must also be identified to the Linkage Editor and Assembler by means of the ENTRY assembler instruction (unless the symbol is the name of a CSECT or START statement). It is identified as a symbol that names an entry point, which means that another module may use that symbol in order to effect a branch operation or data reference. The assembler places this information in the external symbol dictionary.

Similarly, the module that uses a symbol defined in some other module must identify it by the EXTRN or WXTRN assembler instruction. Since the definition of the symbol appears in another module, the assembler arbitrarily assigns a length attribute of 1 and a value of 0. The assembler places this information in the external symbol dictionary.

Another way to obtain symbolic linkages is by using the V-type address constant. The subsection "Data Definition Instructions" in Section 5 contains the details pertinent to writing a V-type address constant. It is sufficient here to note that this constant may be considered an indirect linkage point. It is created from an externally defined symbol, but that symbol does not have to be identified by an EXTRN or WXTRN statement. The V-type address constant is intended to be used for external branch references (i.e., for effecting branches to other programs). Therefore, it should not be used for external data references (i.e., for referring to data in other modules).

ENTRY -- IDENTIFY ENTRY-POINT SYMBOL

The ENTRY instruction identifies linkage symbols that are defined in the module where the ENTRY instruction appears. These symbols can be referred to in other modules.

Name	Operation	Operand
A sequence symbol or not used	ENTRY	One or more relocatable symbols, separated by commas, that also appear as statement names

A source module may contain a maximum of 100 ENTRY symbols. ENTRY symbols which are not defined (not appearing as statement names), although invalid, will also count towards this maximum.

An ENTRY statement operand may not contain a symbol defined in a dummy section or in a blank common control section. An ENTRY statement containing a symbol defined in an unnamed control section can be processed by the assembler, but the DOS/TOS Linkage Editor will not process the resulting deck. The following example identifies the statements named SINE and COSINE as entry points to the program.

Name	Operation	Operand
	ENTRY	SINE,COSINE

Note: Labels of START and CSECT statements are automatically treated as entry-points to a module. Thus they need not be identified by ENTRY statements.

EXTRN -- IDENTIFY EXTERNAL SYMBOL

The EXTRN instruction identifies linkage symbols used by one source module but identified in another module. Each external symbol must be identified. This includes symbols

that refer to control section names. The format of the EXTRN statement is:

Name	Operation	Operand
A sequence symbol or not used	EXTRN	One or more relocatable symbols, separated by commas.

The symbols in the operand field may not appear as names of statements in the module. The following example identifies three external symbols that have been used as operands in the module but are identified in some other module.

Name	Operation	Operand
	EXTRN	RATEBL,PAYCALC
	EXTRN	WITHCALC

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

Note 1: A V-type address constant does not have to be identified by an EXTRN statement.

Note 2: Only one external symbol may be used in an expression.

Addressing External Control Sections

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

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

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

The combined number of control sections and dummy sections plus the number of unique symbols in EXTRN and WXTRN statements and V-type address constants may not exceed 255.
(EXTRN and WXTRN statements are discussed in this section; V-type constants in Section 5 under DC--Define Constant.) If the same symbol appears in a V-type address constant and in the name entry of a CSECT or DSECT statement, it is counted as two symbols.

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

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

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

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

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

WXTRN -- Identify Weak External Symbol
(DOS Assembler 14K D only)

The WXTRN statement has the same format and almost the same use as the EXTRN statement. The only difference is that WXTRN suppresses the AUTOLINK function of the linkage editor for the symbols identified by it. Its format is:

Name	Operation	Operand
A sequence symbol or not used	WXTRN	One or more relocatable symbols, separated by commas

The AUTOLINK (automatic library look-up) function searches the relocatable library for any unresolved external references. If it finds the external reference, it includes the module where the reference appears in the phase produced by the linkage editor.

For more detailed information on AUTOLINK refer to IBM System/360 Disk Operating System: System Control and System Service Programs (GC24-5036).

Note: AUTOLINK will be suppressed for a symbol defined both in a V-type address constant and in a WXTRN statement.

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

MACHINE-INSTRUCTION STATEMENTS

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

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

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

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

Instruction Alignment and Checking

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

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

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

OPERAND FIELDS AND SUBFIELDS

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

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

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

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

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

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

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

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

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

Assembler statement:

```
ST      4,FIELD
```

Assembled instruction:

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

An address may be specified explicitly as a base register and displacement (and index register for RX instructions) by the formats shown in the first column of Table 4-1. The address may be specified as an implied address by the formats shown in the second column. Observe that the two storage addresses required by the SS instructions are presented separately; an implied address may be used for one while an explicit address is used for the other.

Table 4-1. Details of Address Specification

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

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.

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

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

```
MVC   32(16,5),FIELD2
MVC   32(,5),FIELD2 (implied length)
```

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.

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

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

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

LENGTHS -- EXPLICIT AND IMPLIED

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

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

In either case, the length attribute for an expression is the length of the leftmost term in the expression. The length attribute of asterisk (*) is equal to the length of the instruction in which it appears, except that in an EQU to * statement, the length attribute is 1.

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

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

Note: If a length field of zero is desired, the length may be stated as zero or one.

To summarize, the length required in an SS instruction may be specified explicitly by the formats shown in the first column of Table 4-2 or may be implied by the formats shown in the second column. Observe that the two lengths required in one of the SS instruction formats are presented separately. An implied length may be used for one while an explicit length is used for the other.

Table 4-2. Details of Length Specification in SS Instructions

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

MACHINE-INSTRUCTION MNEMONIC CODES

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

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

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

Mnemonic codes for functions involving data usually indicate the data types, by letters that correspond to those for the data types in the DC assembler instruction (see Section 5). Furthermore, letters U and W have been added to indicate short and long, unnormalized floating-point operations, respectively. For example, AE indicates Add Normalized Short, whereas AU indicates Add Unnormalized Short. Where applicable, full-word fixed-point data is implied if the data type is omitted.

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

MACHINE-INSTRUCTION EXAMPLES

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

Implied addressing, control section addressing, and the function of the USING assembler instruction are not considered here. For discussion of these considerations and for examples of coding sequences that illustrate them, refer to Section 3, Program Sectioning and Linking, and Base Register Instructions.

RR Format

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

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

RX Format

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

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

RS Format

Name	Operation	Operand
ALPHA1	BXH	1,2,20(14)
ALPHA2	BXH	REG1,REG2,20(REGD)
ALPHA3	BXH	REG1,REG2,ZETA
ALPHA4	SLL	REG2,15
ALPHA5	SLL	REG2,0(15)

Whereas ALPHA1 and ALPHA2 specify explicit addresses, ALPHA3 specifies an implied address. ALPHA4 is a shift instruction shifting the contents of REG2 left 15 bit positions. ALPHA5 is a shift instruction shifting the contents of REG2 left by the value contained in general register 15.

SI Format

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

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

SS Format

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

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

EXTENDED MNEMONIC CODES

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

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
	Used After Compare Instructions		
BH	D2(X2,B2)	Branch on High	BC 2,D2(X2,B2)
BL	D2(X2,B2)	Branch on Low	BC 4,D2(X2,B2)
BE	D2(X2,B2)	Branch on Equal	BC 8,D2(X2,B2)
BNH	D2(X2,B2)	Branch on Not High	BC 13,D2(X2,B2)
BNL	D2(X2,B2)	Branch on Not Low	BC 11,D2(X2,B2)
BNE	D2(X2,B2)	Branch on Not Equal	BC 7,D2(X2,B2)
	Used After Arithmetic Instructions		
BO	D2(X2,B2)	Branch on Overflow	BC 1,D2(X2,B2)
BP	D2(X2,B2)	Branch on Plus	BC 2,D2(X2,B2)
BM	D2(X2,B2)	Branch on Minus	BC 4,D2(X2,B2)
BZ	D2(X2,B2)	Branch on Zero	BC 8,D2(X2,B2)
BNP	D2(X2,B2)	Branch on Not Plus	BC 13,D2(X2,B2)
BNM	D2(X2,B2)	Branch on Not Minus	BC 11,D2(X2,B2)
BNZ	D2(X2,B2)	Branch on Not Zero	BC 7,D2(X2,B2)
	Used After Test Under Mask Instructions		
BO	D2(X2,B2)	Branch if Ones	BC 1,D2(X2,B2)
BM	D2(X2,B2)	Branch if Mixed	BC 4,D2(X2,B2)
BZ	D2(X2,B2)	Branch if Zeros	BC 8,D2(X2,B2)
BNO	D2(X2,B2)	Branch if Not Ones	BC 14,D2(X2,B2)

Figure 4-1. Extended Mnemonic Codes

by the assembler into the corresponding operation and condition combinations.

The allowable extended mnemonic codes and their operand formats are shown in Figure 4-1, together with their machine-instruction equivalents. Unless otherwise noted, all extended mnemonics shown are for instructions in the RX format. Note that the only difference between the operand fields of the extended mnemonics and those of their machine-instruction equivalents is the absence of the R1 field and the comma that separates it from the rest of the operand field. The extended mnemonic list, like the machine-instruction list, shows explicit address formats only. Each address can also be specified as an implied address.

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

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

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 next instruction is an unconditional branch to the address contained in register 4. The last instruction is a "no operation". It will not branch under any condition because the mask field is zero.

SECTION 5: ASSEMBLER INSTRUCTION STATEMENTS

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

The following is a list of all the assembler instructions.

Symbol Definition Instruction
EQU - Equate Symbol

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

* Program Sectioning and Linking Instructions

START - Start Assembly
CSECT - Identify Control Section
DSECT - Identify Dummy Section
ENTRY - Identify Entry-Point Symbol
EXTRN - Identify External Symbol
WXTRN - Identify Weak External Symbol
COM - Identify Blank Common Control

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

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

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

* Discussed in Section 3.

SYMBOL DEFINITION INSTRUCTION

EQU -- EQUATE SYMBOL

The EQU instruction is used to define a symbol by assigning to it the length, value, and relocatability attributes of an expression in the operand field. The typical form of the EQU instruction statement is as follows:

Name	Operation	Operand
A variable symbol or an ordinary symbol	EQU	An expression

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

The symbol in the name field is given the same attributes as the expression in the operand field. The length attribute of the symbol is that of the leftmost (or only) term of the expression. When that term is * or a self-defining term, the length attribute is 1. The value attribute of the symbol is the value of the expression.

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

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

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

Name	Operation	Operand
FIELD	EQU	ALPHA-BETA+GAMMA

FIELD is defined as ALPHA-BETA+GAMMA and may be used in place of it. Note, however, that ALPHA, BETA, and GAMMA must all be previously defined. If the final result of the expression is negative, the low order 24 bits of the 2s complement is used.

DATA DEFINITION INSTRUCTIONS

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

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

DC -- DEFINE CONSTANT

The DC instruction is used to provide constant data in storage. It may specify one constant or a series of constants, thereby relieving the programmer of the necessity to write a separate data definition statement for each constant desired. Furthermore, 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 typical form of the DC instruction statement is as follows:

Name	Operation	Operand
Any symbol or not used	DC	One operand (D assembler) or one or more operands (F assembler) in the format described below, each separated by a comma.

Each operand consists of four subfields; the first three describe the constant, and the fourth subfield provides the constant or constants. The first and third subfields may be omitted, but the second and fourth must be specified. Note that more than one constant may be specified in the fourth subfield for most types of constants. Each constant so specified must be of the same type; the descriptive subfields that precede the constants apply to all of them. No blanks may occur within any of the subfields (unless provided as characters in a character constant or a character self-defining term), nor may they occur between the subfields of an operand. Similarly, blanks may not occur between operands and the commas that separate them when multiple operands are being specified.

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

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

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

The value attribute of the symbol naming the DC instruction is the address of the leftmost byte (after any necessary 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 in bytes (specified or implied) of the first constant.

Boundary alignment also varies according to the type of constant being specified and the presence of a length specification. Some constant types are only aligned to a byte boundary, but the DS instruction can be used to force any type of word boundary alignment for them. This is explained under "DS -- Define Storage." Other constants are aligned at various word boundaries (half, full, or double) in the absence of a length specification. 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 naming the constant will not receive a value attribute that is the location of a skipped byte.

Any bytes skipped in aligning statements that do not cause information to be assembled are not zeroed. Thus bytes skipped to align a DC statement are zeroed, and bytes skipped to align a DS statement are not zeroed.

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

LITERAL DEFINITIONS: The reader is reminded that the discussion of literals as machine-instruction operands (in Section 2) referred him to the description of the DC operand for the method of writing a literal operand. All subsequent operand specifications are applicable to writing literals, the only differences being:

1. The literal is preceded by an = sign.
2. Unsigned decimal values must be used to express the duplication factor and length modifier values.
3. The duplication factor may not be zero.
4. S-type address constants may not be specified.
5. Signed or unsigned decimal values must be used for exponent and scale modifier values.

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

Operand Subfield 1: Duplication Factor

The duplication factor may be omitted. If specified, it causes the constant(s) to be generated the number of times indicated by the factor. The factor may be specified either by an unsigned decimal self-defining term or by a positive absolute expression that is enclosed by parentheses. The duplication factor is applied after the constant is assembled. All symbols in the expression must be previously defined.

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

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

Operand Subfield 2: Type

The type subfield defines the type of constant being specified. From the type specification, the assembler determines how it is to interpret the constant and translate it into the appropriate machine format. The type is specified by a single-letter code as shown in Figure 5-1.

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

Operand Subfield 3: Modifiers

Modifiers describe the length in bytes desired for a constant (in contrast to an implied length), and the scaling and exponent for the constant. If multiple modifiers are written, they must appear in this sequence: length, scale, exponent. Each is written and used as described in the following text.

LENGTH MODIFIER: This is written as Ln, where n is either an unsigned decimal self-defining term or a positive absolute expression enclosed by parentheses. Any symbols in the expression must be previously defined. The value of n represents the number of bytes of storage that are assembled for the constant. The maximum value permitted for the length modifiers supplied for the various types of constants is summarized in Appendix F. This table also indicates the implied length for each type of constant; the implied length is used unless a length modifier is present. A length modifier may be specified for any type of constant. However, no boundary alignment will be provided when a length modifier is given.

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

Figure 5-1. Type Codes for Constants

Bit-Length Specification (F assembler only): The length of a constant, in bits, is specified by L.n, where n is specified as stated above and represents the number of bits in storage into which the constant is to be assembled. The value of n may exceed eight and is interpreted to mean an integral number of bytes plus so many bits. For example, L.20 is interpreted as a length of two bytes plus four bits.

Assembly of the first or only constant with bit-length specification starts on a byte boundary. The constant is placed in the high or low order end of the field depending on the type of constant being specified. The constant is padded or truncated to fit the field. If the assembled length does not leave the location counter set at a byte boundary, and another bit length constant does not immediately follow in the same statement, the remainder of the last byte used is filled with zeros. This leaves the location counter set at the next byte boundary. Figure 5-2 shows a fixed-point constant with a specified bit-length of 13, as coded, and as it would appear in storage. Note that the constant has been padded on the left to bring it to its designated 13-bit length.

As coded:

Name	Operation	Operand
BLCON	DC	FL.13'579'

In storage:

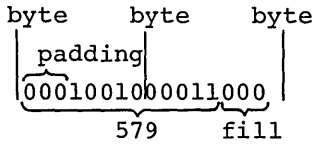


Figure 5-2. Bit-Length Specification (Single Constant)

The implied length of BLCON is two bytes. A reference to BLCON would cause the entire two bytes to be referenced.

When bit-length specification is used in association with multiple constants (see "Operand Subfield 4: Constant" following), each succeeding constant in the list is assembled starting at the next available bit. Figure 5-3 illustrates this.

As coded:

Name	Operation	Operand
BLMCON	DC	FL.10'161,21,57'

In storage:

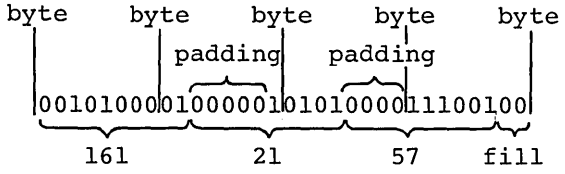


Figure 5-3. Bit-Length Specification (Multiple Constants)

The symbol used as a name entry in a DC assembler instruction takes on the length attribute of the first constant in the list; therefore the implied length of BLMCON in Figure 5-3 is two bytes.

If duplication is specified, filling occurs once at the end of the field occupied by the duplicated constant(s).

When bit-length specification is used in association with multiple operands, assembly of the constant(s) in each succeeding operand starts at the next available bit. Figure 5-4 illustrates this.

As coded:

Name	Operation	Operand
BLMCON	DC	FL.7'9',CL.10'AB',XL.14'C4'

In storage:

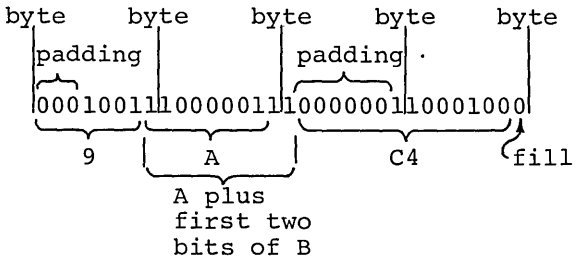


Figure 5-4. Bit-Length Specification (Multiple Operands)

In Figure 5-4, three different types of constants have been specified, one to an operand. Note that the character constant 'AB' which normally would occupy 16 bits is truncated on the right to fit the 10-bit field designated. Note that filling occurs only at the end of the field occupied by all the constants.

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 none is present, a plus sign is assumed. The maximum values for scale modifiers are summarized in Appendix F.

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

Scale Modifier for Fixed-Point Constants: the scale modifier specifies the power of two by which the constant must be multiplied after it has been converted to its binary representation. Just as multiplication of a decimal number by a power of 10 causes the decimal point to move, multiplication of a binary number by a power of two 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 being 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. 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 saved.

Scale Modifier for Floating-Point Constants: 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. Since 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 frac-

tion is shifted, the exponent is adjusted accordingly to retain the correct magnitude. When hexadecimal positions are lost, rounding occurs in the leftmost hexadecimal position of the lost portion. The rounding is reflected in the rightmost hexadecimal position saved.

EXPONENT MODIFIER: This modifier is written as En, where n is either a decimal self-defining term or an absolute expression enclosed by parentheses. Any symbols in the expression must be previously defined. The decimal value or the parenthesized expression may be preceded by a sign; if none is present, a plus sign is assumed. The maximum values for exponent modifiers are summarized in Appendix F.

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

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

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

Operand Subfield 4: Constant

This subfield supplies the constant (or constants) described by the subfields that precede it. A data constant (all types except A,Y,S,and V) is enclosed by apostrophes . An address constant (types A, Y, S, and V) 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., apostrophes 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)

* Not permitted for character, hexadecimal, and binary constants.

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

The total storage requirement of the 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.

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

E and H constants are converted as if they were D and F, respectively, and then shortened.

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

Character Constant -- C: Any of the valid 256 punch combinations may be designated in a character constant. Only one character constant may be specified per statement.

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

The maximum length of a character constant is 256 bytes. No boundary alignment is performed. Each character is translated into one byte. Double apostrophes or double ampersands count as one character. If

no length modifier is given, the size in bytes of the character constant is equal to the number of characters in the constant. If a length 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 110'      |
```

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

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

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

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

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

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

The generated constant would be:

ABCDABCDABCD

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

ABCDE ABCDE ABCDE

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

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

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

Constants that contain an even number of hexadecimal digits are translated as one byte per pair of digits. If an odd number of digits is specified, the leftmost byte has the leftmost four bits 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 1's.

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

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

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

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

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

Binary Constant -- B: A binary constant is written using 1's and 0's enclosed in apostrophes. Only one binary constant may be specified in a statement. Duplication and length may be specified. The maximum length of a binary constant is 256 bytes.

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

The following example shows the coding used to designate a binary constant. BCON would have a length attribute of one.

```
|Name    |Operation  |Operand                |
|--------|-----------|-----------------------|
|BCON    |DC         |B'11011101'            |
|BTRUNC  |DC         |BL1'100100011'         |
|BPAD    |DC         |BL1'101'               |
```

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

00100011

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

00000101

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

1. The number is written as a signed or unsigned decimal value. The decimal point may be placed before, within, or after the number, or it may be omitted, in which case the number is assumed to be an integer. A positive sign is assumed if an unsigned number is specified. Unless a scale modifier accompanies a mixed number or fraction, the fractional portion is lost, as explained under Subfield 3: Modifiers.
2. The exponent is optional. If specified, it is written immediately after the number as En, where n is an optionally signed decimal value specifying the exponent of the factor 10. The exponent may be in the range -85 to +75. If an unsigned exponent is specified, a plus sign is assumed. The exponent causes the value of the constant to be adjusted by the power of 10 that it specifies. The exponent may exceed the permissible range for exponents provided that the sum of the exponent and the exponent modifier do not exceed that range.

The number is converted to a binary number. The binary number is then rounded and assembled into the proper field, according to the specified or implied length. If the value of the number exceeds the length specified or implied, the sign is lost, the necessary leftmost bits are truncated to the length of the field and the value is then assembled into the whole field. Any duplication factor that is present is applied after the constant is assembled. A negative number is carried in 2's complement form. The resulting number will not differ from the exact value by more than one in the last place.

An implied length of four bytes is assumed for a full-word (F) and two bytes for a half-word (H), and the constant is aligned to the proper full-word or half-word boundary, if a length is not specified. However, any length up to and including eight bytes may be specified for either type of constant by a length modifier, in which case no boundary alignment occurs.

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

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

A field of three full-words is generated from the statement shown below. The location attribute of CONWRD is the address of the leftmost byte of the first word, and the length attribute is four, the implied length for a full-word fixed-point constant. The expression CONWRD+4 could be used to address the second constant (second word) in the field.

Name	Operation	Operand
CONWRD	DC	3F'658474'

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

Name	Operation	Operand
HALFCON	DC	HS6'-25.46'

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

Name	Operation	Operand
FULLCON	DC	HS12'3.50E-2'

The same constant could be specified as a literal:

Name	Operation	Operand
	AH	7,=HS12'3.50E-2'

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

Name	Operation	Operand
THREECON	DC	FS4'10,25.3,100'

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

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

2. The exponent is optional. If specified, it is written immediately after the number as En, where n is an optionally signed decimal value specifying the exponent of the factor 10. The exponent may exceed the permissible range for exponents, provided that the sum of the exponent and the exponent modifier does not exceed that range. If an unsigned exponent is specified, a plus sign is assumed.

Machine format for a floating-point number is in two parts: the portion containing the exponent, which is sometimes called the characteristic, followed by the portion containing the fraction, which is sometimes called the mantissa. 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 to the 2nd. Represented as a fraction, it would be .2735 times 10 to the 4th, the exponent having been modified to reflect the shifting of the decimal point. The exponent may also be affected by the presence of an exponent modifier, as explained under Operand Subfield 3: Modifiers.

The exponent is then translated into its binary equivalent, and the fraction is converted to a binary number. Scaling is performed if specified; if not, the fraction is normalized (leading hexadecimal zeros are removed). Rounding of the fraction is then performed according to the specified or implied length, and the number

is assembled into the proper field. 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. Negative fractions are carried in true representation, not in the 2's complement form. The resulting number will not differ from the exact value by more than one in the last place.

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

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

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

The following would each be generated as double-word floating-point constants.

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

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

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

If zoned decimal format is specified (Z), each decimal digit is translated into one byte. The translation is done according to the character set shown in Appendix A. The rightmost byte contains the sign as well as the rightmost digit. For packed decimal format (P), each pair of decimal digits is translated into one byte. The rightmost digit and the sign are translated into the rightmost byte. The bit configuration for the digits is identical to the configurations for the hexadecimal digits 0-9 as shown in Section 3 under "Hexadecimal Self-Defining Value." For both packed and zoned decimals, a plus sign is translated into the hexadecimal digit C, and a minus sign into the digit D.

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

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

1. If the constant requires fewer bytes than the length specifies, the necessary number of bytes is added to the left. For zoned decimal format, the decimal digit zero is placed in each added byte. For packed decimals, the bits of each added byte are set to zero.
2. If the constant requires more bytes than the length specifies, the necessary number of leftmost digits or pairs of digits is dropped, depending on which format is specified.

Examples of decimal constant definitions follow.

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

The following statement specifies three packed decimal constants. The length modifier applies to each packed decimal constant.

Name	Operation	Operand
DECIMALS	DC	PL8'+25.8,-3874,+2.3'

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

Name	Operation	Operand
	UNPK	OUTAREA,=PL2'+25'

ADDRESS CONSTANTS: An address constant is a storage address that is translated into a constant. Address constants can be used for initializing base registers to facilitate the addressing of storage. Furthermore, they provide the means of communicating between control sections of a multisection program. However, storage addressing and control section communication are also dependent on the use of the USING assembler instruction and the loading of registers. Coding examples that illustrate these considerations are provided in Section 3 under "Programming with the Using Instruction."

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

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

The value of the expression is determined when the referenced control sections are loaded. Complex relocatable expressions can be used to determine the distance between two control sections after they are loaded into main storage.

A-Type Address Constant: This constant is specified as an absolute, relocatable, or complex relocatable expression. (Remember that an expression may be single term or multiterm.) The value of the expression is calculated to 32 bits as explained in Section 2, with one exception: the maximum value of the expression may be $2^{31}-1$. The value is then truncated on the left, if necessary, to the specified or implied length of the field and assembled into the rightmost bits of the field. The implied length of an A-type constant is four bytes and alignment is to a full-word boundary unless a length is specified, in which case no alignment will occur. The length that may be specified depends on the type of expression used for the constant; a length of 1-4 bytes (.1 (1 bit) to 4 bytes for DOS F) may be used for an absolute expression, while a length of 3 or 4 bytes may be used for a relocatable or complex relocatable expression.

In the following examples, the field generated from the statement named ACON contains four constants, each of which occupies four bytes. Note that there is a Location Counter reference in one. The value of the Location Counter will be the address of the first byte allocated to the fourth constant. The second statement shows the same set of constants specified as literals (i.e., address constant literals).

Name	Operation	Operand
ACON	DC	A(108,LOP,END-STRT,*+4096)
	LM	4,7,=A(108,LOP,END-STRT,*+4096)

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

Y-type Address Constant: A Y-type address constant has much in common with the A-type constant. It, too, is specified as an absolute, relocatable, or complex relocata-

ble expression. The value of the expression is also calculated to 32 bits as explained in Section 2. However, the maximum value of the expression may be only $2^{15}-1$. The value is then truncated, if necessary, to the specified or implied length of the field and assembled into the rightmost bits of the field. The implied length of a Y-type constant is two bytes and alignment is to a half-word boundary unless a length is specified, in which case no alignment occurs. The maximum length of a Y-type address constant is two bytes. If length specification is used, a length of two bytes may be designated for a relocatable or complex expression and 1 or 2 bytes (.1 (1 bit) to 2 bytes for DOS F) for an absolute expression.

Warning: Specification of relocatable Y-type address constants should be avoided in programs destined to be executed on machines having more than 32,767 bytes of storage capacity.

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

The constant may be specified in two ways:

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

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

If length specification is used, only two bytes may be specified. S-type address constants may not be specified as literals.

V-Type Address Constant: This constant is used to reserve storage for the address of an external symbol that is used for effecting branches to other programs. The constant may not be used for external data references. The constant is specified as one relocatable symbol, which need not be identified by an EXTRN statement. Whatever symbol is used is assumed to be an external symbol by virtue of the fact that it is supplied in a V-type address constant. To suppress the AUTOLINK function of the linkage editor for a constant identified in a V-type address constant, the programmer can identify it in a WXTRN statement (DOS Assembler 14K D only).

Note that specifying a symbol as the operand of a V-type constant does not constitute a definition of the symbol for this assembly. The implied length of a V-type address constant is four bytes, and boundary alignment is to a full word. A length modifier may be used to specify a length of either three or four bytes, in which case no such boundary alignment occurs. In the following example, 12 bytes will be reserved, because there are three symbols. The value of each assembled constant will be zero until the program is loaded.

Name	Operation	Operand
VCONST	DC	V(SORT,MERGE,CALC)

DS -- DEFINE STORAGE

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

Name	Operation	Operand
Any symbol or not used	DS	One operand (D assembler) or one or more operands (F assembler) in the format described below, each separated by a comma.

The format of the DS operand is identical to that of the DC operand; exactly the same subfields are employed and are written in exactly the same sequence as they are in the DC operand. Although the formats are identical, there are two differences in the specification of subfields. They are:

1. The specification of data (subfield 4) is optional in a DS operand, but it is mandatory in a DC operand. If a constant is specified, it must be valid.
2. The maximum length that may be specified for character (C) and hexadecimal (X) field types is 65,535 bytes rather than 256 bytes.

If a DS operand specifies a constant in subfield 4, and no length is specified in subfield 3, the assembler determines the length of the data and reserves the appropriate amount of storage. It does not

assemble the constant. The ability to specify data and have the assembler calculate the storage area that would be required for such data is a convenience to the programmer. If he knows the general format of the data that will be placed in the storage area during program execution, all he needs to do is show it as the fourth subfield in a DS operand. The assembler then determines the correct amount of storage to be reserved, thus relieving the programmer of length considerations.

If the DS instruction is named by a symbol, its value attribute is the location of the leftmost byte of the reserved area. The length attribute of the symbol is determined in the same manner as for a DC. Any positioning required for aligning the storage area to the proper type of boundary is done before the address value is determined. Bytes skipped for alignment are not set to zero.

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

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

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

```
┌──────┬──────────┬──────────────────┐
|Name  |Operation |Operand           |
├──────┼──────────┼──────────────────┤
|FIELD |DS        |4CL10             |
|AREA  |DS        |CL100             |
└──────┴──────────┴──────────────────┘
```

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

Additional examples of DS statements are shown below:

```
┌─────┬────────┬──────────────────────────┐
|Name |Operation|Operand                  |
├─────┼────────┼──────────────────────────┤
|ONE  |DS      |CL80(one 80-byte field,   |
|     |        |  length attribute of 80  |
|TWO  |DS      |80C(80 one-byte fields,   |
|     |        |  length attribute of one|
|THREE|DS      |6F(six full words, length|
|     |        |  attribute of four)      |
|FOUR |DS      |D(one double word, length|
|     |        |  attribute of eight)     |
|FIVE |DS      |4H(four half-words,       |
|     |        |  length attribute of     |
|     |        |  two)                    |
└─────┴────────┴──────────────────────────┘
```

Note: A DS statement causes the storage area to be reserved but not set to zeros. No assumption should be made as to the contents of the reserved area.

Special Uses of the Duplication Factor

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

Name	Operation	Operand
	DS	0D
AREA	DS	CL128

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

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

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

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

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

CCW -- DEFINE CHANNEL COMMAND WORD

The CCW instruction provides a convenient way to define and generate an eight-byte Channel Command Word aligned at a doubleword boundary. The internal machine format of a Channel Command Word is shown in Table 5-1. CCW will cause any bytes skipped to be zeroed. The typical form of the CCW instruction statement is:

Name	Operation	Operand
Any symbol or not used	CCW	Four operands, separated by commas, specifying the contents of the channel command word in the format described in the following text

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

1. An absolute expression that specifies the command code. This expression's value is right-justified in byte 1.
2. An expression specifying the data address. The value of this expression is in bytes 2-4.
3. An absolute expression that specifies the flags for bits 32-36 and zeros for bits 37-39. The value of this expression is right-justified in byte 5. (Byte 6 is set to zero.)
4. An absolute expression that specifies the count. The value of this expression is right-justified in bytes 7-8.

The following is an example of a CCW statement:

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

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

32-35	36-39
0100	1000

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

Table 5-1. Channel Command Word

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

LISTING CONTROL INSTRUCTIONS

The listing control instructions are used to identify an assembly listing and assembly output cards, to provide blank lines in an assembly listing, and to designate how much detail is to be included in an assembly listing. In no case are instructions or constants generated in the object program. Listing control statements except PRINT are not printed, unless the statement is continued. Then the first card of the statement will be printed.

TITLE -- IDENTIFY ASSEMBLY OUTPUT

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

Name	Operation	Operand
A special symbol, a sequence symbol, a variable symbol, or not used	TITLE	One to 100 characters, enclosed in single apostrophes

The name entry may contain a special symbol which is one to four alphabetic or numeric characters in any combination. The contents of the name entry are then punched into columns 73-76 of all the output cards for the program except those produced by the PUNCH and REPRO assembler instructions. Only the first TITLE statement in a program may have a special symbol or variable symbol in the name entry. The name field of all subsequent TITLE statements must be blank or contain a sequence symbol.

The operand field may contain up to 100, characters enclosed in apostrophes. Any ampersands or apostrophes enclosed within the surrounding apostrophes must be represented by two ampersands or apostrophes.

The double ampersands and apostrophes punched into a TITLE card appear as single ampersands and apostrophes in a TITLE statement of an assembler listing. A single apostrophe between the enclosing apostrophes simply terminates the operand field. A single ampersand initiates an attempt to identify a variable symbol. If the variable symbol is not identifiable the statement is flagged as an error.

However, it is the number of printed characters that are counted in the total number of operand characters. The contents of the name and 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. Each TITLE statement encountered after the first one causes the listing to be advanced to a new page (before the heading is printed).

For example, if the following statement is the first TITLE statement to appear in a program:

Name	Operation	Operand
PGM1	TITLE	'FIRST HEADING'

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

If the following statement occurs later in the same program:

Name	Operation	Operand
	TITLE	'A NEW HEADING'

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

Note: The sequence number of the cards in the output deck is contained in columns 77-80, except those produced by the PUNCH and REPRO assembler instructions.

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 typical form of the EJECT instruction statement is as follows:

Name	Operation	Operand
A sequence symbol or not used	EJECT	Not used; should be blank

If the next line of the listing would appear at the top of a new page without the EJECT instruction, the EJECT instruction has no immediate effect. If one or more EJECT statements appear after the first EJECT, one or more pages are skipped. A TITLE instruction followed immediately by an EJECT instruction will result in a page with a title line and a statement heading line. Text following the EJECT instruction will begin at the top of the next page.

SPACE -- SPACE LISTING

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

Name	Operation	Operand
A sequence symbol or not used	SPACE	A decimal value or not used

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

PRINT -- PRINT OPTIONAL DATA

The PRINT instruction controls the content of the assembly listing. The typical form of the PRINT instruction is:

Name	Operation	Operand
A sequence symbol or not used	PRINT	One to three operands

One to three of the following operands are used:

ON - A listing is printed.
or
OFF - No listing is printed.

GEN - All statements generated by macro instructions are printed.
or
NOGEN - Statements generated by macro instructions are not printed, except MNOTE messages which print regardless of NOGEN. However, the outer macro instruction itself will appear in the listing.

DATA - Constants are printed out in full in the listing.
or
NODATA - Only the leftmost eight bytes (16 hexadecimal digits) are printed.

A program may contain any number of PRINT statements. The conditions set by a PRINT statement are in effect until another PRINT statement is encountered.

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

When OFF is specified, GEN and DATA have no effect. When NOGEN is specified, DATA has no effect for generated constants.

If no PRINT statement is encountered, the following default option is assumed:

Name	Operation	Operand
	PRINT	ON,NODATA,GEN

For example, if the statement:

Name	Operation	Operand
	DC	XL256'00'

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

Name	Operation	Operand
	PRINT	DATA

is the last PRINT statement to appear before the DC statement, all 256 bytes of zeros are printed in the assembly listing. However, if there are no previous PRINT statements, or:

Name	Operation	Operand
	PRINT	NODATA

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

PROGRAM CONTROL INSTRUCTIONS

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

ICTL -- INPUT FORMAT CONTROL

The ICTL instruction allows the programmer to alter the format of the statements in his source module. It can only be used to control statements that are read from the system input file (SYSIPT). It cannot be used to control the format of the input from the source statement library. Statements that are brought in from that library (through macro instructions or COPY instructions) are always assumed to be in the standard format.

The ICTL statement must precede all other statements, and may only be used once. Its format is:

Name	Operation	Operand
Not used, must not be present	ICTL	1-3 decimal values of the form b,e,c

Operand b specifies the begin column of the source statement. It must always be specified, and must be from 1-40, inclusive. Operand e specifies the end column of the source statement. The end column, when specified, must be from 41-80, inclusive; when not specified, it is assumed to be 71. The column after the end column is used to indicate whether the next card is a continuation card. Operand c specifies the continue column of the source statement. The continue column, when specified, must be from 2-40 and must be greater than b. If the continue column is not specified, or if column 80 is specified as the end column, the assembler assumes that there are no continuation cards, and all statements must be contained on a single card. The operand forms b,,c and b, are invalid.

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

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

Name	Operation	Operand
	ICTL	25

ISEQ -- INPUT SEQUENCE CHECKING

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

Name	Operation	Operand
Not used, must not be present	ISEQ	Two decimal values of the form 1,r, or not used

The operands 1 and r, respectively, specify the leftmost and rightmost columns of the field in the input cards to be checked. Operand r must be equal to or greater than operand 1. Columns to be checked must not be between the "begin" and "end" columns.

Sequence checking begins with the first card following the ISEQ statement. Comparison of adjacent cards makes use of the eight-bit internal collating sequence. Each card checked must be higher than the preceding one.

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

Sequence checking is only performed on statements contained in the source program. Statements inserted by the COPY assembler-instruction or generated by a macro-instruction are not checked for sequence.

PUNCH -- PUNCH A CARD

The PUNCH assembler-instruction causes the data in the operand to be punched into a card. One PUNCH statement produces one punched card. As many PUNCH statements may be used as are necessary. The typical form is:

Name	Operation	Operand
A sequence symbol or not used	PUNCH	1 to 80 characters enclosed in apostrophes

Using character representation, the operand is written as a string of up to 80 characters enclosed in apostrophes . All characters, including blank, are valid. The position immediately to the right of the left apostrophe is regarded as column one of the card to be punched. The assembly program does not process the data in the operand of a PUNCH statement other than causing it to be punched in a card. For each apostrophe or ampersand desired in the operand, two apostrophes or ampersands must be written. The two apostrophes or ampersands are reduced to a single apostrophe or ampersand. However, they count as only one character in the operand.

PUNCH statements may occur anywhere within a program, except before macro-definitions. They may occur within a macro-definition but not between a MEND statement and the beginning of the next macro. If a PUNCH statement occurs before the first control section, the resultant card will precede all other cards in the object program card deck; otherwise the card will be punched in place. No sequence number or identification is punched in the card.

REPRO -- REPRODUCE FOLLOWING CARD

The REPRO assembler-instruction causes data on the following statement line to be punched into a card. The data is not processed; it is punched in a card and no substitution is performed for variable symbols. No sequence number or identification is punched in the card. One REPRO instruction produces one punched card. The REPRO instruction may not appear before a macro-definition.

REPRO statements that occur before all statements composing the first or only control section will punch cards which precede all cards of the object deck. The form is:

Name	Operation	Operand
A sequence symbol or not used	REPRO	Not used, should not be present

The line to be reproduced may contain any combination of up to 80 characters. Characters may be entered starting in column 1 and continue through column 80 of the line. Column 1 of the line corresponds to column 1 of the card to be punched.

ORG -- SET LOCATION COUNTER

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

Name	Operation	Operand
A sequence symbol or not used	ORG	A relocatable expression or not used

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

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

An ORG statement must not be used to specify a location below the beginning of the control section in which it appears. The effect would be to give the location counter a large value. For example, the statement:

Name	Operation	Operand
	ORG	*-500

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

If it is desired to reset the Location Counter to the next available location in the current control section, the following statement would be used:

Name	Operation	Operand
	ORG	

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

LTORG -- BEGIN LITERAL POOL

The LTORG instruction causes all literals since the previous LTORG or beginning of the program to be assembled at appropriate boundaries starting at the first double-word boundary following the LTORG statement. If no literals follow the LTORG statement, alignment of the next instruction will occur. Bytes skipped are not zeroed. The typical form of the LTORG instruction statement is:

Name	Operation	Operand
Any symbol or not used	LTORG	Not used, should not be present

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

The literal pool is organized into four segments within which the literals are stored in order of appearance, dependent on the divisibility properties of their object lengths (dup factor times total explicit or implied length). The first segment contains all literals whose object length is a multiple of eight. Those remaining literals with lengths divisible by four are stored in the second segment. The third segment holds the remaining even-length literals. Any literals left over have odd lengths and are stored in the fourth segment.

Since each literal pool begins at a double-word boundary, this guarantees that all segment one literals are double-word, segment two full-word, and segment three half-word aligned, with no space wasted except, possibly, at the pool origin.

Literals from the following statement are in the pool, in the segments indicated by the circled numbers,

```
MVC  A (6),=3H'1'  (3)
 AD  2,=D'1'          (1)
 LM  3,5,=3F'1'    (2)
 IC  2,=XL1'1'        (4)
 AD  2,=D'2'       (1)
```

Special Addressing Consideration

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

CNOP -- CONDITIONAL NO OPERATION

The CNOP instruction allows the programmer to align an instruction at a specific word boundary. If any bytes must be skipped in order to align the instruction properly, the assembler insures an unbroken instruction flow by generating no-operation instructions. This facility is useful in

creating calling sequences consisting of a linkage to a subroutine followed by parameters such as channel command words (CCW).

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

The typical form of the CNOP instruction statement is as follows:

Name	Operation	Operand
A sequence symbol or not used	CNOP	Two absolute expressions of the form b,w

Any symbols used in the expressions in the operand field must have been previously defined.

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

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

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

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

Name	Operation	Operand
	CNOP	0,8
	BALR	2,14

has no effect. However, this sequence:

Name	Operation	Operand
	CNOP	6,8
	BALR	2,14

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

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

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

Note: If the location counter is on an odd-numbered byte-boundary when a CNOP instruction is encountered, normal alignment occurs before the CNOP is processed.

Double Word							
Word				Word			
Half word		Half word		Half word		Half word	
Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte
0,4		2,4		0,4		2,4	
0,8		2,8		4,8		6,8	

Figure 5-5. CNOP Alignment

COPY -- COPY PREDEFINED SOURCE CODING

The COPY instruction obtains source-language coding from a system library and includes it in the program currently being assembled. Under the DOS D Assembler, 10K Variant, the coding to be included is obtained from the system source statement library. Under the DOS D Assembler, 14K Variant, and the DOS F Assembler, the coding to be included is obtained from the private source statement library, if one is assigned, or from the system source library, in that order of precedence. Under the TOS D Assembler, 10K and 14K Variants, the coding to be included is obtained from the standard private library. The form of the COPY instruction statement is as follows:

Name	Operation	Operand
Not used, must not be present	COPY	One symbol

The operand is a symbol that identifies the section of coding to be copied. The symbol must not be the same as the mnemonic operation code of a macro definition in the source statement library.

The assembler inserts the requested coding immediately after the COPY statement is encountered. The requested coding may not contain another COPY statement.

If identical COPY statements are encountered, the coding they request is brought into the program each time.

COPYed text is always in the normal format and is not governed by ICTL usage. See Copy Statements in Section 7 for further information. The procedure for placing source language coding in the system library is described in the System Control and System Service Programs publication listed in the Preface.

END -- END ASSEMBLY

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

The typical form of the END instruction statement is as follows:

Name	Operation	Operand
A sequence symbol or not present	END	A relocatable expression or not present

The operand specifies the point to which control may be transferred when loading is complete. For example:

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

NOTE: If macro instructions from the Source Statement Library are included in an assembly, errors detected during macro editing will be flagged after the END statement. The error messages do not follow the macro instructions, because the source statements are not available to the assembler during macro editing. Errors detected while editing Programmer Macros will be flagged in-line.

PART 3 -- CONDITIONAL ASSEMBLY AND MACRO FACILITIES IN THE ASSEMBLER LANGUAGE

SECTION 6: INTRODUCTION TO THE CONDITIONAL ASSEMBLY AND MACRO FACILITIES

The DOS/TOS conditional assembly and macro facilities are part of the DOS/TOS assembler language.

Conditional assembly allows one to specify assembler language statements which may or may not be assembled, depending upon conditions evaluated at assembly time. Conditional assembly statements are used to define, set, change, and test values during the course of the assembly itself.

The conditional assembly instructions may be used to vary the sequence of statements generated for each occurrence of a macro instruction. Conditional assembly instructions may also be used outside macro definitions, i.e., among the assembler language statements in the program.

The macro facilities provide the programmer with a convenient way of generating desired sequences of machine or certain assembler instructions many times in one or more programs. This is accomplished by writing a macro definition.

This macro definition is written only once, and a single statement, a macro instruction statement, is written each time a programmer wants to generate the desired sequence of statements.

The macro facilities simplify the coding of programs, reduce the chance of programming errors, and ensure that standard sequences of statements are used to accomplish desired functions.

THE MACRO INSTRUCTION STATEMENT

A macro instruction statement (also called a macro instruction) is a source program statement used to provide information for generating machine and assembler instructions from a macro definition. The generated instructions are source statements which are then processed by the assembler program.

Three types of macro instructions may be written. Each type has a different form of operand. They are:

1. Positional (Sections 7 and 8).
2. Keyword (Section 10).
3. Mixed-mode (Section 10).

Positional macro instruction operands are written in a fixed order.

Keyword macro instruction operands can be written in any order.

Mixed-mode macro instruction operands are a combination of both positional and keyword operands. That is, certain operand entries (positional) must be written in a fixed order; other operand entries (keyword) can be specified in any order.

THE MACRO DEFINITION

Before a macro instruction can be assembled, a macro definition must be available to the assembler.

A macro definition is a set of statements that provide the assembler with:

1. The name entry, mnemonic operation code, and the form of the macro instruction operand, and
2. The sequence of statements the assembler uses when the macro instruction appears in the source program.

Every macro definition consists of a macro definition header statement, a macro instruction prototype statement, a sequence of model statements, COPY statements, MEXIT, MNOTE, or conditional assembly instructions, and a macro definition trailer statement.

The macro definition header and trailer statements denote the beginning and end, respectively, of a macro-definition.

The macro instruction prototype statement specifies the name entry, mnemonic operation code, and the type of the macro instruction operand.

The model statements contained in a macro definition may be used by the assembler to generate machine instructions and certain assembler instructions that replace each occurrence of the macro instruction.

The COPY statements may be used to copy model statements, MEXIT instructions, MNOTE instructions, and conditional assembly instructions from a source statement library into a macro definition.

The MEXIT instruction can be used to terminate processing of a macro definition.

The MNOTE instruction can be used to generate a message.

The conditional assembly instructions may be used to vary the sequence of statements generated for each occurrence of a macro instruction. Conditional assembly instructions may also be used outside macro definitions, i.e., among the assembler language statements in the program.

If a macro definition is in-line with an assembly, it is called a programmer macro.

SOURCE STATEMENT LIBRARIES

The same macro definition may be made available to more than one source program by placing the macro definition in the system source statement library. The macro definition then becomes a system macro. This system library is a collection of macro definitions that can be used by all the assembler language programs in an installation. Once a macro definition has been placed on the system source statement library it may be used by writing a corresponding macro instruction in a source program. Macro definitions must be in the system source statement library under the same name as the prototype. The procedure for placing macro definitions in the system source statement library is described in the System Control and System Service Programs publication listed in the Preface.

System macro definitions provided by IBM are described in the Supervisor and Input/Output Macros publication, also listed in the Preface.

A macro definition may be made available to a specific assembly by placing the macro definition in a private source statement library. If the private source statement library is assigned at the time of assembly, the macro definitions in the private source statement library may be used by writing a corresponding macro instruction in a source program. The macro definitions in the private source statement library must be under the same name as the prototype. The procedure for placing macro definitions in the private source statement library is described in the System Control and System Service Programs publication listed in the Preface.

Editing errors in user-supplied macro definitions are found at the time the macro is read from the source statement library, i.e., after the END card. To determine where these errors are, it is necessary to punch all such macros, including inner macros, and insert them then in the source program as programmer macros. To aid in debugging it is advisable to run all macros as programmer macros before incorporating them in a source statement library.

VARYING THE GENERATED STATEMENTS

Each time a macro instruction appears in the source program, it is replaced by the same sequence of assembler language statements. Conditional assembly instructions, however, may be used to vary the number and format of the generated statements.

VARIABLE SYMBOLS

A variable symbol is a type of symbol that is assigned various values by either the programmer or the assembler. Thus, variable symbols allow different values to be assigned to one symbol. When the assembler uses a macro definition to determine what statements are to replace a macro instruction, variable symbols in the model statements are replaced with the current values assigned to them.

A variable symbol is written as an ampersand followed by from one to seven letters and/or digits, the first of which must be a letter.

Types of Variable Symbols

There are three types of variable symbols: symbolic parameters, system variable symbols, and SET symbols. The SET symbols are further broken down into SETA symbols, SETB symbols, and SETC symbols. The three types of variable symbols differ in how they are assigned values.

Assigning Values to Variable Symbols

Symbolic parameters are assigned values by the programmer each time he writes a macro instruction.

System variable symbols are assigned values by the assembler each time it processes a macro instruction.

SET symbols are assigned values by the programmer by means of conditional assembly instructions.

Global SET Symbols

The values assigned to SET symbols in one macro definition may be used in other macro definitions. All SET symbols used for this purpose must be defined as global SET symbols. All other SET symbols must be defined by the programmer as local SET symbols. Local SET symbols and the other variable symbols (that is, symbolic parameters and system variable symbols) are local variable symbols. Global SET symbols are global variable symbols.

ORGANIZATION OF THIS PART OF THE PUBLICATION

Sections 7 and 8 describe the basic rules for preparing macro definitions and for writing macro instructions.

Section 9 describes the rules for writing conditional assembly instructions.

Section 10 describes additional features including rules for defining global SET symbols, preparing keyword and mixed-mode macro definitions, and writing keyword and mixed-mode macro instructions.

Appendix G contains a reference summary of the complete macro facilities.

Examples of the use of the features of the language appear throughout the remainder of the publication. These examples illustrate the use of particular features. However, they are not meant to show the full versatility of these features.

SECTION 7: HOW TO PREPARE MACRO DEFINITIONS

A macro definition consists of:

1. A macro definition header statement.
2. A macro instruction prototype statement.
3. Zero or more model statements, COPY statements, MEXIT, MNOTE, or conditional assembly instructions.
4. A macro definition trailer statement.

Except for MEXIT, MNOTE, and conditional assembly instructions, this section of the publication describes the statements that may be used to prepare macro definitions. Conditional assembly instructions are described in Section 9. MEXIT and MNOTE instructions are described in Section 10.

Macro-definitions in a source program must appear before all PUNCH and REPRO statements which appear in the main program. Specifically, only the listing control instructions (EJECT, PRINT, SPACE, and TITLE), ICTL and ISEQ instructions, and comments statements may occur before the macro definitions. All but the ICTL instruction may appear between macro definitions if there is more than one definition in the source program.

Note: A macro definition cannot appear within a macro definition.

MACRO -- MACRO DEFINITION HEADER

The macro definition header statement denotes the beginning of a macro definition. It must be the first statement in every macro definition. The form of this statement is:

Name	Operation	Operand
Not used, must no be present	MACRO	Not used, must not be present

MEND -- MACRO DEFINITION TRAILER

The macro definition trailer statement denotes the end of a macro definition. It must be the last statement in every macro definition. The form of this statement is:

Name	Operation	Operand
A sequence symbol or not used	MEND	Not used, must not be present

This statement also tells the assembler to terminate processing of a macro definition. Processing can be terminated at some other point in a macro definition through the MEXIT instruction.

MACRO INSTRUCTION PROTOTYPE

The macro instruction prototype statement (also called the prototype statement) specifies the name entry, mnemonic operation code, and the form of all macro instructions that refer to the macro definition. It must be the second statement of every macro definition. The typical form of this statement is:

Name	Operation	Operand
A symbolic parameter or not used	A symbol	Zero to 100 (200 for F assembler) symbolic parameters, separated by commas

The symbolic parameters are used in the macro definition to represent the name entry and operands of the corresponding macro instruction. A description of symbolic parameters appears following Model Statements.

The name entry of the prototype statement may be unused or it may contain a symbolic parameter.

The symbol in the operation entry is the mnemonic operation code that must appear in all macro instructions that refer to this macro definition. The mnemonic operation code must not be the same as the mnemonic operation code of another macro definition in the source program or of a machine instruction or assembler instruction.

The operand may contain zero to 100 (200 for F assembler) symbolic parameters separated by commas.

The following is a prototype statement.

Name	Operation	Operand
&NAME	MOVE	&TO,&FROM

Alternate Statement Form

The prototype statement may be written in a form different from that used for machine or assembler instructions. The normal form is described in Part 1 of this publication. The alternate form described here allows the programmer to write an operand on each line, and allows the interspersing of operands and comments in the statement.

In the alternate form, as in the normal form, the name and operation entries must appear on the first line of the statement, and at least one blank must follow the operation entry on that line. Both types of statement forms may be used in the same prototype statement.

The rules for using the alternate statement form are:

1. If an operand is followed by a comma and a blank, and the column after the end column contains a nonblank character, the operand entry may be continued on the next line starting in the continue column. More than one operand may appear on the same line.
2. Comments may appear after the blank that indicated the end of an operand, up to and including the end column.
3. If the next line starts after the continue column, the information entered on that line is considered to be comments, and the operand field is considered terminated. Any subsequent continuation lines are considered to contain only comments.

Note: A prototype statement may be written on as many continuation lines as is necessary to contain 100 (200 for F assembler) operands and associated comments.

The following examples illustrate: (1) the normal statement form, (2) the alternate statement form, and (3) the combination of both statement forms.

Name	Operation	Operand Comments	
NAME1	OP1	OPERAND1,OPERAND2,OPERAN D3 THE NORMAL FORM	X
NAME2	OP2	OPERAND1, THIS IS THE AL OPERAND2,OPERAND3, TERNA TE STATEMENT FORM	X X X
NAME3	OP3	OPERAND1, THIS IS A COMB OPERAND2,OPERAND3,OPERAN D4,OPERAND5 INATION OF BOTH STATEMENT FORMATS	X X X

MODEL STATEMENTS

Model statements are the macro definition statements from which the desired sequences of machine instructions and certain assembler instructions are generated. Zero or more model statements may follow the prototype statement. A model statement consists of one to four entries. They are, from left to right, the name, operation, operand, and comments entries.

The name entry may be unused, or it may contain an ordinary symbol, a sequence symbol or a variable symbol, depending on the particular statement. (Neither * nor .* may be substituted in the begin column of a model statement.)

The operation entry may contain any machine, assembler, or macro instruction mnemonic operation code, except COPY, END, ICTL, ISEQ, and PRINT; or it may contain a variable symbol. Variable symbols may not be used to generate the following mnemonic operation codes, nor may variable symbols be used in the name and operand entries of these instructions: COPY, END, ICTL, or ISEQ. Variable symbols may not be used to generate CSECT, DSECT, PRINT, REPRO, START, MACRO, MEND, MEXIT, LCLA, LCLB, LCLC, GBLA, GBLB, GBLC, SETA, SETB, SETC, AIF, AIFB, AGO, AGOB, ANOP, or macro instruction mnemonic operation codes. Variable symbols may not be used to generate the name and operation code of the ACTR instruction.

Variable symbols may also be used outside of macro definitions to generate mnemonic operation codes with the preceding restrictions.

Although COPY statements may not be used as model statements, they may be part of a macro definition. The use of COPY statements is described under COPY Statements.

The operand entry may contain ordinary symbols or variable symbols. After substitution, the operand must not be greater than 127 (255 for F assembler) characters. Model statement fields must follow the rules for paired apostrophes, ampersnads, and blanks, as macro-instruction operands. (See "Macro-Instruction Operands" in section 8.) Sequence symbols must appear in the operand entry of AGO and AIF instructions.

The comments entry may contain any combination of characters. Substitution by the use of variable symbols is not allowed.

If a REPRO statement is used as a model statement, it must be explicitly written in the operation entry. It may not be generated as a result of replacing a variable symbol by its value. Also, the line following it may not contain variable symbols. Substituted statements may not have blanks in any fields except between paired apostrophes. They may not have leading blanks in the name or operand fields.

SYMBOLIC PARAMETERS

A symbolic parameter is a type of variable symbol consisting of an ampersand followed by one to seven letters and/or numbers, the first of which must be a letter. Symbolic parameters appear in prototype and model statements. They are assigned values by the programmer when he writes a macro-instruction. The programmer may vary statements that are generated for each occurrence of a macro-instruction by varying the values assigned to symbolic parameters.

The programmer should not use &SYS as the first four characters of a symbolic parameter.

The following are valid symbolic parameters:

```
&READER     &LOOP2
&A23456     &N
&X4F2       &S4
```

The following are invalid symbolic parameters:

```
CARDAREA    (first character is not an
              ampersand)
&256B       (first character after
              ampersand is not a
              letter)
&AREA2456   (more than seven characters
              after the ampersand)
&BCD(34)    (contains a special charac-
              ter other than initial
              ampersand)
&IN AREA    (contains a special charac-
              ter, i.e., blank, other
              than initial ampersand)
```

The following is an example of a macro-definition. Note that the symbolic parameters in the model statements appear in the prototype statement.

	Name	Operation	Operand
Header		MACRO	
Prototype	&NAME	MOVE	&TO,&FROM
Model	&NAME	ST	2,SAVE
Model		L	2,&FROM
Model		ST	2,&TO
Model		L	2,SAVE
Trailer		MEND	

Symbolic parameters in model statements are replaced by the characters of the macro-instruction operand that correspond to the symbolic parameters.

In the following example the characters HERE, FIELDA, and FIELDB of the MOVE macro-instruction correspond to the symbolic parameters &NAME, &TO, and &FROM, respectively, of the MOVE prototype statement.

Name	Operation	Operand
HERE	MOVE	FIELDA,FIELDB

Any occurrence of the symbolic parameters &NAME, &TO, and &FROM in a model statement will be replaced by the characters HERE, FIELDA, and FIELDB, respectively. If the preceding macro-instruction was used in a source program, the following assembler language statements would be generated:

Name	Operation	Operand
HERE	ST	2,SAVE
	L	2,FIELDB
	ST	2,FIELDA
	L	2,SAVE

The example below illustrates another use of the MOVE macro-instruction using different operands than those that appear in the preceding example.

	Name	Operation	Operand
Macro	LABEL	MOVE	IN,OUT
Generated	LABEL	ST	2,SAVE
Generated		L	2,OUT
Generated		ST	2,IN
Generated		L	2,SAVE

If a symbolic parameter appears in the comments field of a model statement, it is not replaced by the corresponding characters of the macro-instruction.

Concatenating Symbolic Parameters with Other Characters or Other Symbolic Parameters

Concatenation is the process of linking or joining together in a sequence, with a specified order. To concatenate is to join together in a specified order.

If a symbolic parameter in a model statement is immediately preceded or followed by other characters or another symbolic parameter, the characters that correspond to the symbolic parameter are combined, in the order given, in the generated statement, with the other characters or the characters that correspond to the other symbolic parameter. This process is called concatenation.

The macro-definition, macro-instruction, and generated statements in the following example illustrate these rules.

	Name	Operation	Operand
Header		MACRO	
Prototype	&NAME	MOVE	&TY,&P,&TO,&FROM
Model	&NAME	ST&TY	2,SAVEAREA
Model		L&TY	2,&P&FROM
Model		ST&TY	2,&P&TO
Model		L&TY	2,SAVEAREA
Trailer		MEND	
Macro	HERE	MOVE	D,FIELD,A,B
Generated	HERE	STD	2,SAVEAREA
Generated		LD	2,FIELDB
Generated		STD	2,FIELDA
Generated		LD	2,SAVEAREA

The symbolic parameter &TY is used in each of the four model statements to vary the mnemonic operation code of each of the generated statements. The character D in the macro-instruction corresponds to symbolic parameter &TY. Since &TY is preceded by other characters (i.e., ST and L) in the model statements, the character that corresponds to &TY (i.e., D) is concatenated with the other characters to form the operation fields of the generated statements.

The symbolic parameters &P, &TO, and &FROM are used in two of the model statements to vary part of the operand fields of the corresponding generated statements. The characters FIELD, A, and B correspond to the symbolic parameters &P, &TO, and &FROM, respectively. Since &P is followed by &FROM in the second model statement, the characters that correspond to them (i.e., FIELD and B) are concatenated to form part of the operand field of the second generated statement. Similarly, FIELD and A are concatenated to form part of the operand field of the third generated statement.

If the programmer wishes to concatenate a symbolic parameter with a letter, digit, left parenthesis, or period following the symbolic parameter he must immediately follow the symbolic parameter with a period. A period is optional if the symbolic parameter is to be concatenated with another symbolic parameter, or a special character other than a left parenthesis or another period that follows it.

If a symbolic parameter is immediately followed by a period, then the symbolic parameter and the period are replaced by the characters that correspond to the symbolic parameter. A period that immediately follows a symbolic parameter does not appear in the generated statement.

The following macro-definition, macro-instruction, and generated statements illustrate these rules.

	Name	Operation	Operand
Header		MACRO	
Prototype	&NAME	MOVE	&P,&S,&R1,&R2
Model	&NAME	ST	&R1,&S.(&R2)
Model		L	&R1,&P.B
Model		ST	&R1,&P.A
Model		L	&R1,&S.(&R2)
Trailer		MEND	
Macro	HERE	MOVE	FIELD,SAVE,2,4
Generated	HERE	ST	2,SAVE(4)
Generated		L	2,FIELDB
Generated		ST	2,FIELDA
Generated		L	2,SAVE(4)

The symbolic parameter &P is used in the second and third model statements to vary part of the operand field of each of the corresponding generated statements. The characters FIELD of the macro instruction correspond to &P. Since &P is to be concatenated with a letter (i.e., B and A) in each of the statements, a period immediately follows &P in each of the model statements. The period does not appear in the generated statements.

Similarly, symbolic parameter &S is used in the first and fourth model statements to vary the operand fields of the corresponding generated statements. &S is followed by a period in each of the model statements, because it is to be concatenated with a left parenthesis. The period does not appear in the generated statements.

Comments Statements

A model statement may be a comments statement. A comments statement consists of an asterisk in the begin column, followed by comments. The comments statement is used by the assembler to generate an assembler language comments statement, just as other model statements are used by the assembler to generate assembler language statements.

The programmer may also write comments statements in a macro definition which are not to be generated. These statements must have a period in the begin column, immediately followed by an asterisk and the comments.

The first statement in the following example will be used by the assembler to generate a comments statement; the second statement will not.

```
|Name      |Operation |Operand                |
|-----------------------------------------------|
|*  THIS STATEMENT WILL BE GENERATED            |
|.* THIS ONE WILL NOT BE GENERATED              |
```

The use of variable symbols for substitution in comments statements is not allowed. The * or .* of a comment statement, therefore, cannot be created by substitution for a variable symbol.

COPY STATEMENTS

A COPY statement is not a model statement. COPY statements may be used to copy model statements and MEXIT, MNOTE, and conditional assembly instructions into a macro definition from a system library, just as they may be used outside macro definitions to copy source statements into an assembler language program. Under the DOS D Assembler, 10K Variant, the coding to be included is obtained from the system source statement library. Under the DOS D Assembler, 14K Variant, and the DOS F Assembler, the coding to be included is obtained from the private source statement library, if one is assigned, or from the system source library, in that order of precedence. Under the TOS D Assembler, 10K and 14K Variants, the coding to be included is obtained from the standard private library.

The form of this statement is:

Name	Operation	Operand
Not used, must not be present	COPY	A symbol

The symbol in the operand entry identifies the section of coding to be copied. The symbol must not be the same as the operation mnemonic of a macro definition in a source statement library. Any statement that may be used in a macro definition may be part of the copied coding, except MACRO, MEND, COPY, and prototype statements.

Statements COPYed into the program must obey the restrictions on ordering of statements. For example, COPY must be between global and local declarations in the macro definition or in the main program if the COPYed text contains global and local declarations.

The typical form of a macro instruction is:

Name	Operation	Operand
A symbol, sequence symbol, or not used	Mnemonic operation code	Zero to 100 (200 for F assembler) operands, separated by commas

The name entry of the macro instruction may contain a symbol. The symbol will not be defined in the generation process unless a symbolic parameter appears in the name entry of the prototype and the same parameter appears in the name entry of a generated model statement.

The operation entry contains the mnemonic operation code of the macro instruction. The mnemonic operation code must be the same as the mnemonic operation code of a macro definition in the source program or in a source statement library.

The macro definition with the same mnemonic operation code is used by the assembler to process the macro instruction. Under the DOS D Assembler, 10K Variant, the macro definition is obtained from an inline programmer macro definition or from the system source statement library, in that order of precedence. Under the DOS D Assembler, 14K Variant, and the DOS F Assembler, the macro definition is obtained from an inline programmer macro definition, a private source statement library, if one is assigned, or the system source statement library, in that order of precedence. Under the TOS D Assembler, 10K and 14K Variants, the macro definition is obtained from an inline programmer macro definition or from the standard private library, in that order of precedence.

The placement and order of the operands in the macro instruction may be determined by the placement and order of the symbolic parameters in the operand entry of the prototype statement.

MACRO INSTRUCTION OPERANDS

Any combination of up to 127 (255 for assembler F) characters may be used as a macro instruction operand provided that the following rules concerning apostrophes, parentheses, equal signs, ampersands, commas, and blanks are observed.

Paired Apostrophes: An operand may contain one or more sequences of characters, each of which is enclosed within single apostrophes. (The sequence of characters itself may contain an even number of apostrophes). The single apostrophes, which enclose the sequence of characters, are called paired apostrophes.

The first sequence of characters starts with the first apostrophe in the operand. Subsequent character sequences start with the first apostrophe after the apostrophe that ends the previous sequence of characters.

In the following example, there are two sequences of characters enclosed within single apostrophes. Therefore, there are two sets of paired apostrophes: the first and fourth apostrophes, and the fifth and sixth apostrophes.

```
'A''B'C'D'
```

An apostrophe (not within paired apostrophes), immediately followed by a letter, and immediately preceded by the letter L (when L is preceded by any special character other than an ampersand), is not considered in determining paired apostrophes. For instance, the apostrophe in the following example is not considered.

```
L'SYMBOL
```

'AL'SYMBOL' is an invalid operand.

Paired Parentheses: There must be an equal number of left and right parentheses. The nth left parenthesis must appear to the left of the nth right parenthesis.

Paired parentheses are a left parenthesis and a following right parenthesis without any other parentheses intervening. If there is more than one pair, each additional pair is determined by removing any pairs already recognized and reapplying the above rule for paired parentheses. For instance, in the following example the first and fourth, the second and third, and the fifth and sixth parentheses are each paired parentheses.

```
(A(B)C)D(E)
```

A parenthesis that appears between paired apostrophes is not considered in determining paired parentheses. For instance, in the following example the middle parenthesis is not considered.

```
(')')
```

Equal Signs: An equal sign can only occur as the first character in an operand or between paired apostrophes or paired parentheses. The following examples illustrate these rules.

```
=F'32'
'C=D'
E(F=G)
```

Ampersands: Except as noted under "Inner Macro Instructions," each sequence of consecutive ampersands must be an even number of ampersands. The following example illustrates this rule.

```
&&123&&&&
```

Commas: A comma indicates the end of an operand, unless it is placed between paired apostrophes or paired parentheses. The following example illustrates this rule.

```
(A,B)C',
```

Blanks: Except as noted under Statement Form, a blank indicates the end of the operand entry, unless it is placed between paired apostrophes . The following example illustrates this rule.

```
'A B C'
```

The following are valid macro instruction operands:

```
SYMBOL          A+2
123             (TO(8),FROM)
X'189A'         0(2,3)
*               =F'4096'
L'NAME          AB&&9
'TEN = 10'      'PARENTHESIS IS )'
'COMMA IS ,'    'APOSTROPHE IS'''
```

The following are invalid macro instruction operands:

```
W'NAME          (odd number of apostrophes )
5A)B            (number of left parentheses
                  does not equal number of
                  right parentheses)
(15 B)          (blank not placed between
                  paired apostrophes )
'ONE' IS '1'    (blank not placed between
                  paired apostrophes )
```

STATEMENT FORM

Macro instructions may be written using the same alternate form that can be used to write prototype statements. If this form is used, a blank does not always indicate the end of the operand entry. The alternate form is described in Section 7, under the subsection "Macro Instruction Prototype."

OMITTED OPERANDS

If an operand that appears in the prototype statement is omitted from the macro instruction, then the comma that would have separated it from the next operand must be present. If the last operand(s) is omitted from a macro instruction, then the comma(s) separating the last operand(s) from the next previous operand may be omitted.

The following example shows a macro instruction preceded by its corresponding prototype statement. The macro instruction operands that correspond to the third and sixth operands of the prototype statement are omitted in this example.

```
|Name  |Operation  |Operand                  |
|------|-----------|-------------------------|
|      |EXAMPLE    |&A,&B,&C,&D,&E,&F        |
|      |EXAMPLE    |17,*+4,,AREA,FIELD(6)    |
```

If the symbolic parameter that corresponds to an omitted operand is used in a model statement, a null character value (not a blank) replaces the symbolic parameter in the generated statement, i.e., in effect the symbolic parameter is removed.

For example, the first statement below is a model statement that contains the symbolic parameter &C. If the operand that corresponds to &C was omitted from the macro instruction, the second statement below would be generated from the model statement.

```
|Name  |Operation  |Operand                  |
|------|-----------|-------------------------|
|      |MVC        |THERE&C.25,THIS          |
|      |MVC        |THERE25,THIS             |
```

OPERAND SUBLISTS

An operand of a macro instruction may be a sublist.

Sublists provide the programmer with a convenient way to refer to: (1) a collection of macro instruction operands as a single operand, or (2) a single operand in a collection of operands.

A sublist consists of one or more operands (suboperands) separated by commas and enclosed in paired parentheses. The entire sublist, including the parentheses, is considered to be one macro instruction operand.

A suboperand is always treated as a character string. It is not possible to pass a suboperand containing a sublist to an inner macro instruction (a macro instruction used as a model statement in a macro definition). The inner macro would regard the operand as a character string during generation.

Omitted suboperands are handled in the same way as omitted operands. If () appears as an operand, however, it is treated as a character string, not as a sublist with all suboperands omitted.

If a macro instruction is written in the alternate statement format, each sublist operand may be written on a separate line; the macro instruction may be written on as many lines as there are operands, including sublist operands.

The limit of 127 characters (255 for assembler F) applies to an entire sublist including sub-operands, parentheses, and commas within these parentheses.

If &P1 is a symbolic parameter in a prototype statement, and the corresponding operand of a macro-instruction is a sublist, then &P1(n) may be used in a model statement to refer to the nth operand of the sublist, where n may be any arithmetic expression allowed in a SETA instruction. The SETA instruction is described in Section 9. If &P1 is a symbolic parameter, and the corresponding operand of a macro instruction is a sublist, then &P1 refers to the entire sublist (including parentheses).

If the sublist notation is used, but the operand is not a sublist, then &P1(1) refers to the operand and &P1(2) through &P1(100) (&P1(200) for assembler F) refer to null character value. If an operand has the form (), it is treated as a character string and not as a sublist.

For example, consider the following macro definition, macro instruction, and generated statements.

	Name	Operation	Operand
Header		MACRO	
Prototype		ADDNUM	&NUM,&REG,&AREA
Model		L	&REG,&NUM(1)
Model		A	&REG,&NUM(2)
Model		A	&REG,&NUM(3)
Model		ST	&REG,&AREA
Trailer		MEND	
Macro		ADDNUM	(A,B,C),6,SUM
Generated		L	6,A
Generated		A	6,B
Generated		A	6,C
Generated		ST	6,SUM

The operand of the macro instruction that corresponds to symbolic parameter &NUM is a sublist. One of the operands in the sublist is referred to in the operand entry of three of the model statements. For example, &NUM(1) refers to the first operand in the sublist corresponding to symbolic parameter &NUM. The first operand of the sublist is A. Therefore, A replaces &NUM(1) to form part of the generated statement.

Note: When referring to an operand in a sublist, the left parenthesis of the sublist notation must immediately follow the last character of the symbolic parameter, e.g., &NUM(1). A period should not be placed between the left parenthesis and the last character of the symbolic parameter.

A period may be used between these two characters only when the programmer wants to concatenate the left parenthesis with the characters that the symbolic parameter represents. The following example shows what would be generated if a period appeared between the left parenthesis and the last character of the symbolic parameter in the first model statement of the above example.

	Name	Operation	Operand
Prototype Model		ADDNUM	&NUM,&REG,&AREA
		L	&REG,&NUM.(1)
Macro		ADDNUM	(A,B,C),6,SUM
Generated		L	6,(A,B,C)(1)

The symbolic parameter &NUM is used in the operand entry of the model statement. The characters (A,B,C) of the macro instruction correspond to &NUM. Since &NUM is immediately followed by a period, &NUM and the period are replaced by (A,B,C). The period does not appear in the generated statement. The resulting generated statement is an invalid assembler language statement.

INNER MACRO INSTRUCTIONS

A macro instruction may be used as a model statement in a macro definition. Macro instructions used as model statements are called inner macro instructions.

A macro instruction that is not used as a model statement is referred to as an outer macro instruction.

Any symbolic parameters used in an inner macro-instruction are replaced by the corresponding operands of the outer macro-instruction.

The macro-definition corresponding to an inner macro-instruction is used to generate the statements that replace the inner macro-instruction.

The ADDNUM macro-instruction of the previous example is used as an inner macro-instruction in the following example.

The inner macro-instruction contains two symbolic parameters, &S and &T. The characters (X,Y,Z) and J of the macro-instruction correspond to &S and &T, respectively. Therefore, these characters replace the symbolic parameters in the operand entry of the inner macro-instruction.

The assembler then uses the macro-definition that corresponds to the inner macro-instruction to generate statements to replace the inner macro-instruction. The fourth through seventh generated statements have been generated for the inner macro-instruction.

	Name	Operation	Operand
Header		MACRO	
Prototype		COMP	&R1,&R2,&S,&T,&U
Model		SR	&R1,&R2
Model		C	&R1,&T
Model		BNE	&U
Inner		ADDNUM	&S,12,&T
Model	&U	A	&R1,&T
Trailer		MEND	
		MACRO	
		ADDNUM	&NUM,&REG,&AREA
		L	&REG,&NUM(1)
		A	&REG,&NUM(2)
		A	&REG,&NUM(3)
		ST	&REG,&AREA
		MEND	
Outer	K	COMP	10,11,(X,Y,Z),J,K
Generated		SR	10,11
Generated		C	10,J
Generated		BNE	K
Generated		L	12,X
Generated		A	12,Y
Generated		A	12,Z
Generated		ST	12,J
Generated	K	A	10,J

Note: An ampersand that is part of a symbolic parameter is not considered in determining whether a macro-instruction operand contains an even number of consecutive ampersands.

LEVELS OF MACRO-INSTRUCTIONS

A macro-definition that corresponds to an outer macro-instruction may contain any number of inner macro-instructions. The outer macro-instruction is called a first level macro-instruction. Each of the inner macro-instructions is called a second level macro-instruction.

The macro-definition that corresponds to a second level macro-instruction may contain any number of inner macro-instructions. These macro-instructions are called third level macro-instructions, etc.

The number of levels of macro-instructions that may be used depends upon the complexity of the macro-definition and the amount of storage available. This is described in detail in Appendix H.

SECTION 9: HOW TO WRITE CONDITIONAL ASSEMBLY INSTRUCTIONS

The conditional assembly instructions allow the programmer to: (1) define and assign values to SET symbols that can be used to vary parts of generated statements, and (2) vary the sequence of generated statements. Thus, the programmer can use these instructions to generate many different sequences of statements from the same macro-definition.

There are 13 conditional assembly instructions, 10 of which are described in this section. The other three conditional assembly instructions -- GBLA, GBLB, and GBLC -- are described in Section 10. The instructions described in this section are:

LCLA	SETA	AIF	ANOP
LCLB	SETB	AGO	
LCLC	SETC	ACTR	

The primary use of the conditional assembly instructions is in macro-definitions. However, all of them may be used in an assembler language source program.

Where the use of an instruction outside macro-definitions differs from its use within macro-definitions, the difference is described in the subsequent text.

The LCLA, LCLB, and LCLC instructions are used to define and assign initial values to local SET symbols.

The SETA, SETB, and SETC instructions may be used to assign arithmetic, binary, and character values, respectively, to SET symbols. The SETB instruction is described after the SETA and SETC instructions, because the operand of the SETB instruction is a combination of the operands of the SETA and SETC instructions.

The AIF, AGO, and ANOP instructions may be used in conjunction with sequence symbols to vary the sequence in which statements are assembled. The programmer can test attributes assigned by the assembler to symbols or macro-instruction operands to determine which statements are to be processed. The ACTR instruction may be used to limit the number of AIF and AGO branches executed in any assembly.

Examples illustrating the use of conditional assembly instructions are included throughout this section. A chart summarizing the elements that can be used in each instruction appears at the end of this section.

SET SYMBOLS

SET symbols are one type of variable symbol. The symbolic parameters discussed in Section 7 are another type of variable symbol. SET symbols differ from symbolic parameters in three ways: (1) where they can be used in an assembler language source program, (2) how they are assigned values, and (3) how the values assigned to them can be changed.

Symbolic parameters can only be used in macro-definitions, whereas SET symbols can be used inside and outside macro-definitions.

SET symbols are assigned values by SETA, SETB, and SETC conditional assembly instructions and by local or global declarations.

Each symbolic parameter is assigned a single value for one use of a macro-definition, whereas the values assigned to each SETA, SETB, and SETC symbol are not so restricted.

Defining SET Symbols

SET symbols must be defined by the programmer before they are used. When a SET symbol is defined it is assigned an initial value. SET symbols may be assigned new values by means of the SETA, SETB, and SETC instructions. A SET symbol is defined when it appears as an operand of an LCLA, LCLB, or LCLC instruction.

Using Variable Symbols

The SETA, SETB, and SETC instructions may be used to change the values assigned to SETA, SETB, and SETC symbols, respectively. When a SET symbol appears in the name or operand entry of a statement, the current value of the SET symbol (i.e., the last value assigned to it) replaces the SET symbol in the statement. When a SETC symbol appears in the operation entry of a statement, the current value of the SETC symbol replaces the SET symbol in the statement.

For example, if &A is a symbolic parameter, and the corresponding characters of the macro-instruction are the symbol HERE, then HERE replaces each occurrence of &A in the macro-definition. However, if &A is a SET symbol, the value assigned to &A can be changed, and a different value can replace various occurrences of &A in the macro-definition.

The same variable symbol may not be used as a symbolic parameter and as a SET symbol in the same macro-definition.

The following illustrates this rule.

```
|Name   |Operation  |Operand              |
|-------|-----------|---------------------|
|&NAME  |MOVE       |&TO,&FROM            |
```

If the statement above is a prototype statement, then &NAME, &TO, and &FROM may not be used as SET symbols in the macro-definition.

The same variable symbol may not be used as two different types of SET symbols in the same macro-definition. Similarly, the same variable symbol may not be used as two different types of SET symbols outside macro-definitions.

For example, if &A is a SETA symbol in a macro-definition, it cannot be used as a SETC symbol in that definition. Similarly, if &A is a SETA symbol outside macro-definitions, it cannot be used as a SETC symbol outside macro-definitions.

The same variable symbol if declared local may be used in two or more macro-definitions and outside macro-definitions. If such is the case, the variable symbol will be considered a different variable symbol each time it is used.

For example, if &A is a variable symbol (either SET symbol or symbolic parameter) in one macro-definition, it can be used as a variable symbol (either SET symbol or symbolic parameter) in another definition. Similarly, if &A is a variable symbol (SET symbol or symbolic parameter) in a macro-definition, it can be used as a SET symbol outside macro-definitions.

All variable symbols may be concatenated with other characters in the same way as symbolic parameters. The rules for concatenation are in Section 7 under the subsection Model Statements.

Variable symbols in macro-instructions are replaced by the values assigned to them, immediately prior to the start of processing the definition. If a SET symbol is used in the operand entry of a macro-instruction, and the value assigned to the SET symbol is in the form of sublist notation, the operand is not considered a sublist.

ATTRIBUTES

The assembler assigns attributes to macro-instruction operands and to symbols in the program. These attributes may be referred to only in conditional assembly instructions.

There are six kinds of attributes. They are: type, length, scaling, integer, count, and number.

If an outer macro-instruction operand is a symbol before substitution, then the attributes of the operand are the same as the corresponding attributes of the symbol. The symbol must appear in the name entry of an assembler language statement or in the operand entry of an EXTRN statement in the program. The statement must be outside macro-definitions and must not contain any variable symbols.

If an inner macro-instruction operand i a symbolic parameter, then attributes of the operand are the same as the attributes of the corresponding outer macro-instruction operand.

Each attribute has a notation associated with it. The notations are:

Attribute	Notation
Type	T'
Length	L'
Scaling	S'
Integer	I'
Count	K'
Number	N'

If a macro-instruction operand is a sublist, the programmer may refer to the attributes of either the sublist or each operand in the sublist. The type, length, scaling, and integer attributes of a sublist are the same as the corresponding attributes of the first operand in the sublist.

All the attributes of macro-instruction operands may be referred to in conditional assembly instructions within macro-definitions. However, only the type, length, scaling, and integer attributes of symbols may be referred to in conditional assembly instructions outside macro- definitions. Symbols appearing in the name entry of generated statements are not assigned attributes.

The programmer may refer to an attribute in the following ways:

1. In a statement that is outside macro-definitions, he may write the notation for the attribute immediately followed by a symbol. (E.g., T'NAME refers to the type attribute of the symbol NAME.)

2. In a statement that is in a macro-definition, he may write the notation for the attribute immediately followed by a symbolic parameter. (E.g., L'&NAME refers to the length attribute of the characters in the macro-instruction that correspond to symbolic parameter &NAME; L'&NAME(2) refers to the length attribute of the second operand in the sublist that corresponds to symbolic parameter &NAME.)

Type Attribute (T')

The type attribute of a macro-instruction operand or a symbol is a letter.

The programmer may refer to a type attribute in the operand of a SETC instruction, or in character relations in the operands of SETB or AIF instruction, or in other instructions where use of the character is valid.

The following letters are used for symbols that name DC and DS statements and for outer macro-instruction operands that are symbols that name DC or DS statements.

A	A-type address constant, implied length, aligned.
B	Binary constant.
C	Character constant.
D	Long floating-point constant, implied length, aligned.
E	Short floating-point constant, implied length, aligned.
F	Full-word fixed-point constant, implied length, aligned.
G	Fixed-point constant, explicit length.
H	Half-word fixed-point constant, implied length, aligned.
K	Floating-point constant, explicit length.
P	Packed decimal constant.
R	A-, S-, V-, or Y-type address constant, explicit length.
S	S-type address constant, implied length, aligned.
V	V-type address constant, implied length, aligned.
X	Hexadecimal constant.
Y	Y-type address constant, implied length, aligned.
Z	Zoned decimal constant.

The following letters are used for symbols (and outer macro-instruction operands that are symbols) that name statements other than DC or DS statements, or that appear in the operand field of an EXTRN or WXTRN statement.

I	Machine instruction
J	Control Section Name
M	Macro instruction
T	EXTRN symbol
W	CCW instruction
$	WXTRN symbol

The following letters are used for inner and outer macro-instruction operands only.

N	Self-defining term
O	Omitted operand

The letter U (Undefined) is used for inner and outer macro-instruction operands that cannot be assigned any of the above letters. The type attribute of all literals appearing as macro-instruction operands is U. This also is true for inner macro-instruction operands that are ordinary symbols or variable symbols. Because the attributes are not available at the necessary time, this letter is also assigned to symbols that name EQU and LTORG statements, to any symbols occurring more than once in the name entry of source statements, and to all symbols naming DC and DS statements with expressions or variable symbols as modifiers. The type attribute also is undefined when the modifier expression consists solely of self-defining terms.

The attributes of A, B, C, and D in the following examples are undefined:

```
A  DC  3FL(A-B)'15'
B  DC  (A-B)F'15'
C  DC  &X'1'
D  DC  FL(3-2)'1'
```

Length (L'), Scaling (S'), and Integer (I') Attributes

The length, scaling, and integer attributes of macro-instruction operands and symbols are numeric values.

The length attribute of a symbol (or of a macro-instruction operand that is a symbol) is as described in Part I of this publication. Reference to the length attribute of a variable symbol is illegal except for symbolic parameters in SETA, SETB, and AIF statements. If the basic L' attribute is desired, it can be obtained as follows:

```
&A  SETC  'Z'
```

```
      &B  SETC  'L'''
          MVC   &A.(&B&A),X
```

After generation, this would result in

```
          MVC   Z(L'Z),X
```

Reference must not be made to the length attributes of symbols or macro-instruction operands whose type attributes are the letters M, N, O, T, $, or U.

Scaling and integer attributes are provided for symbols that name fixed-point, floating-point, and decimal DC or DS statements.

Fixed and Floating Point: The scaling attribute of a fixed point or floating point number is the value given by the scale modifier. The integer attribute is a function of the scale and length attributes of the number.

Decimal: The scaling attribute of a decimal number is the number of decimal digits to the right of the decimal point. The integer attribute of a decimal number is the number of decimal digits to the left of the decimal point.

Scaling and integer attributes are available for symbols and macro-instruction operands only if their type attributes are H, F, and G (fixed point); D, E, and K (floating point); or P and Z (decimal).

The programmer may refer to the length, scaling, and integer attributes in the operand field of a SETA instruction, or in arithmetic relations in the operand fields of SETB or AIF instructions.

Count Attribute (K')

The programmer may refer to the count attribute of macro-instruction operands only.

The count attribute is a value equal to the number of characters in the macro-instruction operand after substituting for variable symbols, excluding commas. If the operand is a sublist, the count attribute includes the beginning and ending parentheses and the commas within the sublist. The count attribute of an omitted operand is zero.

If a macro-instruction operand contains variable symbols, the characters that replace the variable symbols, rather than the variable symbols, are used to determine the count attribute.

The programmer may refer to the count attribute in the operand field of a SETA instruction, or in arithmetic relations in the operand fields of SETB and AIF instructions that are part of a macro-definition.

Number Attribute (N')

The programmer may refer to the number attribute of macro-instruction operands only.

The number attribute is a value equal to the number of operands in an operand sublist. The number of operands in an operand sublist is equal to one plus the number of commas that indicate the end of an operand in the sublist.

The following examples illustrate this rule.

```
(A,B,C,D,E)    5 operands
(A,,C,D,E)     5 operands
(A,B,C,D)      4 operands
(,B,C,D,E)     5 operands
(A,B,C,D,)     5 operands
(A,B,C,D,,)    6 operands
```

If the macro-instruction operand is not a sublist, the number attribute is one. If the macro-instruction operand is omitted, the number attribute is zero.

The programmer may refer to the number attribute in the operand field of a SETA instruction, or in arithmetic relations in the operand fields of SETB and AIF instructions that are part of a macro-definition.

Assigning Integer Attributes to Symbols

The integer attribute is computed from the length and scaling attributes.

Fixed Point: The integer attribute of a fixed-point number is equal to eight times the length attribute of the number minus the scaling attribute minus one; i.e., I'=8*L'-S'-1.

Each of the following statements defines a fixed-point field. The length attribute of HALFCON is 2, the scaling attribute is 6, and the integer attribute is 9. The length attribute of ONECON is 4, the scaling attribute is 8, and the integer attribute is 23.

Name	Operation	Operand
HALFCON	DC	HS6'-25.93'
ONECON	DC	FS8'100.3E-2'

Floating Point: The integer attribute of a floating-point number is equal to two times the difference between the length attribute of the number and one, minus the scaling attribute; i.e., I'=2*(L'-1)-S'.

Each of the following statements defines a floating-point value. The length attribute of SHORT is 4, the scaling attribute is 2, and the integer attribute is 4. The length attribute of LONG is 8, the scaling attribute is 5, and the integer attribute is 9.

Name	Operation	Operand
SHORT	DC	ES2'46.415'
LONG	DC	DS5'-3.729'

Decimal: The integer attribute of a packed decimal number is equal to two times the length attribute of the number minus the scaling attribute minus one; i.e., I'=2*L'-S'-1. The integer attribute of a zoned decimal number is equal to the difference between the length attribute and the scaling attribute; i.e., I'=L'-S'.

Each of the following statements defines a decimal field. The length attribute of FIRST is 2, the scaling attribute is 2, and the integer attribute is 1. The length attribute of SECOND is 3, the scaling attribute is 0, and the integer attribute is 3. The length attribute of THIRD is 4, the scaling attribute is 2, and the integer attribute is 2. The length attribute of FOURTH is 3, the scaling attribute is 2, and the integer attribute is 3.

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

SEQUENCE SYMBOLS

The name entry of a statement may contain a sequence symbol. Sequence symbols provide the programmer with the ability to vary the sequence in which statements are processed by the assembler.

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

A sequence symbol may be used in the name entry of any statement that does not contain a symbol or SET symbol, except a prototype statement, or a MACRO, LCLA, LCLB, LCLC, GBLA, GBLB, GBLC, ACTR, ICTL, ISEQ, or COPY instruction.

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

The following are valid sequence symbols:

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

The following are invalid sequence symbols:

```
CARDAREA   (first character is not
             a period)
.246B      (first character after
             period is not a letter)
.AREA2456  (more than seven characters
             after period)
.BCD%84    (contains a special character
             other than initial period)
.IN AREA   (contains a special
             character, i.e., blank,
             other than initial period)
```

If a sequence symbol appears in the name entry of a macro-instruction, and the corresponding prototype statement contains a symbolic parameter in the name entry, the sequence symbol does not replace the symbolic parameter wherever it is used in the macro-definition.

The following example illustrates this rule.

	Name	Operation	Operand
		MACRO	
1	&NAME	MOVE	&TO,&FROM
2	&NAME	ST	2,SAVEAREA
		L	2,&FROM
		ST	2,&TO
		L	2,SAVEAREA
		MEND	
3	.SYM	MOVE	FIELDA,FIELDB
4		ST	2,SAVEAREA
		L	2,FIELDB
		ST	2,FIELDA
		L	2,SAVEAREA

The symbolic parameter &NAME is used in the name entry of the prototype statement (statement 1) and the first model statement (statement 2). In the macro instruction (statement 3) a sequence symbol (.SYM) corresponds to the symbolic parameter &NAME. &NAME is not replaced by .SYM, and, therefore, the generated statement (statement 4) does not contain a name entry.

LCLA,LCLB,LCLC -- DEFINE SET SYMBOLS

The typical form of these instructions is:

Name	Operation	Operand
Not used, must not be present	LCLA, LCLB, or LCLC	One or more variable symbols, that are to be used as SET symbols, separated by commas

The LCLA, LCLB, and LCLC instructions are used to define and assign initial values to SETA, SETB, and SETC symbols, respectively. The SETA, SETB, and SETC symbols are assigned the initial values of 0, 0, and null character value, respectively.

The programmer should not define any SET symbol whose first four characters are &SYS.

All LCLA, LCLB, or LCLC instructions in a macro definition must appear immediately after the prototype statement and all GBLA, GBLB or GBLC instructions. All LCLA, LCLB, or LCLC instructions outside macro definitions must appear after all macro definitions in the source program, after all GBLA, GBLB, and GBLC instructions outside macro definitions, before all conditional assembly instructions, and PUNCH and REPRO statements outside macro definitions, and before the first control section of the program.

SETA -- SET ARITHMETIC

The SETA instruction may be used to assign an arithmetic value to a SETA symbol. The form of this instruction is:

Name	Operation	Operand
A SETA symbol	SETA	A SETA arithmetic expression

The expression in the operand entry is evaluated as a signed 32-bit arithmetic value which is assigned to the SETA symbol in the name entry. The minimum and maximum allowable values of the expression are -2^{31} and $+2^{31}-1$, respectively.

The expression may consist of one term or an arithmetic combination of terms. The terms that may be used alone or in combination with each other are self-defining terms, variable symbols, and the length, scaling, integer, count, and number attributes. Self-defining terms are described in Part 1 of this publication.

Note: A SETC variable symbol may appear in a SETA expression only if the value of the SETC variable is one to eight decimal digits. The decimal digits will be converted to a positive arithmetic value.

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

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

The following are valid operand fields of SETA instructions:

```
&AREA+X'2D'    I'&N/25
&BETA*10       &EXIT-S'&ENTRY+1
L'&HERE+32     29
```

The following are invalid operand fields of SETA instructions:

&AREAX'C'	(two terms in succession)
&FIELD+-	(two operators in succession)
-&DELTA*2	(begins with an operator)
*+32	(begins with an operator; two operators in succession)
NAME/15	(NAME is not a valid term)

EVALUATION OF ARITHMETIC EXPRESSIONS

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

The following evaluation procedure is used:

1. Each term is given its numerical value.
2. The arithmetic operations are performed moving from left to right. However, multiplication and/or division are performed before addition and subtraction.
3. The computed result is the value assigned to the SETA symbol in the name entry.

The arithmetic expression in the operand entry of a SETA instruction may contain one or more sequences of arithmetically combined terms that are enclosed in parentheses. A sequence of parenthesized terms may appear within another parenthesized sequence.

The following are examples of SETA instruction operands that contain parenthesized sequences of terms.

```
(L'&HERE+32)*29
&AREA+X'2D'/(&EXIT-S'&ENTRY+1)
&BETA*10*(I'&N/25/(&EXIT-S'&ENTRY+1))
```

The parenthesized portion or portions of an arithmetic expression are evaluated before the rest of the terms in the expression are evaluated. If a sequence of parenthesized terms appears within another parenthesized sequence, the innermost sequence is evaluated first.

The SETA arithmetic expression can only have five levels of parentheses. The parentheses required in subscripting, substring, and sublist notation count when determining these levels. A counter is maintained for each SETA statement and increased by one for each occurrence of a variable symbol as well as the operation entry. The maximum value this counter may attain is 35. (See Appendix H).

Using SETA Symbols

The arithmetic value assigned to a SETA symbol is substituted for the SETA symbol when it is used in an arithmetic relation. If the SETA symbol is not used in an arithmetic expression, the arithmetic value is completely converted to an unsigned integer, with leading zeros removed. If the value is zero, it is converted to a single zero.

The following example illustrates this rule:

	Name	Operation	Operand
		MACRO	
	&NAME	MOVE	&TO,&FROM
		LCLA	&A,&B,&C,&D
1	&A	SETA	10
2	&B	SETA	12
3	&C	SETA	&A-&B
4	&D	SETA	&A+&C
	&NAME	ST	2,SAVEAREA
5		L	2,&FROM&C
6		ST	2,&TO&D
		L	2,SAVEAREA
		MEND	
	HERE	MOVE	FIELDA,FIELDB
	HERE	ST	2,SAVEAREA
		L	2,FIELDB2
		ST	2,FIELDA8
		L	2,SAVEAREA

Statements 1 and 2 assign to the SETA symbols &A and &B the arithmetic values +10 and +12, respectively. Therefore, statement 3 assigns the SETA symbol &C the arithmetic value -2. When &C is used in statement 5, the arithmetic value -2 is converted to the unsigned integer 2. When &C is used in statement 4, however, the arithmetic value -2 is used. Therefore, &D is assigned the arithmetic value +8. When &D is used in statement 6, the arithmetic value +8 is converted to the unsigned integer 8.

The following example shows how the value assigned to a SETA symbol may be changed in a macro definition.

	Name	Operation	Operand
		MACRO	
	&NAME	MOVE	&TO,&FROM
		LCLA	&A
1	&A	SETA	5
	&NAME	ST	2,SAVEAREA
2		L	2,&FROM&A
3	&A	SETA	8
4		ST	2,&TO&A
		L	2,SAVEAREA
		MEND	
	HERE	MOVE	FIELDA,FIELDB
	HERE	ST	2,SAVEAREA
		L	2,FIELDB5
		ST	2,FIELDA8
		L	2,SAVEAREA

Statement 1 assigns the arithmetic value +5 to SETA symbol &A. In statement 2, &A is converted to the unsigned integer 5. Statement 3 assigns the arithmetic value +8 to &A. In statement 4, therefore, &A is converted to the unsigned integer 8, instead of 5.

A SETA symbol may be used with a symbolic parameter to refer to an operand in an operand sublist. If a SETA symbol is used for this purpose it must have been assigned a value in the range 1 to 100.

Any expression that may be used in the operand of a SETA instruction may be used to refer to an operand in an operand sublist.

Sublists are described in Section 8 under Operand Sublists.

The following macro-definition may be used to add the last operand in an operand sublist to the first operand in an operand sublist and store the result at the first operand. A sample macro-instruction and generated statements follow the macro-definition.

	Name	Operation	Operand
		MACRO	
1		ADDX	&NUMBER,&REG
		LCLA	&LAST
2	&LAST	SETA	N'&NUMBER
		L	&REG,&NUMBER(1)
3		A	&REG,&NUMBER(&LAST)
		ST	&REG,&NUMBER(1)
		MEND	
4		ADDX	(A,B,C,D,E),3
		L	3,A
		A	3,E
		ST	3,A

&NUMBER is the first symbolic parameter in the operand entry of the prototype statement (statement 1). The corresponding characters, (A,B,C,D,E), of the macro-instruction (statement 4) are a sublist. Statement 2 assigns to &LAST the arithmetic value +5, which is equal to the number of operands in the sublist. Therefore, in statement 3, &NUMBER(&LAST) is replaced by the fifth operand of the sublist.

SETC -- SET CHARACTER

The SETC instruction is used to assign a character value to a SETC symbol. The form of this instruction is:

Name	Operation	Operand
A SETC symbol	SETC	One operand, of the form described below

The operand may consist of the type attribute, a character expression, a substring notation, or a concatenation of substring notations and character expressions. A SETA symbol may appear in the operand of a SETC statement. The result is the character representation of the decimal value, unsigned, with leading zeros removed. If the value is zero, one decimal zero is used.

TYPE ATTRIBUTE

The character value assigned to a SETC symbol may be a type attribute. If the type attribute is used, it must appear

alone in the operand field. The following example assigns to the SETC symbol &TYPE the letter that is the type attribute of the macro-instruction operand that corresponds to the symbolic parameter &ABC.

Name	Operation	Operand
&TYPE	SETC	T'&ABC

CHARACTER EXPRESSION

A character expression consists of any combination of characters enclosed in apostrophes. The maximum length of a character expression is 127 characters.

The character value enclosed in apostrophes in the operand field is assigned to the SETC symbol in the name entry. The maximum length character value that can be assigned to a SETC symbol is eight characters. If a value greater than 8 is specified, the leftmost 8 characters will be used.

EVALUATION OF CHARACTER EXPRESSIONS: The following statement assigns the character value AB%4 to the SETC symbol &ALPHA:

Name	Operation	Operand
&ALPHA	SETC	'AB%4'

More than one character expression may be concatenated into a single character expression by placing a period between the terminating apostrophe of one character expression and the opening apostrophe of the next character expression. For example, either of the following statements may be used to assign the character value ABCDEF to the SETC symbol &BETA.

Name	Operation	Operand
&BETA	SETC	'ABCDEF'
&BETA	SETC	'ABC'.'DEF'

Two apostrophes must be used to represent a apostrophe that is part of a character expression.

The following statement assigns the character value L'SYMBOL to the SETC symbol &LENGTH.

Name	Operation	Operand
&LENGTH	SETC	'L''SYMBOL'

Variable symbols may be concatenated with other characters in the operand field of a SETC instruction according to the general rules for concatenating variable symbols with other characters (see Section 7).

If &ALPHA has been assigned the character value AB%4, the following statement may be used to assign the character value AB%4RST to the variable symbol &GAMMA.

Name	Operation	Operand
&GAMMA	SETC	'&ALPHA.RST'

Name	Operation	Operand
&DELTA	SETC	'&ALPHA'.'RST'

Two ampersands must be used to represent an ampersand that is not part of a variable symbol. Both ampersands become part of the character value assigned to the SETC symbol. They are not replaced by a single ampersand.

The following statement assigns the character value HALF&& to the SETC symbol &AND.

Name	Operation	Operand
&AND	SETC	'HALF&&'

In this example,

Name	Operation	Operand
&A	SETC	'&&BETA'(2,5)

'&&BETA'(2,5) produces &BETA which is considered a character string, not a variable symbol.

SUBSTRING NOTATION

The character value assigned to a SETC symbol may be a substring character value. Substring character values permit the pro-

grammer to assign part of a character value to a SETC symbol.

If the programmer wants to assign part of a character value to a SETC symbol, he must indicate to the assembler in the operand of a SETC instruction: (1) the character value itself, and (2) the part of the character value he wants to assign to the SETC symbol. The concatenation of (1) and (2) in the operand of a SETC instruction is called a substring notation. The character value that is assigned to the SETC symbol in the name entry is called a substring character value.

Substring notation consists of a character expression, immediately followed by two arithmetic expressions that are separated from each other by a comma and are enclosed in parentheses. These parentheses count when determining the number of levels of parentheses. The two arithmetic expressions may be any expression that is allowed in the operand of a SETA instruction. They may not be zero.

The first expression indicates the first character (in the character expression) that is to be assigned to the SETC symbol in the name entry. The second expression indicates the number of consecutive characters in the character expression (starting with the character indicated by the first expression) that are to be assigned to the SETC symbol. If a substring specifies more characters than are in the character string, the number of available characters will be supplied.

The maximum size character expression the substring character value can be chosen from is 127 characters.

The following are valid substring notations:

```
'&ALPHA'(2,5)
'AB%4'(&AREA+2,1)
'&ALPHA'.'RST'(6,&A)
'ABC&GAMMA'(&A,&AREA+2)
```

The following are invalid substring notations:

```
'&BETA' (4,6)
      (blanks between character value
        and arithmetic expressions)
'L''SYMBOL'(142-&XYZ)
      (only one arithmetic expression)
'AB%4&ALPHA'(8 &FIELD*2)
      (arithmetic expressions
        not separated by a comma)
'BETA'4,6
      (arithmetic expressions
        not enclosed in parentheses)
'&ALPHA'(2,4)(1,1)
      (double substring notation is not
        permitted)
```

CONCATENATING SUBSTRING NOTATIONS AND CHARACTER EXPRESSIONS: Substring notations may be concatenated with character expressions in the operand of a SETC instruction. If a substring notation follows a character expression, the two may be concatenated by placing a period between the terminating apostrophe of the character expression and the opening apostrophe of the substring notation.

For example, if &ALPHA has been assigned the character value AB%4, and &BETA has been assigned the character value ABCDEF, then the following statement assigns &GAMMA the character value AB%4BCD.

```
|Name    |Operation |Operand                  |
|--------|----------|-------------------------|
|&GAMMA  |SETC      |'&ALPHA'.'&BETA'(2,3)    |
```

If a substring notation precedes a character expression or another substring notation, the two may be concatenated by writing the opening apostrophe of the second item immediately after the closing parenthesis of the substring notation.

The programmer may optionally place a period between the closing parenthesis of a substring notation and the opening apostrophe of the next item in the operand.

If &ALPHA has been assigned the character value AB%4, and &ABC has been assigned the character value 5RS, either of the following statements may be used to assign &WORD the character value AB%45RS.

```
|Name  |Operation|Operand                        |
|------|---------|-------------------------------|
|&WORD |SETC     |'&ALPHA'(1,4)'&ABC'            |
|&WORD |SETC     |'&ALPHA'(1,4)'&ABC'(1,3)       |
```

If a SETC symbol is used in the operand of a SETA instruction, the character value assigned to the SETC symbol must be one to eight decimal digits.

If a SETA symbol is used in the operand of a SETC statement, the arithmetic value is converted to an unsigned integer with leading zeros removed. If the value is zero, it is converted to a single zero.

Using SETC Symbols

The character value assigned to a SETC symbol is substituted for the SETC symbol when it is used in the name, operation, or operand of a statement.

For example, consider the following macro-definition, macro-instruction, and generated statements.

	Name	Operation	Operand
		MACRO	
	&NAME	MOVE	&TO,&FROM
		LCLC	&PREFIX
1	&PREFIX	SETC	'FIELD'
	&NAME	ST	2,SAVEAREA
2		L	2,&PREFIX&FROM
3		ST	2,&PREFIX&TO
		L	2,SAVEAREA
		MEND	
	HERE	MOVE	A,B
	HERE	ST	2,SAVEAREA
		L	2,FIELDB
		ST	2,FIELDA
		L	2,SAVEAREA

Statement 1 assigns the character value FIELD to the SETC symbol &PREFIX. In statements 2 and 3, &PREFIX is replaced by FIELD.

The following example shows how the value assigned to a SETC symbol may be changed in a macro-definition.

	Name	Operation	Operand
		MACRO	
	&NAME	MOVE	&TO,&FROM
		LCLC	&PREFIX
1	&PREFIX	SETC	'FIELD'
	&NAME	ST	2,SAVEAREA
2		L	2,&PREFIX&FROM
3	&PREFIX	SETC	'AREA'
4		ST	2,&PREFIX&TO
		L	2,SAVEAREA
		MEND	
	HERE	MOVE	A,B
	HERE	ST	2,SAVEAREA
		L	2,FIELDB
		ST	2,AREAA
		L	2,SAVEAREA

Statement 1 assigns the character value FIELD to the SETC symbol &PREFIX. Therefore, &PREFIX is replaced by FIELD in statement 2. Statement 3 assigns the character value AREA to &PREFIX. Therefore, &PREFIX is replaced by AREA, instead of FIELD, in statement 4.

The following example illustrates the use of a substring notation as the operand field of a SETC instruction.

	Name	Operation	Operand
		MACRO	
	&NAME	MOVE	&TO,&FROM
		LCLC	&PREFIX
1	&PREFIX	SETC	'&TO'(1,5)
	&NAME	ST	2,SAVEAREA
2		L	2,&PREFIX&FROM
		ST	2,&TO
		L	2,SAVEAREA
		MEND	
	HERE	MOVE	FIELDA,B
	HERE	ST	2,SAVEAREA
		L	2,FIELDB
		ST	2,FIELDA
		L	2,SAVEAREA

Statement 1 assigns the substring character value FIELD (the first five characters corresponding to symbolic parameter &TO) to the SETC symbol &PREFIX. Therefore, FIELD replaces &PREFIX in statement 2.

SETB -- SET BINARY

The SETB instruction may be used to assign the binary value 0 or 1 to a SETB symbol. The form of this instruction is:

Name	Operation	Operand
A SETB symbol	SETB	A 0 or a 1,(0) or (1), or a logical expression enclosed in parentheses

The operand may contain a 0 or a 1 or a logical expression enclosed in parentheses. (No explicit binary zeros or ones are allowed in parentheses other than in the form (0) or (1).) A logical expression is evaluated to determine if it is true or false; the SETB symbol in the name entry is then assigned the binary value 1 or 0 corresponding to true or false, respectively.

Note: The parentheses enclosing a logical expression do not count towards the parenthesis level limit.

A logical expression consists of one term or a logical combination of terms. The terms that may be used alone or in combination with each other are arithmetic relations, character relations, and SETB symbols. The logical operators used to combine the terms of an expression are AND, OR, and NOT.

A logical expression may not contain two terms in succession. A logical expression may contain two operators in succession only if the first operator is either AND or OR and the second operator is NOT. A logical expression may begin with the operator NOT. It may not begin with the operators AND or OR.

An arithmetic relation consists of two arithmetic expressions connected by a relational operator. A character relation consists of two character strings connected by a relational operator. The relational operators are EQ (equal), NE (not equal), LT (less than), GT (greater than), LE (less than or equal), and GE (greater than or equal).

Any expression that may be used in the operand of a SETA instruction, may be used as an arithmetic expression in the operand of a SETB instruction. Anything that may be used in the operand of a SETC instruction, may be used as a character string in the operand of a SETB instruction. This includes substring and type attribute notations. The maximum size of the character values that can be compared is 127 characters. If the two character values are of unequal length, then the shorter one will always compare less than the longer one, regardless of the characters present.

The relational and logical operators must be immediately preceded and followed by at least one blank or other special character. Each relation may or may not be enclosed in parentheses. If a relation is not enclosed in parentheses, it must be separated from the logical operators by at least one blank or other special character.

The following are valid operand fields of SETB instructions:

```
1
(&AREA+2 GT 29)
('AB%4' EQ '&ALPHA')
(T'&ABC NE T'&XYZ)
(T'&P12 EQ 'F')
(&AREA+2 GT 29 OR &B)
(NOT &B AND &AREA+X'2D' GT 29)
('&C'EQ'MB')
```

The following are invalid operand fields of SETB instructions:

```
&B              (not enclosed in parentheses)

(T'&P12 EQ 'F' &B)
                (two terms in succession)
('AB%4' EQ 'ALPHA' NOT &B)
                (the NOT operator must be
                  preceded by AND or OR)
(AND T'&P12 EQ 'F')
                (expression begins with AND)
```

Evaluation of Logical Expressions

The following procedure is used to evaluate a logical expression in the operand field of a SETB instruction:

1. Each term (i.e., arithmetic relation, character relation, or SETB symbol) is evaluated and given its logical value (true or false).
2. The logical operations are performed moving from left to right. However, NOTs are performed before ANDs, and ANDs are performed before ORs.
3. The computed result is the value assigned to the SETB symbol in the name field.

The logical expression in the operand of a SETB instruction may contain one or more sequences of logically combined terms that are enclosed in parentheses. A sequence of parenthesized terms may appear within another parenthesized sequence.

The following are examples of SETB instruction operands that contain parenthesized sequences of terms.

```
(NOT(&B AND &AREA+X'2D' GT 29))
(&B AND(T'&P12 EQ'F'OR&B)
```

The parenthesized portion or portions of a logical expression are evaluated before the rest of the terms in the expression are evaluated. If a sequence of parenthesized terms appears within another parenthesized sequence, the innermost sequence is evaluated first.

Logical expressions may have only five levels of parentheses. Subscripting, substring notation, and logical expression nesting count when determining the level of parentheses. The parentheses surrounding the SETB operand do not count. A counter is maintained for each statement and is

increased by one for each occurrence of a variable symbol and an operation entry. The maximum value this counter may attain is 35. See Appendix H.

Using SETB Symbols

The logical value assigned to a SETB symbol is used for the SETB symbol appearing in the operand of an AIF instruction or another SETB instruction.

If a SETB symbol is used in the operand of a SETA instruction, or in arithmetic relations in the operands of AIF and SETB instructions, the binary values 1 (true) and 0 (false) are converted to the arithmetic values +1 and +0, respectively.

If a SETB symbol is used in the operand of a SETC instruction, in character relations in the operands of AIF and SETB instructions, or in any other statement, the binary values 1 (true) and 0 (false), are converted to the character values 1 and 0, respectively.

The following example illustrates these rules. It is assumed that L'&TO EQ 4 is true, and S'&TO EQ 0 is false.

```
   |Name   |Operation |Operand              |
   |-------|----------|---------------------|
   |       |MACRO     |                     |
   |&NAME  |MOVE      |&TO,&FROM            |
   |       |LCLA      |&A1                  |
   |       |LCLB      |&B1,&B2              |
   |       |LCLC      |&C1                  |
1  |&B1    |SETB      |(L'&TO EQ 4)         |
2  |&B2    |SETB      |(S'&TO EQ 0)         |
3  |&A1    |SETA      |&B1                  |
4  |&C1    |SETC      |'&B2'                |
   |&NAME  |ST        |2,SAVEAREA           |
   |       |L         |2,&FROM&A1           |
   |       |ST        |2,&TO&C1             |
   |       |L         |2,SAVEAREA           |
   |       |MEND      |                     |
   |-------|----------|---------------------|
   |HERE   |MOVE      |FIELDA,FIELDB        |
   |-------|----------|---------------------|
   |HERE   |ST        |2,SAVEAREA           |
   |       |L         |2,FIELDB1            |
   |       |ST        |2,FIELDA0            |
   |       |L         |2,SAVEAREA           |
```

Because the operand of statement 1 is true, &B1 is assigned the binary value 1. Therefore, the arithmetic value +1 is substituted for &B1 in statement 3. Because the operand of statement 2 is false, &B2 is assigned the binary value 0. Therefore, the character value 0 is substituted for &B2 in statement 4.

AIF -- CONDITIONAL BRANCH

The AIF instruction is used to alter conditionally the sequence in which source program statements are processed by the assembler. The typical form of this instruction is:

```
|Name      |Operation|Operand                  |
|----------|---------|-------------------------|
|A se-     |AIF      |A logical expression     |
|quence     |         |enclosed in paren-       |
|symbol or |         |theses, immediately      |
|not used  |         |followed by a            |
|          |         |sequence symbol          |
```

Any logical expression that may be used in the operand of a SETB instruction may be used in the operand of an AIF instruction. However, the forms

```
AIF    (0), sequence symbol    and
AIF    (1), sequence symbol
```

are invalid. The sequence symbol in the operand must immediately follow the closing parenthesis of the logical expression. AIF operand entries must not contain explicit zeros or ones.

Note: The parentheses enclosing the logical expression do not count toward the level limit.

The logical expression in the operand is evaluated to determine if it is true or false. If the expression is true, the statement named by the sequence symbol in the operand is the next statement processed by the assembler; however, sequence checking is not affected. If the expression is false, the next sequential statement is processed by the assembler.

The statement named by the sequence symbol may precede or follow the AIF instruction.

If an AIF instruction is in a macro-definition, then the sequence symbol in the operand must appear in the name entry of a statement in the definition. If an AIF instruction appears outside macro-definitions, then the sequence symbol in the operand must appear in the name entry of a statement outside macro-definitions.

The following are valid operands of AIF instructions:

```
(&AREA+X'2D' GT 29).READER
(T'&P12 EQ 'F').THERE
```

The following are invalid operands of AIF instructions:

```
(T'&ABC NE T'&XYZ)   (no sequence symbol)
.X4F2                (no logical expression)
(T'&ABC NE T'&XYZ) .X4F2
                     (blanks between logical
                       expression and se-
                       quence symbol)
```

The following macro-definition may be used to generate the statements needed to move a full-word fixed-point number from one storage area to another. The statements will be generated only if the type attribute of both storage areas is the letter F.

```
   |Name |Operation|Operand                |
   |-----|---------|-----------------------|
   |     |MACRO    |                       |
   |&N   |MOVE     |&T,&F                  |
 1 |     |AIF      |(T'&T NE T'&F).END     |
 2 |     |AIF      |(T'&T NE 'F').END      |
 3 |&N   |ST       |2,SAVEAREA             |
   |     |L        |2,&F                   |
   |     |ST       |2,&T                   |
   |     |L        |2,SAVEAREA             |
 4 |.END |MEND     |                       |
```

The logical expression in the operand of statement 1 has the value true if the type attributes of the two macro-instruction operands are not equal. If the type attributes are equal, the expression has the logical value false.

Therefore, if the type attributes are not equal, statement 4 (the statement named by the sequence symbol .END) is the next statement processed by the assembler. If the type attributes are equal, statement 2 (the next sequential statement) is processed.

The logical expression in the operand of statement 2 has the value true if the type attribute of the first macro-instruction operand is not the letter F. If the type attribute is the letter F, the expression has the logical value false.

Therefore, if the type attribute is not the letter F, statement 4 (the statement named by the sequence symbol .END) is the next statement processed by the assembler. If the type attribute is the letter F, statement 3 (the next sequential statement) is processed.

AGO -- UNCONDITIONAL BRANCH

The AGO instruction is used to unconditionally alter the sequence in which source program statements are processed by the assembler. The typical form of this instruction is:

```
|Name       |Operation|Operand               |
|-----------|---------|----------------------|
|A sequence |AGO      |A sequence symbol     |
|symbol or  |         |                      |
|not used   |         |                      |
```

The statement named by the sequence symbol in the operand is the next statement processed by the assembler.

The statement named by the sequence symbol may precede or follow the AGO instruction.

If an AGO instruction is part of a macro-definition, then the sequence symbol in the operand must appear in the name entry of a statement that is in that definition. If an AGO instruction appears outside macro-definitions, then the sequence symbol in the operand must appear in the name entry of a statement outside macro-definitions.

The following example illustrates the use of the AGO instruction.

```
   |Name   |Operation|Operand                |
   |-------|---------|-----------------------|
   |       |MACRO    |                       |
   |&NAME  |MOVE     |&T,&F                  |
 1 |       |AIF      |(T'&T EQ 'F').FIRST    |
 2 |       |AGO      |.END                   |
 3 |.FIRST |AIF      |(T'&T NE T'&F).END     |
   |&NAME  |ST       |2,SAVEAREA             |
   |       |L        |2,&F                   |
   |       |ST       |2,&T                   |
   |       |L        |2,SAVEAREA             |
 4 |.END   |MEND     |                       |
```

Statement 1 is used to determine if the type attribute of the first macro-instruction operand is the letter F. If the type attribute is the letter F, statement 3 is the next statement processed by the assembler. If the type attribute is not the letter F, statement 2 is the next statement processed by the assembler.

Statement 2 is used to indicate to the assembler that the next statement to be

processed is statement 4 (the statement named by sequence symbol .END).

ACTR -- CONDITIONAL ASSEMBLY LOOP COUNTER

The ACTR instruction is used to limit the number of AGO and AIF branches executed within a macro-definition or within the main source program.

A separate ACTR statement may be used in each macro-definition and in the main program. These counters are independent.

The form of this instruction is:

Name	Operation	Operand
Not used must not be present	ACTR	Any valid SETA expression

This statement must immediately follow any global or local declarations, if any. This statement causes a counter to be set to the value in its operand. Each time an AGO or AIF branch is executed, the counter is decremented by one. If the count is zero before decrementing, the assembler takes one of two actions:

1. If a macro definition is being processed, the processing of it and any macros above it in a nest is terminated, and the next statement in the main portion of the program is processed.

2. If the main portion of the program is being processed, conditional assembly is terminated, and the portion of the program generated so far is assembled.

If an ACTR statement is not given, the assumed value of the counter is 150 for the D assembler and 4096 for the F assembler.

ANOP -- ASSEMBLY NO OPERATION

The ANOP instruction facilitates conditional and unconditional branching to statements named by symbols or variable symbols.

The typical form of this instruction is:

Name	Operation	Operand
A sequence symbol	ANOP	Not used, must not be present

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

The following example illustrates the use of the ANOP instruction.

```
   Name     Operation  Operand
            MACRO
   &NAME    MOVE       &T,&F
            LCLC       &TYPE
1           AIF        (T'&T EQ 'F').FTYPE
2  &TYPE    SETC       'E'
3  .FTYPE   ANOP
4  &NAME    ST&TYPE    2,SAVEAREA
            L&TYPE     2,&F
            ST&TYPE    2,&T
            L&TYPE     2,SAVEAREA
            MEND
```

Statement 1 is used to determine if the type attribute of the first macro-instruction operand is the letter F. If the type attribute is not the letter F, statement 2 is the next statement processed by the assembler. If the type attribute is the letter F, statement 4 should be processed next. However, since there is a variable symbol (&NAME) in the name field of statement 4, the required sequence symbol (.FTYPE) cannot be placed in the name field. Therefore, an ANOP instruction (statement 3) must be placed before statement 4.

Then, if the type attribute of the first operand is the letter F, the next statement processed by the assembler is the statement named by sequence symbol .FTYPE. The value of &TYPE retains its initial null character value because the SETC instruction is not processed. Since .FTYPE names an ANOP

instruction, the next statement processed by the assembler is statement 4, the statement following the ANOP instruction.

CONDITIONAL ASSEMBLY ELEMENTS

The following chart summarizes the elements that can be used in each conditional assembly instruction. Each row in this chart indicates which elements can be used in a single conditional assembly instruction. Each column is used to indicate the conditional assembly instructions in which a particular element can be used.

The intersection of a column and a row indicates whether an element can be used in an instruction, and if so, in what fields of the instruction the element can be used. For example, the intersection of the first row and the first column of the chart indicates that symbolic parameters can be used in the operand field of SETA instructions.

	Variable Symbols				Attributes						
		SET Symbols									
	S.P.	SETA	SETB	SETC	T'	L'	S'	I'	K'	N'	S.S.
SETA	O	N,O	O	O[3]		O	O	O	O	O	
SETB	O	O	N,O	O	O[1]	O[2]	O[2]	O[2]	O[2]	O[2]	
SETC	O	O	O	N,O	O						
AIF	O	O	O	O	O[1]	O[2]	O[2]	O[2]	O[2]	O[2]	N,O
AGO											N,O
ANOP											N
ACTR	O	O	O	O[3]		O	O	O	O	O	

[1] Only in character relations
[2] Only in arithmetic relations
[3] Only if one to eight decimal digits

Abbreviations

N	is Name	L'	is Length Attribute	K'	is Count Attribute
O	is Operand	S'	is Scaling Attribute	N'	is Number Attribute
S.P.	is Symbolic Parameter	I'	is Integer Attribute	S.S.	is Sequence Symbol

SECTION 10: ADDITIONAL FEATURES

The additional features of the assembler language allow the programmer to:

1. Terminate processing of a macro-definition.
2. Generate error messages.
3. Define global SET symbols.
4. Define subscripted SET symbols.
5. Use system variable symbols.
6. Prepare keyword and mixed-mode macro-definitions and write keyword and mixed-mode macro-instructions.

MEXIT -- MACRO-DEFINITION EXIT

The MEXIT instruction is used to indicate to the assembler that it should terminate processing of a macro-definition. The typical form of this instruction is:

Name	Operation	Operand
A sequence symbol or not used	MEXIT	Not used, must not be present

The MEXIT instruction may only be used in a macro-definition.

If the assembler processes an MEXIT instruction that is in a macro-definition corresponding to an outer macro-instruction, the next statement processed by the assembler is the next statement outside macro-definitions.

If the assembler processes an MEXIT instruction that is in a macro-definition corresponding to a second or third level macro-instruction, the next statement processed by the assembler is the next statement after the second or third level macro-instruction in the macro-definition, respectively.

MEXIT should not be confused with MEND. MEND indicates the end of a macro-definition. MEND must be the last statement of every macro-definition, including those that contain one or more MEXIT instructions.

The following example illustrates the use of the MEXIT instruction.

	Name	Operation	Operand
		MACRO	
	&NAME	MOVE	&T,&F
1		AIF	(T'&T EQ 'F').OK
2		MEXIT	
3	.OK	ANOP	
	&NAME	ST	2,SAVEAREA
		L	2,&F
		ST	2,&T
		L	2,SAVEAREA
		MEND	

Statement 1 is used to determine if the type attribute of the first macro-instruction operand is the letter F. If the type attribute is the letter F, the assembler processes the remainder of the macro-definition starting with statement 3. If the type attribute is not the letter F, the next statement processed by the assembler is statement 2. Statement 2 indicates to the assembler that it is to terminate processing of the macro-definition.

MNOTE STATEMENT

The MNOTE instruction may be used to generate a message and to indicate what error severity code, if any, is to be associated with the message. The severity code is for the programmer's information only and is not used by the DOS assembler or control program. The typical form of this instruction is:

Name	Operation	Operand
A sequence symbol or not used	MNOTE	See examples below.

The operand entry of the MNOTE assembler-instruction may be written in one of the following forms:

1. severity-code,'message'
2. ,'message'
3. 'message'

For 2 and 3 above, the severity code is assumed to be one.

The MNOTE instruction may only be used in a macro definition. Variable symbols may be used to generate the MNOTE mnemonic operation code, the severity code indicator, and the message.

The resulting severity code indicator may be a decimal integer 0 to 255, blank, or an asterisk. The integers indicate the severity of the error. (0 is the least severe; 255 is the most severe). If the severity code indicator is blank or omitted, 1 is assumed. If the severity code is an asterisk, the MNOTE is not considered an error message. Messages can be generated with substitution using variable symbols.

The MNOTE statement appears in the listing with a statement number at the point where it was generated. It appears even if PRINT NOGEN is specified. If the severity code indicator was an integer or a blank, this statement number is placed in a list of statement numbers of MNOTE and other error statements near the end of the assembly listing. If the severity code is an asterisk, the statement number is not placed in this list.

Since the message portion of the MNOTE operand is enclosed in apostrophes, two apostrophes must be used to represent a single apostrophe. Any variable symbols used in the message operand are replaced by values assigned to them. Two ampersands must be used to represent a single ampersand that is not part of a variable symbol.

The following example illustrates the use of the MNOTE instruction.

```
   | Name   | Operation | Operand
   |        | MACRO     |
   | &NAME  | MOVE      | &T,&F
1  |        | AIF       | (T'&T NE T'&F).M1
2  |        | AIF       | (T'&T NE 'F').M2
3  | &NAME  | ST        | 2,SAVEAREA
   |        | L         | 2,&F
   |        | ST        | 2,&T
   |        | L         | 2,SAVEAREA
4  |        | MNOTE     | *,'MOVE GENERATED'
   |        | MEXIT     |
5  | .M1    | MNOTE     | 8,'TYPE NOT SAME'
   |        | MEXIT     |
6  | .M2    | MNOTE     | 8,'TYPE NOT F'
   |        | MEND      |
```

Statement 1 is used to determine if the type attributes of both macro instruction operands are the same. If they are, statement 2 is the next statement processed by the assembler. If they are not, statement 5 is the next statement processed by the assembler. Statement 5 causes an error message -- 8,TYPE NOT SAME -- to be printed in the source program listing.

Statement 2 is used to determine if the type attribute of the first macro instruction operand is the letter F. If the type attribute is the letter F, statement 3 is the next statement processed by the assembler. If the attribute is not the letter F, statement 6 is the next statement processed by the assembler. Statement 6 causes an error message -- 8,TYPE NOT F -- to be printed in the source program listing. Statement 4 is an MNOTE which is not treated as an error message.

GLOBAL AND LOCAL VARIABLE SYMBOLS

The following are local variable symbols:

1. Symbolic parameters.
2. Local SET symbols.
3. System variable symbols.

Global SET symbols are the only global variable symbols.

The GBLA, GBLB, and GBLC instructions define global SET symbols, just as the LCLA, LCLB, and LCLC instructions define the SET symbols described in Section 9. Hereinafter, SET symbols defined by LCLA, LCLB, and LCLC instructions will be called local SET symbols.

Global SET symbols may communicate values between statements in one or more macro definitions and statements outside macro definitions. However, local SET symbols communicate values between statements in the same macro definition, or between statements outside macro definitions.

If a local SET symbol is defined in two or more macro definitions, or in a macro definition and outside macro definitions, the SET symbol is considered to be a different SET symbol in each case. However, a global SET symbol is the same SET symbol each place it is defined.

A SET symbol must be defined as a global SET symbol in each macro definition in which it is to be used as a global SET symbol. A SET symbol must be defined as a global SET symbol outside macro definitions, if it is to be used as a global SET symbol outside macro definitions.

If the same SET symbol is defined as a global SET symbol in one or more places, and as a local SET symbol elsewhere, it is considered the same symbol wherever it is

defined as a global SET symbol, and a different symbol wherever it is defined as as local SET symbol.

Defining Local and Global SET Symbols

Local SET symbols are defined when they appear in the operand entry of an LCLA, LCLB, or LCLC instruction. These instructions are discussed in Section 9 under Defining SET Symbols.

Global SET symbols are defined when they appear in the operand entry of a GBLA, GBLB, or GBLC instruction. The typical forms of these instructions are:

Name	Operation	Operand
Not used, must not be present	GBLA, GBLB, or GBLC	One or more variable symbols that are to be used as global SET symbols, separated by commas

The GBLA, GBLB, and GBLC instructions define global SETA, SETB, and SETC symbols, respectively, and assign the same initial values as the corresponding types of local SET symbols. However, a global SET symbol is assigned an initial value by only the first GBLA, GBLB, or GBLC instruction processed in which the symbol appears. Subsequent GBLA, GBLB, or GBLC instructions processed by the assembler do not affect the value assigned to the SET symbol.

The programmer should not define any global SET symbols whose first four characters are &SYS.

If a GBLA, GBLB, or GBLC instruction is part of a macro definition, it must immediately follow the prototype statement, or another GBLA, GBLB, or GBLC instruction. GBLA, GBLB, and GBLC instructions outside macro definitions must appear after all macro definitions in the source program, before all conditional assembly instructions and PUNCH and REPRO statements outside macro definitions, and before the first control section of the program.

All GBLA, GBLB, and GBLC instructions in a macro definition must appear before all LCLA, LCLB, and LCLC instructions in that macro definition. All GBLA, GBLB, and GBLC instructions outside macro definitions must appear before all LCLA, LCLB, and LCLC instructions outside macro definitions.

Using Global and Local SET Symbols

The following examples illustrate the use of global and local SET symbols. Each example consists of two parts. The first part is an assembler language source program. The second part shows the statements that would be generated by the assembler after it processed the statements in the source program.

Example 1: This example illustrates how the same SET symbol can be used to communicate (1) values between statements in the same macro definitions, and (2) different values between statements outside macro definitions.

```
   Name     Operation   Operand
            MACRO
   &NAME    LOADA
1           LCLA        &A
2  &NAME    LR          15,&A
3  &A       SETA        &A+1
            MEND

4           LCLA        &A
   FIRST    LOADA
5           LR          15,&A
            LOADA
6           LR          15,&A
            END         FIRST
   ---------------------------
   FIRST    LR          15,0
            LR          15,0
            LR          15,0
            LR          15,0
            END         FIRST
```

&A is defined as a local SETA symbol in a macro definition (statement 1) and outside macro definitions (statement 4). &A is used twice within macro definition (statements 2 and 3) and twice outside macro definitions (statements 5 and 6).

Since &A is a local SETA symbol in the macro definition and outside macro definitions, it is one SETA symbol in the macro definition, and another SETA symbol outside macro definitions. Therefore, statement 3 (which is in the macro definition) does not affect the value used for &A in statements 5 and 6 (which are outside macro definitions).

Example 2: This example illustrates how a SET symbol can be used to communicate values between statements that are part of a macro definition and statements outside macro definitions.

	Name	Operation	Operand
		MACRO	
	&NAME	LOADA	
1		GBLA	&A
2	&NAME	LR	15,&A
3	&A	SETA	&A+1
		MEND	
4		GBLA	&A
	FIRST	LOADA	
5		LR	15,&A
		LOADA	
6		LR	15,&A
		END	FIRST
	FIRST	LR	15,0
		LR	15,1
		LR	15,1
		LR	15,2
		END	FIRST

&A is defined as a global SETA symbol in a macro-definition (statement 1) and outside macro-definitions (statement 4). &A is used twice within the macro-definition (statements 2 and 3) and twice outside macro-definitions (statements 5 and 6).

Since &A is a global SETA symbol in the macro-definition and outside macro-definitions, it is the same SETA symbol in both cases. Therefore, statement 3 (which is in the macro-definition) affects the value used for &A in statements 5 and 6 (which are outside macro-definitions).

Example 3: This example illustrates how the same SET symbol can be used to communicate: (1) values between statements in one macro-definition, and (2) different values between statements in a different macro-definition.

&A is defined as a local SETA symbol in two different macro-definitions (statements 1 and 4). &A is used twice within each macro-definition (statements 2,3,5 and 6).

Since &A is a local SETA symbol in each macro-definition, it is one SETA symbol in one macro-definition, and another SETA symbol in the other macro-definition. Therefore, statement 3 (which is in one macro-definition) does not affect the value used for &A in statement 5 (which is in the other macro-definition). Similarly, statement 6 does not affect the value used for &A in statement 2.

	Name	Operation	Operand
		MACRO	
	&NAME	LOADA	
1		LCLA	&A
2	&NAME	LR	15,&A
3	&A	SETA	&A+1
		MEND	
		MACRO	
		LOADB	
4		LCLA	&A
5		LR	15,&A
6	&A	SETA	&A+1
		MEND	
	FIRST	LOADA	
		LOADB	
		LOADA	
		LOADB	
		END	FIRST
	FIRST	LR	15,0
		LR	15,0
		LR	15,0
		LR	15,0
		END	FIRST

Example 4: This example illustrates how a SET symbol can be used to communicate values between statements that are part of two different macro-definitions.

	Name	Operation	Operand
		MACRO	
	&NAME	LOADA	
1		GBLA	&A
2	&NAME	LR	15,&A
3	&A	SETA	&A+1
		MEND	
		MACRO	
		LOADB	
4		GBLA	&A
5		LR	15,&A
6	&A	SETA	&A+1
		MEND	
	FIRST	LOADA	
		LOADB	
		LOADA	
		LOADB	
		END	FIRST
	FIRST	LR	15,0
		LR	15,1
		LR	15,2
		LR	15,3
		END	FIRST

&A is defined as a global SETA symbol in two different macro-definitions (statements 1 and 4). &A is used twice within each macro-definition (statements 2,3,5, and 6).

Since &A is a global SETA symbol in each macro-definition, it is the same SETA symbol in each macro-definition. Therefore, statement 3 (which is in one macro-definition) affects the value used for &A in statement 5 (which is in the other macro-definition). Similarly, statement 6 affects the value used for &A in statement 2.

Example 5: This example illustrates how the same SET symbol can be used to communicate: (1) values between statements in two different macro-definitions, and (2) different values between statements outside macro-definitions.

```
   |Name    |Operation |Operand             |
   |--------|----------|--------------------|
   |        |MACRO     |                    |
   |&NAME   |LOADA     |                    |
1  |        |GBLA      |&A                  |
2  |&NAME   |LR        |15,&A               |
3  |&A      |SETA      |&A+1                |
   |        |MEND      |                    |
   |        |          |                    |
   |        |MACRO     |                    |
   |        |LOADB     |                    |
4  |        |GBLA      |&A                  |
5  |        |LR        |15,&A               |
6  |&A      |SETA      |&A+1                |
   |        |MEND      |                    |
   |        |          |                    |
7  |        |LCLA      |&A                  |
   |FIRST   |LOADA     |                    |
   |        |LOADB     |                    |
8  |        |LR        |15,&A               |
   |        |LOADA     |                    |
   |        |LOADB     |                    |
9  |        |LR        |15,&A               |
   |        |END       |FIRST               |
   |--------|----------|--------------------|
   |FIRST   |LR        |15,0                |
   |        |LR        |15,1                |
   |        |LR        |15,0                |
   |        |LR        |15,2                |
   |        |LR        |15,3                |
   |        |LR        |15,0                |
   |        |END       |FIRST               |
```

&A is defined as a global SETA symbol in two different macro-definitions (statements 1 and 4), but it is defined as a local SETA symbol outside macro-definitions (statement 7). &A is used twice within each macro-definition and twice outside macro-definitions (statements 2,3,5,6,8, and 9).

Since &A is a global SETA symbol in each macro-definition, it is the same SETA symbol in each macro-definition. However, since &A is a local SETA symbol outside macro-definitions, it is a different SETA symbol outside macro-definitions.

Therefore, statement 3 (which is in one macro-definition) affects the value used for &A in statement 5 (which is in the other macro-definition), but it does not affect the value used for &A in statements 8 and 9 (which are outside macro-definitions). Similarly, statement 6 affects the value used for &A in statement 2, but it does not affect the value used for &A in statements 8 and 9.

Subscripted SET Symbols

Both global and local SET symbols may be defined as subscripted SET symbols. The local SET symbols defined in Section 9 were all nonsubscripted SET symbols.

Subscripted SET symbols provide the programmer with a convenient way to use one SET symbol plus a subscript to refer to many arithmetic, binary, or character values.

A subscripted SET symbol consists of a SET symbol immediately followed by a subscript that is enclosed in parentheses. The subscript may be any arithmetic expression that is allowed in the operand of a SETA statement in the range of 1 to the specified dimension.

Only five levels of parentheses are permitted in a SETA or SETB operand.

The following are valid subscripted SET symbols.

```
&READER(17)
&A23456(&S4)
&X4F2(25+&A2)
```

The following are invalid subscripted SET symbols.

```
&X4F2          (no subscript)
(25)           (no SET symbol)
&X4F2 (25)     (subscript does not
                 immediately follow
                 SET symbol)
```

Defining Subscripted SET Symbols: If the programmer wants to use a subscripted SET symbol, he must write in a GBLA, GBLB, GBLC, LCLA, LCLB, or LCLC instruction, a SET symbol immediately followed by an unsigned decimal integer enclosed in parentheses. The decimal integer, called a

dimension, indicates the number of SET variables associated with the SET symbol. Every variable associated with a SET symbol is assigned an initial value that is the same as the initial value assigned to the corresponding type of nonsubcripted SET symbol.

If a subscripted SET symbol is defined as global, the same dimension must be used with the SET symbol each time it is defined as global.

The maximum dimension that can be used with a SETA, SETB, or SETC symbol is 255.

A subscripted SET symbol may be used only if the declaration was subscripted. A nonsubscripted SET symbol may be used only if the declaration had no subscript.

The following statements define the global SET symbols &SBOX, &WBOX, and &PSW, and the local SET symbol &TSW. &SBOX has 50 arithmetic variables associated with it, &WBOX has 20 character variables, &PSW and &TSW each have 230 binary variables.

```
|Name  |Operation |Operand              |
|------|----------|---------------------|
|      |GBLA      |&SBOX(50)            |
|      |GBLC      |&WBOX(20)            |
|      |GBLB      |&PSW(230)            |
|      |LCLB      |&TSW(230)            |
```

Using Subscripted SET Symbols: After the programmer has associated a number of SET variables with a SET symbol, he may assign values to each of the variables and use them in other statements.

If the statements in the previous example were part of a macro-definition, (and &A was defined as a SETA symbol in the same definition), the following statements could be part of the same macro-definition.

```
  |Name      |Operation |Operand              |
  |----------|----------|---------------------|
1 |&A        |SETA      |5                    |
2 |&PSW(&A)  |SETB      |(6 LT 2)             |
3 |&TSW(9)   |SETB      |(&PSW(&A))           |
4 |          |A         |2,=F'&SBOX(45)'      |
5 |          |CLI       |AREA,C'&WBOX(17)'    |
```

Statement 1 assigns the arithmetic value 5 to the nonsubscripted SETA symbol &A. Statements 2 and 3 then assign the binary value 0 to subscripted SETB symbols &PSW(5) and &TSW(9), respectively. Statements 4 and 5 generate statements that add the value assigned to &SBOX(45) to general register 2, and compare the value assigned to &WBOX(17) to the value stored at AREA, respectively.

SYSTEM VARIABLE SYMBOLS

System variable symbols are local variable symbols that are assigned values automatically by the assembler. There are three system variable symbols: &SYSNDX, &SYSECT, and &SYSLIST. System variable symbols may be used in the name, operation and operand entries of statements in macro-definitions, but not in statements outside macro-definitions. They may not be defined as symbolic parameters or SET symbols, nor may they be assigned values by SETA, SETB, and SETC instructions.

&SYSNDX -- Macro-Instruction Index

The system variable symbol &SYSNDX may be combined with other characters to create unique names for statements generated from the same model statement.

&SYSNDX is assigned the four-digit number 0001 for the first macro-instruction processed by the assembler, and it is incremented by one for each subsequent inner and outer macro-instruction processed.

If &SYSNDX is used in a model statement, SETC or MNOTE instruction, or a character relation in a SETB or AIF instruction, the value substituted for &SYSNDX is the four-digit number of the macro-instruction being processed, including leading zeros.

If &SYSNDX appears in arithmetic expressions (e.g., in the operand of a SETA instruction), the value used for &SYSNDX is an arithmetic value.

Throughout one use of a macro definition, the value of &SYSNDX may be considered a constant, independent of any inner macro-instruction in that definition.

The example in the next column illustrates these rules. It is assumed that the first macro-instruction processed, OUTER 1, is the 106th macro-instruction processed by the assembler.

Statement 7 is the 106th macro-instruction processed. Therefore, &SYSNDX is assigned the number 0106 for that macro-instruction. The number 0106 is substituted for &SYSNDX when it is used in statements 4 and 6. Statement 4 is used to assign the character value 0106 to the SETC symbol &NDXNUM. Statement 6 is used to create the unique name B0106.

```
   |Name       |Operation |Operand          |
   |-----------|----------|-----------------|
   |           |MACRO     |                 |
   |           |INNER1    |                 |
   |           |GBLC      |&NDXNUM          |
1  |A&SYSNDX   |SR        |2,5              |
   |           |CR        |2,5              |
2  |           |BE        |B&NDXNUM         |
3  |           |B         |A&SYSNDX         |
   |           |MEND      |                 |
   |           |          |                 |
   |           |MACRO     |                 |
   |&NAME      |OUTER1    |                 |
   |           |GBLC      |&NDXNUM          |
4  |&NDXNUM    |SETC      |'&SYSNDX'        |
   |&NAME      |SR        |2,4              |
   |           |AR        |2,6              |
5  |           |INNER1    |                 |
6  |B&SYSNDX   |S         |2,=F'1000'       |
   |           |MEND      |                 |
   |-----------|----------|-----------------|
7  |ALPHA      |OUTER1    |                 |
8  |BETA       |OUTER1    |                 |
   |-----------|----------|-----------------|
   |ALPHA      |SR        |2,4              |
   |           |AR        |2,6              |
   |A0107      |SR        |2,5              |
   |           |CR        |2,5              |
   |           |BE        |B0106            |
   |           |B         |A0107            |
   |B0106      |S         |2,=F'1000'       |
   |BETA       |SR        |2,4              |
   |           |AR        |2,6              |
   |A0109      |SR        |2,5              |
   |           |CR        |2,5              |
   |           |BE        |B0108            |
   |           |B         |A0109            |
   |B0108      |S         |2,=F'1000'       |
```

Statement 5 is the 107th macro-instruction processed. Therefore, &SYSNDX is assigned the number 0107 for that macro-instruction. The number 0107 is substituted for &SYSNDX when it is used in statements 1 and 3. The number 0106 is substituted for the global SETC symbol &NDXNUM in statement 2.

Statement 8 is the 108th macro-instruction processed. Therefore,each occurrence of &SYSNDX is replaced by the number 0108. For example, statement 6 is used to create the unique name B0108.

When statement 5 is used to process the 108th macro-instruction, statement 5 becomes the 109th macro-instruction processed. Therefore, each occurrence of &SYSNDX is replaced by the number 0109. For example, statement 1 is used to create the unique name A0109.

&SYSECT -- Current Control Section

The system variable symbol &SYSECT may be used to represent the name of the control section in which a macro-instruction appears. For each inner and outer macro-instruction processed by the assembler, &SYSECT is assigned a value that is the name of the control section in which the macro-instruction appears.

When &SYSECT is used in a macro-definition, the value substituted for &SYSECT is the name of the last CSECT, DSECT, or START statement that occurs before the macro-instruction. If no named CSECT, DSECT, or START statements occur before a macro-instruction, &SYSECT is assigned a null character value for that macro-instruction.

CSECT or DSECT statements processed in a macro-definition affect the value for &SYSECT for any subsequent inner macro-instructions in that definition, and for any other outer and inner macro-instructions.

Throughout the use of a macro-definition, the value of &SYSECT may be considered a constant, independent of any CSECT or DSECT statements or inner macro-instructions in that definition. &SYSECT will take on the name of the last CSECT, DSECT, or START statement regardless of whether or not that statement is correct.

The next example illustrates these rules.

Statement 8 is the last CSECT, DSECT, or START statement processed before statement 9 is processed. Therefore, &SYSECT is assigned the value MAINPROG for macro-instruction OUTER1 in statement 9. MAINPROG is substituted for &SYSECT when it appears in statement 6.

Statement 3 is the last CSECT, DSECT, or START statement processed before statement 4 is processed. Therefore, &SYSECT is assigned the value CSOUT1 for macro-instruction INNER in statement 4. CSOUT1 is substituted for &SYSECT when it appears in statement 2.

Statement 1 is used to generate a CSECT statement for statement 4. This is the last CSECT, DSECT, or START statement that appears before statement 5. Therefore, &SYSECT is assigned the value INA for macro instruction INNER in statement 5. INA is substituted for &SYSECT when it appears in statement 2.

	Name	Operation	Operand
		MACRO	
		INNER	&INCSECT
1	&INCSECT	CSECT	
2		DC	A(&SYSECT)
		MEND	
		MACRO	
		OUTER1	
3	CSOUT1	CSECT	
		DS	100C
4		INNER	INA
5		INNER	INB
6		DC	A(&SYSECT)
		MEND	
		MACRO	
		OUTER2	
7		DC	A(&SYSECT)
		MEND	
8	MAINPROG	CSECT	
		DS	200C
9		OUTER1	
10		OUTER2	
	MAINPROG	CSECT	
		DS	200C
	CSOUT1	CSECT	
		DS	100C
	INA	CSECT	
		DC	A(CSOUT1)
	INB	CSECT	
		DC	A(INA)
		DC	A(MAINPROG)
		DC	A(INB)

Statement 1 is used to generate a CSECT statement for statement 5. This is the last CSECT, DSECT, or START statement that appears before statement 10. Therefore, &SYSECT is assigned the value INB for macro instruction OUTER2 in statement 10. INB is substituted for &SYSECT when it appears in statement 7.

&SYSLIST - Accessing Positional Operands in a Macro Instruction

The system variable symbol &SYSLIST provides the programmer with an alternative to symbolic parameters for referring to positional macro instruction operands.

&SYSLIST may be coded, along with all other variable symbols (including symbolic parameters), in the model statements of any macro definition. (In the Tape Operating System (TOS), &SYSLIST cannot be used in macro definitions having any keyword symbolic parameters). When used to access a macro instruction operand, &SYSLIST is written with one or two subscripts:

1. &SYSLIST(m) will access the positional macro instruction operand corresponding to the positional operand subscript m. The programmer, therefore, does not have to define a positional parameter in the macro definition prototype statement. This allows him to access a different number of positional macro instruction operands in different calls to the same macro. The positional operand subscript m can be a self-defining term or an absolute expression, but its value must be a positive, whole number within the range of the number of operands permitted in a macro instruction.

   Note: A null string will be generated in place of &SYSLIST(m) if:

   a. m=0

   b. m is greater than the number of positional operands in the macro instruction.

   c. m accesses a specifically omitted operand.

   The model statement containing &SYSLIST(m) will be flagged in error if:

   a. m is negative.

   b. m is greater than 100 (for the D Assembler).

      m is greater than 200 (for the F Assembler).

2. &SYSLIST(m,n) accesses elements of positional operand sublists in macro instructions. The positional operand subsript m fulfills the same function as above, and is subjected to the same restrictions. The positional operand sublist subscript n refers to the sublist element of the positional operand in a macro instruction corresponding to m. Again positional parameters need not have been previously defined in the macro definition prototype statement.

   Note: A null string will be generated in place of &SYSLIST(m,n), m > 0 and otherwise within its allowable range, if:

a. n=0

b. n is greater than the number of elements in the positional operand sublist in the macro instruction.

c. n accesses a specifically omitted operand sublist element.

The type, length, scaling, integer, and count attributes of &SYSLIST(m) and &SYSLIST(m,n) and the number attributes of &SYSLIST(m) and &SYSLIST may be used in conditional assembly instructions. Attributes are discussed in Section 9 under Attributes.

N'&SYSLIST refers to the total number of positional operands in the macro instruction statement. When none have been called, N'&SYSLIST has the value 0. If, however, some positional operands in the macro instruction are specifically omitted (by means of commas), N'&SYSLIST will include the omitted operands in its count (see MAC2 and MAC3 in the examples below). A sublist is considered to be one operand (see MAC3 below):

Macro Instructions	N'&SYSLIST
MAC1 K1=DS	0
MAC2 ,,K1=DC	2
MAC3 FULL,,F,('1','2','3'),K1=DC	4

N'&SYSLIST(m) refers to the total number of elements in the macro instruction operand sublist corresponding to the positional operand subscript m. If the mth operand is omitted, N'&SYSLIST(m) is 0; if the mth operand is not a sublist, N'&SYSLIST(m) is 1.

In the MAC3 macro instruction above:

```
N'&SYSLIST(4) is 3
N'&SYSLIST(5) is 0
N'&SYSLIST(2) is 0
N'&SYSLIST(1) is 1
```

KEYWORD MACRO DEFINITIONS AND INSTRUCTIONS

Keyword macro definitions provide the programmer with an alternate way of preparing macro definitions.

A keyword macro definition enables a programmer to reduce the number of operands in each macro instruction that corresponds to the definition, and to write the operands in any order.

The macro instructions that correspond to the macro definitions described in Section 7 (hereinafter called positional macro instructions and positional macro definitions, respectively) require the operands to be written in the same order as the corresponding symbolic parameters in the operand entry of the prototype statement.

In a keyword macro definition, the programmer can assign values to any symbolic parameters that appear in the operand of the prototype statement. The value assigned to a symbolic parameter is substituted for the symbolic parameter, if the programmer does not write anything in the operand of the macro instruction to correspond to the symbolic parameter.

When a keyword macro instruction is written, the programmer need only write one operand for each symbolic parameter whose value he wants to change.

Keyword macro definitions are prepared the same way as positional macro definitions, except that the prototype statement is written differently, and &SYSLIST may not be used in the definition. The rules for preparing positional macro definitions are in Section 7.

Keyword Prototype

The typical form of this statement is:

Name	Operation	Operand
A symbolic parameter or not used	A symbol	One to 100 (200 for F assembler) operands of the form described below, separated by commas.

Each operand must consist of a symbolic parameter, immediately followed by an equal sign and optionally followed by a value. Nested keywords are not permitted.

A value that is part of an operand must immediately follow the equal sign.

Anything that may be used as an operand in a macro instruction except variable symbols, may be used as a value in a keyword prototype statement. The rules for forming valid macro instruction operands are detailed in Section 8.

The following are valid keyword prototype operands.

```
&READER=
&LOOP2=SYMBOL
&S4==F'4096'
```

The following are invalid keyword prototype operands.

```
CARDAREA        (no symbolic parameter)
&TYPE           (no equal sign)
&TWO =123       (equal sign does not
                  immediately follow
                  symbolic parameter)
&AREA= X'189A'  ( value does
                  not immediately follow
                  equal sign)
```

The following keyword prototype statement contains a symbolic parameter in the name entry and four operand entries in the operand. The first two operand entries contain values. The mnemonic operation code is MOVE.

Name	Operation	Operand
&N	MOVE	&R=2,&A=S,&T=,&F=

Keyword Macro Instruction

After a programmer has prepared a keyword macro definition he may use it by writing a keyword macro instruction.

The typical form of a keyword macro instruction is:

Name	Operation	Operand
A symbol, sequence symbol, or not used	Mnemonic operation code	Zero to 100 operands (200 for F assembler) of the form described below, separated by commas

Each operand consists of a keyword immediately followed by an equal sign and an optional value. Nested keywords are not permitted. Anything that may be used as an operand in a positional macro instruction may be used as a value in a keyword macro instruction. The rules for forming valid positional macro instruction operands are detailed in Section 8.

A keyword consists of one through seven letters and digits, the first of which must be a letter.

The keyword part of each keyword macro instruction operand must correspond to one of the symbolic parameters that appears in the operand of the keyword prototype statement. A keyword corresponds to a

symbolic parameter if the characters of the keyword are identical to the characters of the symbolic parameter that follow the ampersand.

The following are valid keyword macro instruction operands.

```
LOOP2=SYMBOL
S4==F'4096'
TO=
```

The following are invalid keyword macro instruction operands.

```
&X4F2=0(2,3)      (keyword does not begin
                   with a letter)
CARDAREA=A+2      (keyword is more than
                   seven characters)
=(TO(8),(FROM))   (no keyword)
```

The operands in a keyword macro instruction may be written in any order. If an operand appeared in a keyword prototype statement, a corresponding operand does not have to appear in the keyword macro instruction. If an operand is omitted, the comma that would have separated it from the next operand need not be written.

The following rules are used to replace the symbolic parameters in the statements of a keyword macro definition.

1. If a symbolic parameter appears in the name entry of the prototype statement, and the name entry of the macro instruction contains a symbol, the symbolic parameter is replaced by the symbol. If the name entry of the macro instruction is unused or contains a sequence symbol, the symbolic parameter is replaced by a null character value.

2. If a symbolic parameter appears in the operand of the prototype statement, and the macro instruction contains a keyword that corresponds to the symbolic parameter, the value assigned to the keyword replaces the symbolic parameter.

3. If a symbolic parameter was assigned a value by a prototype statement, and the macro instruction does not contain a keyword that corresponds to the symbolic parameter, the standard value assigned to the symbolic parameter replaces the symbolic parameter. Otherwise, the symbolic parameter is replaced by a null character value.

Note: If a symbolic parameter value is a self-defining term the type attribute assigned to the value is the letter N. If a symbolic parameter value is omitted the type attribute assigned to the value is the letter O. All other values are assigned the type attribute U.

The following keyword macro definition, keyword macro instruction, and generated statements illustrate these rules.

Statement 1 assigns the values 2 and S to the symbolic parameters &R and &A, respectively. Statement 6 assigns the values FA, FB, and THERE to the keywords T, F, and A, respectively. The symbol HERE is used in the name entry of statement 6.

Since a symbolic parameter (&N) appears in the name entry of the prototype statement (statement 1), and the corresponding characters (HERE) of the macro instruction (statement 6) are a symbol, &N is replaced by HERE in statement 2.

```
   |Name |Operation |Operand                  |
   |-----|----------|-------------------------|
   |     |MACRO     |                         |
1  |&N   |MOVE      |&R=2,&A=S,&T=,&F=        |
2  |&N   |ST        |&R,&A                    |
3  |     |L         |&R,&F                    |
4  |     |ST        |&R,&T                    |
5  |     |L         |&R,&A                    |
   |     |MEND      |                         |
   |-----|----------|-------------------------|
6  |HERE |MOVE      |T=FA,F=FB,A=THERE        |
   |-----|----------|-------------------------|
   |HERE |ST        |2,THERE                  |
   |     |L         |2,FB                     |
   |     |ST        |2,FA                     |
   |     |L         |2,THERE                  |
```

Since &T appears in the operand of statement 1, and statement 6 contains the keyword (T) that corresponds to &T, the value assigned to T (FA) replaces &T in statement 4. Similarly, FB and THERE replace &F and &A in statement 3 and in statements 2 and 5, respectively. Note that the value assigned to &A in statement 6 is used instead of the value assigned to &A in statement 1.

Since &R appears in the operand of statement 1, and statement 6 does not contain a corresponding keyword, the value assigned to &R (2), replaces &R in statements 2, 3, 4, and 5.

Operand Sublists: The value assigned to a keyword and the value assigned to a symbolic parameter may be an operand sublist. Anything that may be used as an operand sublist in a positional macro instruction may be used as a value in a keyword macro instruction and as a value in a keyword prototype statement. The rules for forming valid operand sublists are detailed in Section 8 under "Operand Sublists."

Keyword Inner Macro Instructions: Keyword and positional inner macro instructions may be used as model statements in either keyword or positional macro definitions.

MIXED-MODE MACRO DEFINITIONS AND INSTRUCTIONS

Mixed-mode macro definitions allow the programmer to use the features of keyword and positional macro definitions in the same macro definition.

Mixed-mode macro definitions are prepared the same way as positional macro definitions, except that the prototype statement is written differently. (In TOS &SYSLIST may not be used in the definition). The rules for preparing positional macro definitions are in Section 7.

Mixed-Mode Prototype

The typical form of this statement is:

Name	Operation	Operand
A symbolic parameter or not used	A symbol	Two to 100 (200 for F assembler) operands of the form described below, separated by commas

The operands must be valid operands of positional and keyword prototype statements. All the positional operands must precede the first keyword operand. The rules for forming positional operands are discussed in Section 7 under Macro Instruction Prototype. The rules for forming keyword operands are discussed under Keyword Prototype.

The following sample mixed-mode prototype statement contains three positional operands and two keyword operands.

Name	Operation	Operand
&N	MOVE	&TY,&P,&R,&TO=,&F=

Mixed-Mode Macro Instruction

The typical form of a mixed-mode macro instruction is:

Name	Operation	Operand
A symbol, sequence symbol, or not used	Mnemonic operation code	Zero to 100 (200 for F assembler) operands of the form described below, separated by commas

The operand consists of two parts. The first part corresponds to the positional prototype operands. This part of the operand is written in the same way that the operand entry of a positional macro instruction is written. The rules for writing positional macro instructions are in Section 8.

The second part of the operand corresponds to the keyword prototype operands. This part of the operand is written in the same way that the operand entry of a keyword macro instruction is written. The rules for writing keyword macro instructions are described under Keyword Macro Instruction.

The following mixed-mode macro definition, mixed-mode macro instruction, and generated statements illustrate these facilities.

	Name	Operation	Operand
		MACRO	
1	&N	MOVE	&TY,&P,&R,&TO=,&F=
	&N	ST&TY	&R,SAVE
		L&TY	&R,&P&F
		ST&TY	&R,&P&TO
		L&TY	&R,SAVE
2	HERE	MOVE	H,,2,F=FB,TO=FA
	HERE	STH	2,SAVE
		LH	2,FB
		STH	2,FA
		LH	2,SAVE

The prototype statement (statement 1) contains three positional operands (&TY,&P, and &R) and two keyword operands (&TO and &F). In the macro instruction (statement 2) the positional operands are written in the same order as the positional operands in the prototype statement (the second

operand is omitted). The keyword operands are written in an order that is different from the order of keyword operands in the prototype statement.

Mixed-mode inner macro-instructions may be used as model statements in mixed-mode, keyword, and positional macro-definitions. Keyword and positional inner macro-instructions may be used as model statements in mixed-mode macro-definitions.

CONDITIONAL ASSEMBLY COMPATIBILITY

Macro-definitions prepared for use with the other System/360 assemblers having macro language facilities may be used with the DOS/TOS Assembler provided that all SET symbols are declared in an appropriate LCLA, LCLB, LCLC, GBLA, GBLB, or GBLC statement. The AIFB and AGOB instructions are processed by the DOS/TOS Assembler the same way that the AIF and AGO instructions are processed. AIFB and AGOB instructions cause the count set up by the ACTR instruction to be decremented exactly like the AGO and AIF instructions.

APPENDIX A: EXTENDED BINARY CODED DECIMAL INTERCHANGE CODE (EBCDIC)

The following charts and the associated key show the bit configurations of the 256 possible codes (characters) of the Extended BCD Interchange Code. To write a given character in binary, locate the character on the chart. The top row of coordinates equates to bit positions 0 and 1, the second row to bit positions 2 and 3, and the left row of coordinates equates to bit positions 4, 5, 6 and 7.

Examples:

Character A equals:

top row - 11 (bit positions 0, 1)

2nd row - 00 (bit positions 2, 3)

left row - 0001 (bit positions 4, 5, 6 and 7)

Therefore, character A is shown as: 1100 0001.

Character $ equals:

top row - 01 (bit positions 0, 1)

2nd row - 01 (bit positions 2, 3)

left row - 1011 (bit positions 4, 5, 6 and 7)

Therefore, character $ is shown as: 0101 1011.

The coordinates on the bottom of the chart are the three zone punches required to reproduce the character in a punched card; the coordinates on the right side represent the numeric punches.

Examples:

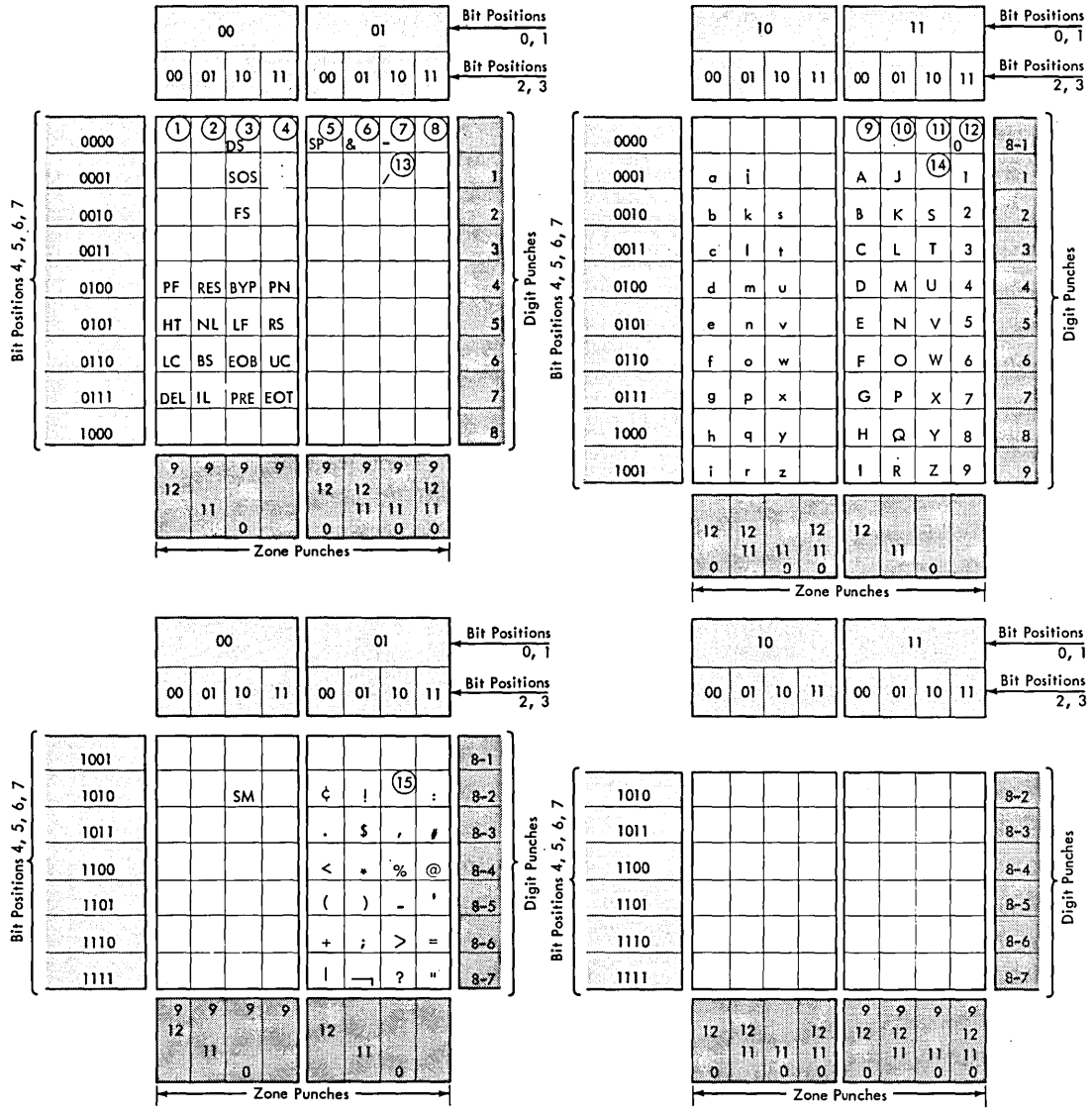
Character A = bottom row - 12 punch
right row - 1 punch

Therefore, Character A is shown by a 12 and a 1 punch in the same card column.

Character $ = bottom row - 11 punch
right row - 8 and 3 punches

Therefore, Character $ is shown by 11, 8, and 3 punches in the same card column.

There are fifteen exceptions to the punching equated to bit positions. These exceptions are shown in the chart by circled numbers 1 through 15, and the substituted punching is shown below the chart under Exceptions.



① 12-0-9-8-1
② 12-11-9-8-1
③ 11-0-9-8-1
④ 12-11-0-9-8-1
⑤ No Punches
⑥ 12
⑦ 11
⑧ 12-11-0
⑨ 12-0
⑩ 11-0
⑪ 0-8-2
⑫ 0
⑬ 0-1
⑭ 11-0-9-1
⑮ 12-11

Extended Binary Coded Decimal Interchange Code (Part 1 of 2)

Control Characters

PF	Punch Off	BS	Backspace	PN	Punch On
HT	Horizontal Tab	IL	Idle	RS	Reader Stop
LC	Lower Case	BY	Bypass	UC	Upper Case
DL	Delete	LF	Line Feed	ET	End of Transmission
RE	Restore	EB	End of Block	SM	Set Mode
NL	New Line	PR	Prefix	SP	Space
DS	Digit Select	SOS	Start of Significance	FS	Field Separator

Special Graphic Characters

¢	Cent Sign	*	Asterisk	>	Greater-than Sign
.	Period, Decimal Point	)	Right Parenthesis	?	Question Mark
<	Less-than Sign	;	Semicolon	:	Colon
(	Left Parenthesis	¬	Logical NOT	#	Number Sign
+	Plus Sign	-	Minus Sign, Hyphen	@	At Sign
|	Vertical Bar, Logical OR	/	Slash	'	Prime, Apostrophe
&	Ampersand	,	Comma	=	Equal Sign
!	Exclamation Point	%	Percent	"	Quotation Mark
$	Dollar Sign	_	Underscore		

Examples	Type	Bit Pattern Bit Positions 01 23 4567	Hole Pattern	
			Zone Punches	Digit Punches
PF	Control Character	00 00 0100	12 -9	4
%	Special Graphic	01 10 1100	0	8 - 4
R	Upper Case	11 01 1001	11	9
a	Lower Case	10 00 0001	12 -0	1
	Control Character, function not yet assigned	00 11 0000	12 - 11 - 0 -9	8 - 1

Extended Binary Coded Decimal Interchange Code (Part 2 of 2)

APPENDIX B: HEXADECIMAL-DECIMAL NUMBER CONVERSION TABLE

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

Hexadecimal	Decimal
000 to FFF	0000 to 4095

Decimal numbers (0000-4095) are given within the 5-part table. The first two characters (high-order) of hexadecimal numbers (000-FFF) are given in the lefthand column of the table; the third character (x) is arranged across the top of each part of the table.

To find the decimal equivalent of the hexadecimal number 0C9, look for 0C in the left column, and across that row under the column for x = 9. The decimal number is 0201.

To convert from decimal to hexadecimal, look up the decimal number within the table and read the hexadecimal number by a combination of the hex characters in the left column, and the value for x at the top of the column containing the decimal number.

For example, the decimal number 123 has the hexadecimal equivalent of 07B; the decimal number 1478 has the hexadecimal equivalent of 5C6.

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

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

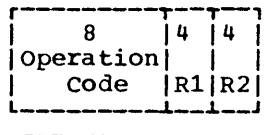
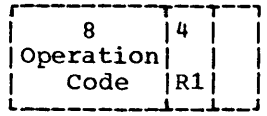
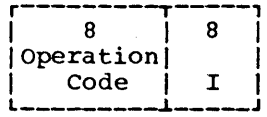
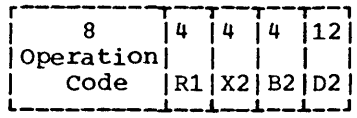
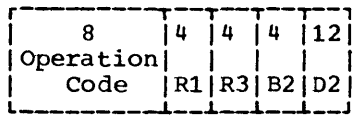
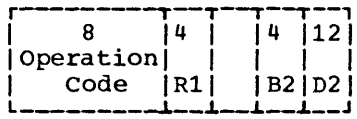
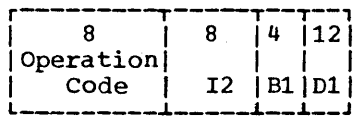
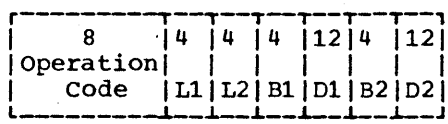
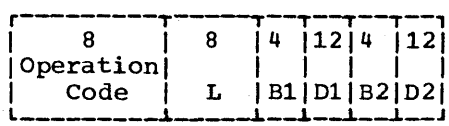
	x = 0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00x	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
01x	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
02x	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
03x	0048	0049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

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

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

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

APPENDIX C: MACHINE-INSTRUCTION FORMAT

	BASIC MACHINE FORMAT	ASSEMBLER OPERAND FIELD FORMAT	APPLICABLE INSTRUCTIONS
RR		R1,R2	All RR instructions except SPM and SVC
		R1	SPM
		I (See Notes 1, 6, 8, and 9)	SVC
RX		R1,D2(X2,B2) R1,D2(,B2) R1,S2(X2) (See notes 1-4, 7, and 9)	All RX instructions
RS		R1,R3,D2(B2) R1,R3,S2	BXH,BXLE,LM,STM
		R1,D2(B2) R1,S2 (See Notes 1-3,7, and 8)	All shift instructions
SI		D1(B1),I2 S1,I2	All SI instructions except LPSW,SSM, HIO,SIO,TIO,TCH,TS
		D1(B1) S1 (See Notes 2, 3, and 6-8)	LPSW,SSM,HIO,SIO, TIO,TCH,TS
SS		D1(L1,B1),D2(L2,B2) S1(L1),S2(L2)	PACK,UNPK,MVO,AP, CP,DP,MP,SP,ZAP
		D1(L,B1),D2(B2) S1(L),S2 (See Notes 2,3,5, and 7)	NC,OC,XC,CLC,MVC,MVN, MVZ,TR,TRT,ED,EDMK

Notes for Appendix C :

1. R1, R2, and R3 are absolute expressions that specify general or floating-point registers. The general register numbers are 0 through 15; floating-point register numbers are 0, 2, 4, and 6.

2. D1 and D2 are absolute expressions that specify displacements. A value of 0 - 4095 may be specified.

3. B1 and B2 are absolute expressions that specify base registers. Register numbers are 0 - 15.

4. X2 is an absolute expression that specifies an index register. Register numbers are 0 - 15.

5. L, L1, and L2 are absolute expressions that specify field lengths. An L expression can specify a value of 1 - 256. L1 and L2 expressions can specify a value of 1 - 16. In all cases, the assembled value will be one less than the specified value.

6. I and I2 are absolute expressions that provide immediate data. The value of the expression may be 0 - 255.

7. S1 and S2 are absolute or relocatable expressions that specify an address.

8. RR, RS, and SI instruction fields that are blank under BASIC MACHINE FORMAT are not examined during instruction execution. The fields are not written in the symbolic operand, but are assembled as binary zeros.

9. R1 specifies a 4-bit mask in the BC and BCR machine instructions.

APPENDIX D: MACHINE-INSTRUCTION MNEMONIC OPERATION CODES

Figure D-1 lists all machine operation codes and their associated assembler instructions and mnemonics in operation code order.

Figure D-2 contains the mnemonic operation codes for all machine instructions that can be represented in assembler language, including extended mnemonic operation codes. It is in alphabetic order by instruction. Indicated for each instruction are both the mnemonic and machine operation codes, explicit and implicit operand formats, program interruptions possible, and condition code set.

The column headings in this appendix and the information each column provides follow.

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

Mnemonic Operation Code: This column gives the mnemonic operation code for the machine instruction. This is written in the operation field when coding the instruction.

Machine Operation Code: This column contains the hexadecimal equivalent of the actual machine operation code. The operation code will appear in this form in most storage dumps and when displayed on the system control panel. For extended mnemonics, this column also contains the mnemonic code of the instruction from which the extended mnemonic is derived.

Operand Format: This column shows the symbolic format of the operand field in both explicit and implicit form. For both forms, R1, R2, and R3 indicate general registers in operands one, two, and three respectively. X2 indicates a general register used as an index register in the second operand. Instructions which require an index register (X2) but are not to be indexed are shown with a 0 replacing X2. L, L1, and L2 indicate lengths for either operand, operand one, and operand two respectively.

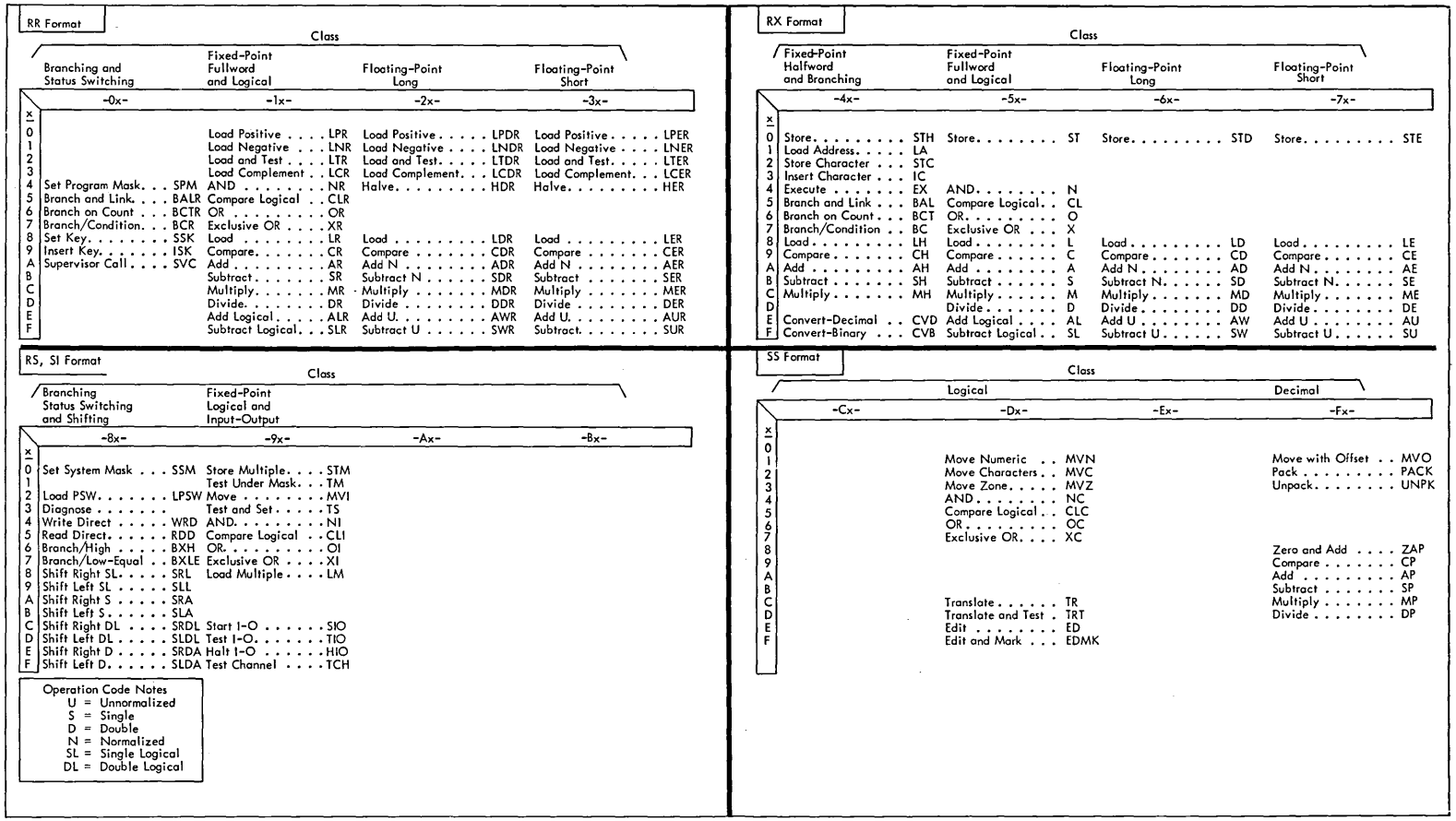
For the explicit format, D1 and D2 indicate a displacement and B1 and B2 indicate a base register for operands one and two.

For the implicit format, D1, B1, and D2, B2 are replaced by S1 and S2 which indicate a storage address in operands one and two.

Type of Instruction: This column gives the basic machine format of the instruction (RR, RX, SI, or SS). If an instruction is included in a special feature or is an extended mnemonic, this is also indicated.

Program Interruptions Possible: This column indicates the possible program interrupts for this instruction. The abbreviations used are: A - Addressing, S - Specification, Ov - Overflow, P - Protection, Op - Operation (if feature is not installed) and Other - other interruptions which are listed. The type of overflow is indicated by: D - Decimal, E - Exponent, or F - Fixed Point.

Condition Code Set: The condition set as a result of this instruction is indicated in this column. (See legend following the table.)



RR Format

x	Branching and Status Switching -0x-	Fixed-Point Fullword and Logical -1x-	Floating-Point Long -2x-	Floating-Point Short -3x-
0		Load Positive . . . . LPR	Load Positive . . . . . LPDR	Load Positive . . . . . LPER
1		Load Negative . . . LNR	Load Negative . . . . LNDR	Load Negative . . . . LNER
2		Load and Test . . . . LTR	Load and Test. . . . . LTDR	Load and Test. . . . . LTER
3		Load Complement . . LCR	Load Complement. . . LCDR	Load Complement. . . LCER
4	Set Program Mask. . . SPM	AND . . . . . NR	Halve. . . . . HDR	Halve. . . . . HER
5	Branch and Link. . . . BALR	Compare Logical . . CLR		
6	Branch on Count . . . BCTR	OR . . . . . OR		
7	Branch/Condition. . . BCR	Exclusive OR . . . . XR		
8	Set Key. . . . . SSK	Load . . . . . LR	Load . . . . . LDR	Load . . . . . LER
9	Insert Key. . . . . ISK	Compare. . . . . CR	Compare . . . . . CDR	Compare . . . . . CER
A	Supervisor Call . . . . SVC	Add . . . . . AR	Add N . . . . . ADR	Add N . . . . . AER
B		Subtract . . . . . SR	Subtract N . . . . . SDR	Subtract . . . . . SER
C		Multiply. . . . . MR	Multiply . . . . . MDR	Multiply . . . . . MER
D		Divide. . . . . DR	Divide . . . . . DDR	Divide . . . . . DER
E		Add Logical . . . . . ALR	Add U. . . . . AWR	Add U. . . . . AUR
F		Subtract Logical. . . SLR	Subtract U . . . . . SWR	Subtract. . . . . SUR

RX Format

x	Fixed-Point Halfword and Branching -4x-	Fixed-Point Fullword and Logical -5x-	Floating-Point Long -6x-	Floating-Point Short -7x-
0	Store. . . . . STH	Store. . . . . ST	Store. . . . . STD	Store. . . . . STE
1	Load Address. . . . . LA			
2	Store Character . . . STC			
3	Insert Character . . . IC			
4	Execute . . . . . EX	AND. . . . . N		
5	Branch and Link . . . BAL	Compare Logical. . CL		
6	Branch on Count . . . BCT	OR. . . . . O		
7	Branch/Condition . . BC	Exclusive OR . . . X		
8	Load . . . . . LH	Load L	Load . . . . . LD	Load . . . . . LE
9	Compare . . . . . CH	Compare C	Compare . . . . . CD	Compare . . . . . CE
A	Add . . . . . AH	Add . . . . . A	Add N . . . . . AD	Add N . . . . . AE
B	Subtract . . . . . SH	Subtract . . . . . S	Subtract N. . . . . SD	Subtract N. . . . . SE
C	Multiply . . . . . MH	Multiply . . . . . M	Multiply . . . . . MD	Multiply . . . . . ME
D		Divide . . . . . D	Divide . . . . . DD	Divide . . . . . DE
E	Convert-Decimal . . CVD	Add Logical . . . . AL	Add U . . . . . AW	Add U . . . . . AU
F	Convert-Binary . . . CVB	Subtract Logical . . SL	Subtract U. . . . . SW	Subtract U. . . . . SU

RS, SI Format

x	Branching Status Switching and Shifting -8x-	Fixed-Point Logical and Input-Output -9x-	-Ax-	-Bx-
0	Set System Mask . . . SSM	Store Multiple. . . . STM		
1		Test Under Mask. . . TM		
2	Load PSW. . . . . LPSW	Move . . . . . MVI		
3	Diagnose	Test and Set . . . . . TS		
4	Write Direct . . . . . WRD	AND. . . . . NI		
5	Read Direct. . . . . RDD	Compare Logical . . CLI		
6	Branch/High . . . . . BXH	OR. . . . . OI		
7	Branch/Low-Equal . . BXLE	Exclusive OR XI		
8	Shift Right SL. . . . . SRL	Load Multiple . . . . LM		
9	Shift Left SL . . . . . SLL			
A	Shift Right S . . . . . SRA			
B	Shift Left S. . . . . SLA			
C	Shift Right DL . . . . . SRDL	Start I-O . . . . . SIO		
D	Shift Left DL . . . . . SLDL	Test I-O. . . . . TIO		
E	Shift Right D . . . . . SRDA	Halt I-O . . . . . HIO		
F	Shift Left D. . . . . SLDA	Test Channel . . . . TCH		

Operation Code Notes
U = Unnormalized
S = Single
D = Double
N = Normalized
SL = Single Logical
DL = Double Logical

SS Format

x	Logical -Cx-	-Dx-	-Ex-	Decimal -Fx-
0				
1		Move Numeric . . MVN		Move with Offset . . MVO
2		Move Characters . . MVC		Pack . . . . . PACK
3		Move Zone. . . . . MVZ		Unpack. . . . . UNPK
4		AND. . . . . NC		
5		Compare Logical . . CLC		
6		OR . . . . . OC		
7		Exclusive OR. . . . XC		
8				Zero and Add . . . . ZAP
9				Compare . . . . . CP
A				Add . . . . . AP
B				Subtract . . . . . SP
C		Translate . . . . . TR		Multiply . . . . . MP
D		Translate and Test . TRT		Divide . . . . . DP
E		Edit . . . . . ED		
F		Edit and Mark . . . EDMK		

• Figure D-1. List of Machine Instructions by Operation Code

Instruction	Mnemonic Operation Code	Machine Operation Code	Operand Format	
			Explicit	Implicit
Add	A	5A	R1,D2(X2,B2) or R1, D2(,B2)	R1,S2(X2) or R1,S2
Add	AR	1A	R1,R2	
Add Decimal	AP	FA	D1(L1,B1),D2(L2,B2)	S1(L1),S2(L2)or S1,S2
Add Halfword	AH	4A	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2)or R1,S2
Add Logical	AL	5E	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2)or R1,S2
Add Logical	ALR	1E	R1,R2	
Add Normalized,Long	AD	6A	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2)or R1,S2
Add Normalized,Long	ADR	2A	R1,R2	
Add Normalized,Short	AE	7A	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2)or R1,S2
Add Normalized, Short	AER	3A	R1,R2	
Add Unnormalized,Long	AW	6E	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2)or R1,S2
Add Unnormalized,Long	AWR	2E	R1,R2	
Add Unnormalized,Short	AU	7E	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2)or R1,S2
Add Unnormalized,Short	AUR	3E	R1,R2	
And Logical	N	54	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2)or R1,S2
And Logical	NC	D4	D1(L,B1),D2(B2)	S1(L),S2 or S1,S2
And Logical	NR	14	R1,R2	
And Logical Immediate	NI	94	D1(B1),I2	S1,I2
Branch and Link	BAL	45	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2)or R1,S2
Branch and Link	BALR	05	R1,R2	
Branch on Condition	BC	47	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2)or R1,S2
Branch on Condition	BCR	07	R1,R2	
Branch on Count	BCT	46	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2)or R1,S2
Branch on Count	BCTR	06	R1,R2	
Branch on Equal	BE	47(BC 8)	D2(X2,B2)or D2(,B2)	S2(X2) or S2
Branch on High	BH	47(BC 2)	D2(X2,B2)or D2(,B2)	S2(X2) or S2
Branch on Index High	BXH	86	R1,R3,D2(B2)	R1,R3,S2
Branch on Index Low or Equal	BXLE	87	R1,R3,D2(B2)	R1,R3,S2
Branch on Low	BL	47(BC 4)	D2(X2,B2)or D2(,B2)	S2(X2) or S2
Branch if Mixed	BM	47(BC 4)	D2(X2,B2)or D2(,B2)	S2(X2) or S2
Branch on Minus	BM	47(BC 4)	D2(X2,B2)or D2(,B2)	S2(X2) or S2
Branch on Not Equal	BNE	47(BC 7)	D2(X2,B2)or D2(,B2)	S2(X2) or S2
Branch on Not High	BNH	47(BC 13)	D2(X2,B2)or D2(,B2)	S2(X2) or S2
Branch on Not Low	BNL	47(BC 11)	D2(X2,B2)or D2(,B2)	S2(X2) or S2
Branch on Not Minus	BNM	47(BC 11)	D2(X2,B2)or D2(,B2)	S2(X2) or S2
Branch on Not Ones	BNO	47(BC 14)	D2(X2,B2)or D2(,B2)	S2(X2) or S2
Branch on Not Plus	BNP	47(BC 13)	D2(X2,B2)or D2(,B2)	S2(X2) or S2
Branch on Not Zeros	BNZ	47(BC 7)	D2(X2,B2)or D2(,B2)	S2(X2) or S2
Branch if Ones	BO	47(BC 1)	D2(X2,B2)or D2(,B2)	S2(X2) or S2
Branch on Overflow	BO	47(BC 1)	D2(X2,B2)or D2(,B2)	S2(X2) or S2
Branch on Plus	BP	47(BC 2)	D2(X2,B2)or D2(,B2)	S2(X2) or S2
Branch if Zeros	BZ	47(BC 8)	D2(X2,B2)or D2(,B2)	S2(X2) or S2
Branch on Zero	BZ	47(BC 8)	D2(X2,B2)or D2(,B2)	S2(X2) or S2
Branch Unconditional	B	47(BC 15)	D2(X2,B2)or D2(,B2)	S2(X2) or S2
Branch Unconditional	BR	07(BCR 15)	R2	
Compare Algebraic	C	59	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2 or R1,S2
Compare Algebraic	CR	19	R1,R2	
Compare Decimal	CP	F9	D1(L1,B1),D2(L2,B2)	S1(L1),S2(L2)or S1,S2
Compare Halfword	CH	49	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2)or R1,S2
Compare Logical	CL	55	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2)or R1,S2
Compare Logical	CLC	D5	D1(L,B1),D2(B2)	S1(L),S2 or S1,S2
Compare Logical	CLR	15	R1,R2	
Compare Logical Immediate	CLI	95	D1(B1),I2	S1,I2
Compare,Long	CD	69	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2)or R1,S2
Compare,Long	CDR	29	R1,R2	
Compare,Short	CE	79	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2)or R1,S2
Compare,Short	CER	39	R1,R2	
Convert to Binary	CVB	4F	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2)or R1,S2
Convert to Decimal	CVD	4E	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2)or R1,S2

Figure D-2. Machine Instruction Summary (1 of 7)

Instruction	Type of Instruction	Program Interruption Possible						Condition Code Set			
		A	S	Ov	P	Op	Other	00	01	10	11
Add	RX	x	x	F				Sum=0	Sum<0	Sum>0	Overflow
Add	RR			F				Sum=0	Sum<0	Sum>0	Overflow
Add Decimal	SS, Decimal	x		D	x	x	Data	Sum=0	Sum<0	Sum>0	Overflow
Add Halfword	RX	x	x	F				Sum=0	Sum<0	Sum>0	Overflow
Add Logical	RX	x	x					Sum=0 Ⓗ	Sum 0 Ⓗ	Sum= 0 Ⓘ	Sum 0 Ⓘ
Add Logical	RR							Sum=0 Ⓗ	Sum= 0 Ⓗ	Sum= 0 Ⓘ	Sum 0 Ⓘ
Add Normalized, Long	RX, Floating Pt.	x	x	E		x	B, C	R	L	M	P
Add Normalized, Long	RR, Floating Pt.		x	E		x	B, C	R	L	M	P
Add Normalized, Short	RX, Floating Pt.	x	x	E		x	B, C	R	L	M	P
Add Normalized, Short	RR, Floating Pt.		x	E		x	B, C	R	L	M	P
Add Unnormalized, Long	RX, Floating Pt.	x	x	E		x	C	R	L	M	P
Add Unnormalized, Long	RR, Floating Pt.		x	E		x	C	R	L	M	P
Add Unnormalized, Short	RX, Floating Pt.	x	x	E		x	C	R	L	M	P
Add Unnormalized, Short	RR, Floating Pt.		x	E		x	C	R	L	M	P
Add Logical	RX	x	x					J	K		
And Logical	SS	x			x			J	K		
And Logical	RR							J	K		
And Logical Immediate	SI	x			x			J	K		
Branch and Link	RX							N	N	N	N
Branch and Link	RR							N	N	N	N
Branch on Condition	RX							N	N	N	N
Branch on Condition	RR							N	N	N	N
Branch on Count	RX							N	N	N	N
Branch on Count	RR							N	N	N	N
Branch on Equal	RX, Ext. Mnemonic							N	N	N	N
Branch on High	RX, Ext. Mnemonic							N	N	N	N
Branch on Index High	RX, Ext. Mnemonic							N	N	N	N
Branch on Index Low or Equal	RX, Ext. Mnemonic							N	N	N	N
Branch on Low	RX, Ext. Mnemonic							N	N	N	N
Branch if Mixed	RX, Ext. Mnemonic							N	N	N	N
Branch on Minus	RX, Ext. Mnemonic							N	N	N	N
Branch on Not Equal	RX, Ext. Mnemonic							N	N	N	N
Branch on Not High	RX, Ext. Mnemonic							N	N	N	N
Branch on Not Low	RX, Ext. Mnemonic							N	N	N	N
Branch on Not Minus	RX, Ext. Mnemonic							N	N	N	N
Branch on Not Ones	RX, Ext. Mnemonic							N	N	N	N
Branch on Not Plus	RX, Ext. Mnemonic							N	N	N	N
Branch on Not Zeros	RX, Ext. Mnemonic							N	N	N	N
Branch if Ones	RX, Ext. Mnemonic							N	N	N	N
Branch on Overflow	RX, Ext. Mnemonic							N	N	N	N
Branch on Plus	RX, Ext. Mnemonic							N	N	N	N
Branch if Zeros	RX, Ext. Mnemonic							N	N	N	N
Branch on Zero	RX, Ext. Mnemonic							N	N	N	N
Branch Unconditional	RX, Ext. Mnemonic							N	N	N	N
Branch Unconditional	RR, Ext. Mnemonic							N	N	N	N
Compare Algebraic	RX	x	x					Z	AA	BB	
Compare Algebraic	RR							Z	AA	BB	
Compare Decimal	SS, Decimal	x				x	Data	Z	AA	BB	
Compare Halfword	RX	x	x					Z	AA	BB	
Compare Logical	RX	x	x					Z	AA	BB	
Compare Logical	SS	x	x					Z	AA	BB	
Compare Logical	RR	x						Z	AA	BB	
Compare Logical Immediate	SI	x						Z	AA	BB	
Compare, Long	RX, Floating Pt.	x	x			x		Z	AA	BB	
Compare, Long	RR, Floating Pt.	x	x			x		Z	AA	BB	
Compare, Short	RX, Floating Pt.	x	x			x		Z	AA	BB	
Compare, Short	RR, Floating Pt.		x			x		Z	AA	BB	
Convert to Binary	RX	x	x				Data, F	N	N	N	N
Convert to Decimal	RX	x	x		x			N	N	N	N

Figure D-2. Machine Instruction Summary (2 of 7)

Instruction	Mnemonic Operation Code	Machine Operation Code	Operand Format	
			Explicit	Implicit
Divide	D	5D	R1,D2(X2,B2) or R1,D2(,B2)	R1, S2(X2) or R1,S2
Divide	DR	1D	R1,R2	
Divide Decimal	DP	FD	D1,(L1,B1),D2(L2,B2)	S1(L1),S2(L2)or S1,S2
Divide, Long	DD	6D	R1,D2(X2,B2),or R1,D2(,B2)	R1,S2(X2) or R1,S2
Divide, Long	DDR	2D	R1,R2	
Divide, Short	DE	7D	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2) or R1,S2
Divide, Short	DER	3D	R1,R2	
Edit	ED	DE	D1(L,B1),D2(B2)	S1(L),S2 or S1,S2
Edit and Mark	EDMK	DF	D1(L,B1),D2(B2)	S1(L),S2 or S1,S2
Exclusive Or	X	57	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Exclusive Or	XC	D7	D1(L,B1),D2(B2)	S1(L),S2 or S1,S2
Exclusive Or	XR	17	R1,R2	
Exclusive Or Immediate	XI	97	D1(B1),I2	S1,I2
Execute	EX	44	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Halve, Long	HDR	24	R1,R2	
Halve,Short	HER	34	R1,R2	
Halt I/O	HIO	9E	D1(B1)	
Insert Character	IC	43	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Insert Storage Key	ISK	09	R1,R2	
Load	L	58	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Load	LR	18	R1,R2	
Load Address	LA	41	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Load and Test	LTR	12	R1,R2	
Load and Test, Long	LTDR	22	R1,R2	
Load and Test, Short	LTER	32	R1,R2	
Load Complement	LCR	13	R1,R2	
Load Complement, Long	LCDR	23	R1,R2	
Load Complement, Short	LCER	33	R1,R2	
Load Halfword	LH	48	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Load, Long	LD	68	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Load, Long	LDR	28	R1,R2	
Load Multiple	LM	98	R1,R3,D2(B2)	R1,R3,S2
Load Negative	LNR	11	R1,R2	
Load Negative, Long	LNDR	21	R1,R2	
Load Negative, Short	LNER	31	R1,R2	
Load Positive	LPR	10	R1,R2	
Load Positive, Long	LPDR	20	R1,R2	
Load Positive, Short	LPER	30	R1,R2	
Load PSW	LPSW	82	D1(B1)	
Load, Short	LE	78	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Load, Short	LER	38	R1,R2	
Move Characters	MVC	D2	D1(L,B1),D2(B2)	S1(L),S2 or S1,S2
Move Immediate	MVI	92	D1(B1),I2	S1,I2
Move Numerics	MVN	D1	D1(L,B1),D2(B2)	S1(L),S2 or S1,S2
Move with Offset	MVO	F1	D1(L1,B1),D2(L2,B2)	S1(L1),S2(L2)or S1,S2
Move Zones	MVZ	D3	D1(L,B1),D2(B2)	S1(L),S2 or S1,S2
Multiply	M	5C	R1,D2(X2,B2)or R1,D2(,B2)	R1,S2(X2) or R1,S2
Multiply	MR	1C	R1,R2	
Multiply Decimal	MP	FC	D1(L1,B1),D2(L2,B2)	S1(L1),S2(L2) or S1,S2
Mulitply Halfword	MH	4C	R1,D2(X2.B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Multiply, Long	MD	6C	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Multiply, Long	MDR	2C	R1,R2	
Multiply, Short	ME	7C	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Multiply, Short	MER	3C	R1,R2	
No Operation	NOP	47(BC 0)	D2(X2,B2) or D2(,B2)	S2(X2) or S2

Figure D-2. Machine Instruction Summary (3 of 7)

Instruction	Type of Instruction	Program Interruptions Possible						Condition Code Set			
		A	S	Ov	P	Op	Other	00	01	10	11
Divide	RX	x	x				F	N	N	N	N
Divide	RR		x				F	N	N	N	N
Divide Decimal	SS, Decimal	x	x		x	x	D, Data	N	N	N	N
Divide, Long	RX, Floating Pt.	x	x	E		x	B, E	N	N	N	N
Divide, Long	RR, Floating Pt.		x	E		x	B, E	N	N	N	N
Divide, Short	RX, Floating Pt.	x	x	E		x	B, E	N	N	N	N
Divide, Short	RR, Floating Pt.		x	E		x	B, E	N	N	N	N
Edit	SS, Decimal	x			x	x	Data	S	T	U	
Edit and Mark	SS, Decimal	x			x	x	Data	S	T	U	
Exclusive Or	RX	x	x					J	K		
Exclusive Or	SS	x			x			J	K		
Exclusive Or	RR							J	K		
Exclusive Or Immediate	SI	x			x			J	K		
Execute	RX	x	x				G	(May be set by this instruction)			
Halve, Long	RR, Floating Pt.		x			x		N	N	N	N
Halve, Short	RR, Floating Pt.		x			x		N	N	N	N
Halt I/O	SI						A	DD	CC	GG	KK
Insert Character	RX	x						N	N	N	N
Insert Storage Key	RR	x	x			x	A	N	N	N	N
Load	RX	x	x					N	N	N	N
Load	RR							N	N	N	N
Load Address	RX							N	N	N	N
Load and Test	RR							J	L	M	
Load and Test, Long	RR, Floating Pt.		x			x		R	L	M	
Load and Test, Short	RR, Floating Pt.		x			x		R	L	M	
Load Complement	RR			F				P	L	M	O
Load Complement, Long	RR, Floating Pt.		x			x		R	L	M	
Load Complement, Short	RR, Floating Pt.		x			x		R	L	M	
Load Halfword	RX	x	x					N	N	N	N
Load, Long	RX, Floating Pt.	x	x			x		N	N	N	N
Load, Long	RR, Floating Pt.		x			x		N	N	N	N
Load Multiple	RS	x	x					N	N	N	N
Load Negative	RR							J	L		
Load Negative, Long	RR, Floating Pt.		x			x		R	L		
Load Negative, Short	RR, Floating Pt.		x			x		R	L		
Load Positive	RR			F				J		M	O
Load Positive, Long	RR, Floating Pt.		x			x		R	L	M	
Load Positive, Short	RR, Floating Pt.		x			x		R	L	M	
Load PSW	SI	x	x				A	QQ	QQ	QQ	QQ
Load, Short	RX, Floating Pt.	x	x			x		N	N	N	N
Load, Short	RR, Floating Pt.		x			x		N	N	N	N
Move Characters	SS	x			x			N	N	N	N
Move Immediate	SI	x			x			N	N	N	N
Move Numerics	SS	x			x			N	N	N	N
Move with Offset	SS	x			x			N	N	N	N
Move Zones	SS	x			x			N	N	N	N
Multiply	RX	x	x					N	N	N	N
Multiply	RR		x					N	N	N	N
Multiply Decimal	SS, Decimal	x	x		x	x	Data	N	N	N	N
Multiply Halfword	RX	x	x					N	N	N	N
Multiply, Long	RX, Floating Pt.	x	x	E		x	B	N	N	N	N
Multiply, Long	RR, Floating Pt.		x	E		x	B	N	N	N	N
Multiply, Short	RX, Floating Pt.	x	x	E		x	B	N	N	N	N
Multiply, Short	RR, Floating Pt.		x	E		x	B	N	N	N	N
No Operation	RX, Ext. Mnemonic							N	N	N	N

Figure D-2. Machine Instruction Summary (4 of 7)

Instruction	Mnemonic Operation Code	Machine Operation Code	Operand Format	
			Explicit	Implicit
No Operation	NOPR	07(BCR 0)	R2	
Or Logical	O	56	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Or Logical	OC	D6	D1(L,B1),D2(B2)	S1(L),S2 or S1,S2
Or Logical	OR	16	R1,R2	
Or Logical Immediate	OI	96	D1(B1),I2	S1,I2
Pack	PACK	F2	D1(L1,B1),D2(L2,B2)	S1(L1),S2(L2) or S1,S2
Read Direct	RDD	85	D1(B1),I2	S1,I2
Set Program Mask	SPM	04	R1	
Set System Key	SSK	08	R1,R2	
Set System Mask	SSM	80	D1(B1)	S1
Shift Left Double Algebraic	SLDA	8F	R1,D2(B2)	R1,S2
Shift Left Double Logical	SLDL	8D	R1,D2(B2)	R1,S2
Shift Left Single Algebraic	SLA	8B	R1,D2(B2)	R1,S2
Shift Left Single Logical	SLL	89	R1,D2(B2)	R1,S2
Shift Right Double Algebraic	SRDA	8E	R1,D2(B2)	R1,S2
Shift Right Double Logical	SRDL	8C	R1,D2(B2)	R1,S2
Shift Right Single Algebraic	SRA	8A	R1,D2(B2)	R1,S2
Shift Right Single Logical	SRL	88	R1,D2(B2)	R1,S2
Start I/O	SIO	9C	D1(B1)	S1
Store	ST	50	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Store Character	STC	42	R1,D2(X2,B2) or R1,D2(,B2	R1,D2(X2) or R1,S2
Store Halfword	STH	40	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Store Long	STD	60	R1,D2(X2,B2)	R1,S2(X2) or R1,S2
Store Multiple	STM	90	R1,R2,D2(B2)	R1,R2,S2
Store Short	STE	70	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Subtract	S	5B	R1,D2(X2	R1,S2(X2) or R1,S2
Subtract	SR	1B	R1,R2	
Subtract Decimal	SP	FB	D1(L1,B1),D2(L2,B2)	S1(L1),S2(L2) or S1,S2
Subtract Halfword	SH	4B	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Subtract Logical	SL	5F	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Subtract Logical	SLR	1F	R1,R2	
Subtract Normalized, Long	SD	6B	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Subtract Normalized, Long	SDR	2B	R1,R2	
Subtract Normalized, Short	SE	7B	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Subtract Normalized,	SER	3B	R1,R2	
Subtract Unnormalized, Long	SW	6F	R1,D2(X2,B2) or R1,D2(,B2)	R1,S2(X2) or R1,S2
Subtract Unnormalized, Long	SWR	2F	R1,R2	
Subtract Unnormalized, Short	SU	7F	R1,D2(X2,B2) or R1, D2(,B2)	R1,S2(X2) or R1,S2
Subtract Unnormalized, Short	SUR	3F	R1,R2	
Supervisor Call	SVC	0A	I	
Test and Set	TS	93	D1(B1)	S1
Test Channel	TCH	9F	D1(B1)	S1
Test I/O	TIO	9D	D1(B1)	S1
Test Under Mask	TM	91	D1(B1),I2	S1,I2
Translate	TR	DC	D1(L,B1),D2(B2)	S1(L),S2 orS1,S2
Translate and Test	TRT	DD	D1(L,B1),D2(B2)	S1(L),S2 orS1,S2
Unpack	UNPK	F3	D1(L1,B1),D2(L2,B2)	S1(L1),S2(L2)or S1,S2
Write Direct	WRD	84	D1(B1),I2	S1,I2
Zero and Add Decimal	ZAP	F8	D1(L1,B1),D2(L2,B2)	S1(L1),S2(L2)or S1,S2

Figure D-2. Machine Instruction Summary (5 of 7)

Instruction	Type of Instruction	Program Interruptions Possible						Condition Code Set			
		A	S	Ov	P	Op	Other	00	01	10	11
No Operation	RR, Ext. Mnemonic							N	N	N	N
Or Logical	RX	x	x					J	K		
Or Logical	SS	x			x			J	K		
Or Logical	RR							J	K		
Or Logical Immediate	SI	x			x			J	K		
Pack	SS	x			x			N	N	N	N
Read Direct	SI	x			x	x	A	N	N	N	N
Set Program Mask	RR							RR	RR	RR	RR
Set Storage Key	RR	x	x			x	A	N	N	N	N
Set System Mask	SI	x					A	N	N	N	N
Shift Left Double Algebraic	RS		x	F				J	L	M	O
Shift Left Double Logical	RS		x					N	N	N	N
Shift Left Single Algebraic	RS			F				J	L	M	O
Shift Left Single Logical	RS							N	N	N	N
Shift Right Double Algebraic	RS		x					J	L	M	
Shift Right Double Logical	RS		x					N	N	N	N
Shift Right Single Algebraic	RS							J	L	M	
Shift Right Single Logical	RS							N	N	N	N
Start I/O	SI						A	MM	CC	EE	AA
Store	RX	x	x		x			N	N	N	N
Store Character	RX	x			x			N	N	N	N
Store Halfword	RX	x	x		x			N	N	N	N
Store Long	RX, Floating Pt.	x	x		x	x		N	N	N	N
Store Multiple	RS	x	x		x			N	N	N	N
Store Short	RX, Floating Pt.	x	x		x	x		N	N	N	N
Subtract	RX	x	x	F				V	X	Y	O
Subtract	RR			F				V	X	Y	O
Subtract Decimal	SS, Decimal	x		D	x	x	Data	V	X	Y	O
Subtract Halfword	RX	x	x	F				V	X	Y	O
Subtract Logical	RX	x	x						W,H	V,I	W,I
Subtract Logical	RR								W,H	V,I	W,I
Subtract Normalized, Long	RX, Floating Pt.	x	x	E		x	B,C	R	L	M	Q
Subtract Normalized, Long	RR, Floating Pt.		x	E		x	B,C	R	L	M	Q
Subtract Normalized, Short	RX, Floating Pt.	x	x	E		x	B,C	R	L	M	Q
Subtract Normalized, Short	RR, Floating Pt.		x	E		x	B,C	R	L	M	Q
Subtract Unnormalized, Long	RX, Floating Pt.	x	x	E		x	C	R	L	M	Q
Subtract Unnormalized, Long	RR, Floating Pt.		x	E		x	C	R	L	M	Q
Subtract Unnormalized, Short	RX, Floating Pt.	x	x	E		x	C	R	L	M	Q
Subtract Unnormalized, Short	RR, Floating Pt.		x	E		x	C	R	L	M	Q
Supervisor Call	RR							N	N	N	N
Test and Set	SI	x			x			SS	TT		
Test Channel	SI						A	JJ	II	FF	HH
Test I/O	SI						A	LL	CC	EE	KK
Test Under Mask	SI	x						UU	VV		WW
Translate	SS	x			x			N	N	N	N
Translate and Test	SS	x						PP	NN	OO	
Unpack	SS	x			x			N	N	N	N
Write Direct	SI	x				x	A	N	N	N	N
Zero and Add Decimal	SS, Decimal	x		D	x	x	Data	J	L	M	O

Figure D-2. Machine Instruction Summary (6 of 7)

Program Interruptions Possible

Under Ov:
- D = Decimal
- E = Exponent
- F = Fixed Point

Under Other:
- A Privileged Operation
- B Exponent Underflow
- C Significance
- D Decimal Divide
- E Floating Point Divide
- F Fixed Point Divide
- G Execute

Condition Code Set

H	No Carry
I	Carry
J	Result = 0
K	Result is Not Equal to Zero
L	Result is Less Than Zero
M	Result is Greater Than Zero
N	Not Changed
O	Overflow
P	Result Exponent Underflows
Q	Result Exponent Overflows
R	Result Fraction = 0
S	Result Field Equals Zero
T	Result Field is Less Than Zero
U	Result Field is Greater Than Zero
V	Difference = 0
W	Difference is Not Equal to Zero
X	Difference is Less Than Zero
Y	Difference is Greater Than Zero
Z	First Operand Equals Second Operand
AA	First Operand is Less Than Second Operand
BB	First Operand is Greater Than Second Operand
CC	CSW Stored
DD	Channel and Subchannel not Working
EE	Channel or Subchannel Busy
FF	Channel Operating in Burst Mode
GG	Burst Operation Terminated
HH	Channel Not Operational
II	Interruption Pending in Channel
JJ	Channel Available
KK	Not Operational
LL	Available
MM	I/O Operation Initiated and Channel Proceeding With its Execution
NN	Nonzero Function Byte Found Before the First Operand Field is Exhausted
OO	Last Function Byte is Nonzero
PP	All Function Bytes Are Zero
QQ	Set According to Bits 34 and 35 of the New PSW Loaded
RR	Set According to Bits 2 and 3 of the Register Specified by R1
SS	Leftmost Bit of Byte Specified = 0
TT	Leftmost Bit of Byte Specified = 1
UU	Selected Bits Are All Zeros; Mask is All Zeros
VV	Selected Bits Are Mixed (zeros and ones)
WW	Selected Bits Are All Ones

Figure D-2. Machine Instruction Summary (7 of 7)

Operation Entry	Name Entry	Operand Entry
ACTR	Not used, must not be present	An arithmetic SETA expression
AGO	A sequence symbol or not present	A sequence symbol
AIF	A sequence symbol or not present	A logical expression enclosed in parentheses, immediately followed by a sequence symbol
ANOP	A sequence symbol	Not used, must not be present
CCW	Any symbol or not present	Four operands, separated by commas
CNOP	A sequence symbol or not present	Two absolute expressions, separated by a comma
COM	A sequence symbol or not present	Not used, should not be present
COPY	Not used, must not be present	A symbol
CSECT	Any symbol or not present	Not used, should not be present
DC	Any symbol or not present	One operand
DROP	A sequence symbol or not present	One to sixteen absolute expressions, separated by commas
DS	Any symbol or not present	One operand
DSECT	A variable symbol or an ordinary symbol	Not used, should not be present
EJECT	A sequence symbol or not present	Not used, should not be present
END	A sequence symbol or not present	A relocatable expression or not present
ENTRY	A sequence symbol or not present	One or more relocatable symbols, separated by commas
EQU	A variable symbol or an ordinary symbol	An absolute or relocatable expression
EXTRN	A sequence symbol or not present	One or more relocatable symbols, separated by commas
GBLA	Not used, must not be present	One or more variable symbols that are to be used as SET symbols, separated by commas[2]
GBLB	Not used, must not be present	One or more variable symbols that are to be used as SET symbols, separated by commas[2]
GBLC	Not used, must not be present	One or more variable symbols that are to be used as SET symbols, separated by commas[2]
ICTL	Not used, must not be present	One to three decimal values, separated by commas

Operation Entry	Name Entry	Operand Entry
ISEQ	Not used, must not be present	Two decimal values, separated by a comma
LCLA	Not used, must not be present	One or more variable symbols that are to be used as SET symbols, separated by commas[2]
LCLB	Not used, must not be present	One or more variable symbols that are to be used as SET symbols, separated by commas[2]
LCLC	Not used, must not be present	One or more variable symbols separated by commas[2]
LTORG	Any symbol or not present	Not used, should not be present
MACRO[1]	Not used, must not be present	Not used, should not be present
MEND[1]	A sequence symbol or not present	Not used, must not be present
MEXIT[1]	A sequence symbol or not present	Not used, must not be present
MNOTE[1]	A sequence symbol, a variable symbol or not present	A severity code, followed by a comma, followed by any combination of characters enclosed in apostrophes
ORG	A sequence symbol or not used	A relocatable expression or not used
PRINT	A sequence symbol or not present	One to three operands
PUNCH	A sequence symbol or not present	One to eighty characters enclosed in apostrophes
REPRO	A sequence symbol or not used	Not used, must not be present
SETA	A SETA symbol	An arithmetic expression
SETB	A SETB symbol	A 0 or a 1, or logical expression enclosed in parentheses
SETC	A SETC symbol	A type attribute, a character expression, a substring notation, or a concatenation of character expressions and substring notations
SPACE	A sequence symbol or not present	A decimal self-defining term or not used
START	Any symbol or not present	A self-defining term or not used
TITLE[3]	A special symbol (0 to 4 characters), a sequence symbol, a variable symbol, or not present	One to 100 characters, enclosed in apostrophes
USING	A sequence symbol or not present	An absolute or relocatable expression followed by 1 to 16 absolute expressions, separated by commas
WXTRN[4]	A sequence symbol or not present	One or more relocatable symbols, separated by commas

[1]May only be used as part of a macro definition
[2]SET symbols may be defined as subscripted SET symbols.
[3]See Section 5 for the description of the name entry.
[4]DOS Assembler 14K D only.

ASSEMBLER STATEMENTS

INSTRUCTION	NAME ENTRY	OPERAND ENTRY
Model Statements[3] [4] (A variable symbol or any assembler language mnemonic operation code except COPY, END, ICTL, ISEQ, and PRINT	An ordinary symbol, variable symbol, sequence symbol, a combination of variable symbols and other characters that is equivalent to a symbol, or not used	Any combination of characters (including variable symbols)
Prototype Statement[1]	A symbolic parameter or not used	Zero or more operands that are symbolic parameters, separated by commas, followed by zero or more operands (separated by commas) of the form symbolic parameter, equal sign, optional standard value
Macro Instruction Statement[1]	An ordinary symbol, a variable symbol, a sequence symbol, a combination of variable symbols and other characters that is equivalent to a symbol,[2] or not used	Zero or more positional operands separated by commas, followed by zero or more keyword operands (separated by commas) of the form keyword, equal sign, value[2]
Assembler Language Statement[3] [4]	An ordinary symbol, a variable symbol, a sequence symbol, a combination of variable symbols and other characters that is equivalent to a symbol, or not used	Any combination of characters (including variable symbols)

[1] May only be used as part of a macro definition.
[2] Variable symbols appearing in a macro instruction are replaced by their values before the macro instruction is processed.
[3] Variable symbols may not be used to generate the following mnemonic operation codes: ACTR, COPY, END, ICTL, CSECT, DSECT, ISEQ, PRINT, REPRO, and START. Variable symbols may not be used in the name and operand entries of the following instructions: COPY, END, ICTL, and ISEQ. Variable symbols may not be used in the name entry of the ACTR instruction.
[4] The line following a REPRO statement may not contain variable symbols.

APPENDIX F: SUMMARY OF CONSTANTS

TYPE AND DELIM.	IMPLIED LENGTH (BYTES)	ALIGN-MENT	LENGTH MODI-FIER RANGE	SPECIFIED BY	CON-STANTS PER OPERAND	RANGE FOR EX-PONENTS	RANGE FOR SCALE	TRUN-CATION/ PADDING SIDE
C "	as needed	byte	1 to 256 (1)	characters	one			right
X "	as needed	byte	1 to 256 (1)	hexadecimal digits	one			left
B "	as needed	byte	1 to 256	binary digits	one			left
F "	4	word	1 to 8	decimal digits	multiple	-85 to +75	-187 to +346	left
H "	2	half word	1 to 8	decimal digits	multiple	-85 to +75	-187 +346	left
E "	4	word	1 to 8	decimal digits	multiple	-85 to +75	0 to 14	right
D "	8	double word	1 to 8	decimal digits	multiple	-85 to +75	0 to 14	right
P "	as needed	byte	1 to 16	decimal digits	multiple			left
Z "	as needed	byte	1 to 16	decimal digits	multiple			left
A ()	4	word	1 to 4	an absolute expression	multiple			left
			3 or 4	a relocatable or complex relocatable expression				
V ()	4	word	3 or 4	relocatable symbol	multiple			left
S ()	2	half word	2 only	one absolute or relocatable expression or two absolute expressions: exp (exp)	multiple			
Y ()	2	half word	1 or 2	an absolute expression	multiple			left
			2 only	a relocatable or complex relocatable expression				

(1) In a DS assembler instruction, C and X type constants may have length specification to 65535.

APPENDIX G: MACRO FACILITY SUMMARY

The four charts in this appendix summarize the macro facility described in Part 2 of this publication.

Figure G-1 indicates which macro facility elements may be used in the name and operand entries of each statement.

Figure G-2 is a summary of the expressions that may be used in macro-instruction statements.

Figure G-3 is a summary of the attributes that may be used in each expression.

Figure G-4 is a summary of the variable symbols that may be used in each expression.

Statement	Variable Symbols										Attributes						Sequence Symbol
	Global SET Symbols				Local SET Symbols			System Variable Symbols									
	Symbolic Parameter	SETA	SETB	SETC	SETA	SETB	SETC	&SYSNDX	&SYSECT	&SYSLIST	Type	Length	Scaling	Integer	Count	Number	
MACRO																	
Prototype Statement	Name Operand																
GBLA		Operand															
GBLB			Operand														
GBLC				Operand													
LCLA					Operand												
LCLB						Operand											
LCLC							Operand										
Model Statement	Name Operation Operand	Name Operation Operand	Name Operation Operand	Name Operation Operand	Name Operation Operand	Name Operation Operand	Name Operation Operand	Name Operation Operand	Name Operation Operand	Name Operation Operand							Name
COPY																	Name
SETA	Operand[2]	Name Operand	Operand[3]	Operand[9]	Name Operand	Operand[3]	Operand[9]	Operand		Operand[2]		Operand	Operand	Operand	Operand	Operand	
SETB	Operand[6]	Operand[6]	Name Operand	Operand[6]	Operand[6]	Name Operand	Operand[6]	Operand[6]	Operand[4]	Operand[6]	Operand[4]	Operand[5]	Operand[5]	Operand[5]	Operand[5]	Operand[5]	
SETC	Operand	Operand[7]	Operand[8]	Name Operand	Operand[7]	Operand[8]	Name Operand	Operand	Operand	Operand	Operand						
AIF	Operand[6]	Operand[6]	Operand	Operand[6]	Operand[6]	Operand	Operand[6]	Operand[6]	Operand[4]	Operand[6]	Operand[4]	Operand[5]	Operand[5]	Operand[5]	Operand[5]	Operand[5]	Name Operand
AGO																	Name Operand
ACTR	Operand[2]	Operand	Operand[3]	Operand[2]	Operand	Operand[3]	Operand[2]	Operand		Operand[2]		Operand	Operand	Operand	Operand	Operand	
ANOP																	Name
MEXIT																	Name
MNOTE	Operand	Operand	Operand	Operand	Operand	Operand	Operand	Operand	Operand	Operand							Name
MEND																	Name
Outer Macro		Name Operand	Name Operand	Name Operand	Name Operand	Name Operand	Name Operand										Name
Inner Macro	Name Operand	Name Operand	Name Operand	Name Operand	Name Operand	Name Operand	Name Operand	Name Operand	Name Operand	Name Operand							Name
Assembler Language Statement		Name Operation Operand	Name Operation Operand	Name Operation Operand	Name Operation Operand	Name Operation Operand	Name Operation Operand										Name

1. Variable symbols in macro-instructions are replaced by their values before processing.
2. Only if value is self-defining term.
3. Converted to arithmetic +1 or +0.
4. Only in character relations.
5. Only in arithmetic relations.
6. Only in arithmetic or character relations.
7. Converted to unsigned number.
8. Converted to character 1 or 0.
9. Only if one to eight decimal digits.

Figure G-1. Macro Facility Elements

Expression	Arithmetic Expressions	Character Expressions	Logical Expressions
May contain	1. Self-defining terms 2. Length, scaling, integer, count, and number attributes 3. SETA and SETB symbols 4. SETC symbols whose value is 1-8 decimal digits 5. Symbolic parameters if the corresponding operand is a self-defining term 6. &SYSLIST(n) if the corresponding operand is a self-defining term 7. &SYSLIST(n,m) if the corresponding operand is a self-defining term 8. &SYSNDX	1. Any combination of characters enclosed in apostrophes 2. Any variable symbol enclosed in apostrophes 3. A concatenation of variable symbols and other characters enclosed in apostrophes 4. A request for a type attribute.	1. SETB symbols 2. Arithmetic relations[1] 3. Character relations[2]
Operators are	+,-,*, and / parentheses permitted	concatenation , with a period (.)	AND, OR, and NOT parentheses permitted
Range of values	-2^{31} to $+2^{31}-1$	0 through 127 (255 for assembler F) characters.	0 (false) or 1 (true)
May be used in	1. SETA operands 2. Arithmetic relations 3. Subscripted SET symbols 4. &SYSLIST 5. Substring notation 6. Sublist notation 7. SETC operands 8. ACTR operands	1. SETC operands[3] 2. Character relations[2] 3. SETA operands[4]	1. SETB operands 2. AIF operands

[1] An arithmetic relation consists of two arithmetic expressions related by the operators GT, LT, EQ, NE, GE, or LE.

[2] A character relation consists of two character expressions related by the operator GT, LT, EQ, NE, GE, or LE. The type attribute notation and the substring notation may also be used in character relations. The maximum length of the character expressions that can be compared is 127 (255 for assembler F) characters. If the two character expressions are of unequal length, then the shorter one will always compare less than the longer.

[3] Maximum of eight characters will be assigned.

[4] If one to eight decimal digits.

Figure G-2. Expressions

Attribute	Notation	May be used with:	May be used only if type attribute is:	May be used in
Type	T'	Symbols outside macro-definitions; symbolic parameters, &SYSLIST(n), and &SYSLIST(n,m) inside macro-definitions	(May always be used)	1. SETC operand fields 2. Character relations (SETB)
Length	L'	Symbols outside macro-definitions; symbolic parameters, &SYSLIST(n), and &SYSLIST(n,m) inside macro-definitions	Any letter except M,N,O,T, and U	Arithmetic expressions
Scaling	S'	Symbols outside macro-definitions; symbolic parameters, &SYSLIST(n), and &SYSLIST(n,m) inside macro-definitions	H,F,G,D,E,K,P, and Z	Arithmetic expressions
Integer	I'	Symbols outside macro-definitions; symbolic parameters, &SYSLIST(n), and &SYSLIST(n,m) inside macro-definitions	H,F,G,D,E,K,P, and Z	Arithmetic expressions
Count	K'	Symbolic parameters corresponding to macro-instruction operands, &SYSLIST (n), and &SYSLIST(n,m) inside macro-definitions	Any letter	Arithmetic expressions
Number	N'	Symbolic parameters, &SYSLIST, and &SYSLIST(n) inside macro-definitions	Any letter	Arithmetic expressions

Figure G-3. Attributes

Variable symbol	Defined by:	Initialized, or set to:	Value changed by:	May be used in:
Symbolic[1] parameter	Prototype statement	Corresponding macro-instruction operand	(Constant throughout definition)	1. Arithmetic expressions if operand is self-defining term 2. Character expressions
SETA	LCLA or GBLA instruction	0	SETA instruction	1. Arithmetic expressions 2. Character expressions
SETB	LCLB or GBLB instruction	0	SETB instruction	1. Arithmetic expressions 2. Character expressions 3. Logical expressions
SETC	LCLC or GBLC instruction	Null character value	SETC instruction	1. Arithmetic expressions if value is one to eight decimal digits 2. Character expressions
&SYSNDX[1]	The assembler	Macro-instruction index	(Constant throughout definition; unique for each macro-instruction)	1. Arithmetic expressions 2. Character expressions
&SYSECT[1]	The assembler	Control section in which macro-instruction appears	(Constant throughout definition; set by CSECT, DSECT, and START)	Character expressions
&SYSLIST[1]	The assembler	Not applicable	Not applicable	N'&SYSLIST in arithmetic expressions
&SYSLIST(n)[1] &SYSLIST(n,m)[1]	The assembler	Corresponding macro-instruction operand	(Constant throughout definition)	1. Arithmetic expressions if operand is self-defining term 2. Character expressions

[1] May only be used in macro-definitions.

Figure G-4. Variable Symbols

APPENDIX H: DICTIONARY AND SOURCE STATEMENT SIZES

PART 1: DICTIONARIES USED IN MACRO GENERATION

A. Dictionaries at Collection Time

Two or more dictionaries must be constructed to enable the macro generator portion of the assembler to accomplish macro generation and conditional assembly: a global dictionary and one or more local dictionaries.

Global Dictionary

A global dictionary containing macro instruction mnemonics and global SET variable names is built for the entire program. Dictionary entries are fitted into blocks of fixed size, 256 bytes for Assembler (D) and 1024 bytes for Assembler (F).

Each block contains complete entries. If an entry cannot fit into the remainder of one block, it is put into the next block and bytes in the remainder are not used. The sizes of various kinds of dictionary entries are as follows:

Macro Mnemonic Operation Code	10 bytes plus mnemonic*
Global SET Variable Name	6 bytes plus name* (A dimensioned global SET variable is counted only once)
Fixed Overhead	8 bytes for first block 4 bytes for each succeeding block 5 bytes for last block

The maximum size of the global dictionary depends on the size of the partition and can be no greater than 64 blocks. Maximum dictionary sizes for Assembler (D) variants in the smallest possible partitions are as follows (in bytes):

DOS Assembler (D) variant \ Partition size	10K	12K	14K	16K
10K with tape work files	512	1024	1536	2048
10K with disk work files	512	1024	1536	1536
14K	-	-	1024	1536

An additional limitation on dictionary size is that the maximum number of distinct global symbols is 400.

* One byte is used for each character in the name or mnemonic.

Local Dictionary

A local dictionary containing ordinary symbols relevant to macro generation and conditional assembly, sequence symbols and local SET variable names is constructed for the main portion of the program. In addition, a local dictionary containing an entry for each local SET variable name, sequence symbol and prototype symbolic parameter declared within a macro definition is constructed for each different macro definition used in the program. Dictionary entries are fitted into blocks of fixed size, 256 bytes for Assembler (D) and 1024 bytes for Assembler (F).

Each block contains complete entries. If an entry cannot fit into the remainder of one block, it is put into the next block and bytes in the remainder are not used. The sizes of various kinds of dictionary entries are as follows:

Sequence Symbol Names	10 bytes plus name * (A reference to sequence symbols after definition, a backwards branch, causes an additional entry to be made in the local dictionary.)
Local SET Variable Names	6 bytes plus name * (A dimensioned local SET variable is counted only once.)
Prototype Symbolic Parameters	5 bytes plus name *
Relevant ordinary symbols appearing in the main portion of the program	10 bytes plus name *
Fixed Overhead	8 bytes for first block (32 bytes if a macro local dictionary) 4 bytes for each succeeding block 5 bytes for last block

* One byte is used for each character in the name or mnemonic.

The maximum size of a local dictionary depends on the size of the partition and can be no greater than 64 blocks. Maximum dictionary sizes for DOS Assembler (D) variants in the smallest possible partitions are as follows (in bytes):

Partition size / DOS Assembler (D) variant	10K	12K	14K	16K
10K with tape work files	1536	2048	2816	3072
10K with disk work files	1024	1536	2304	3072
14K	-	-	2048	2304

B. Dictionaries at Generation Time

To conserve storage during the actual conditional assembly and macro generation, the contents of the Global Dictionary and Local Dictionaries are restructured as follows:

Global Dictionary

Fixed Overhead	4 bytes plus word alignment
Macro Mnemonic Operation Code	3 bytes
Global SETA dimensioned	1 byte plus 4N
Global SETA undimensioned	4 bytes
Global SETB dimensioned	1 byte plus (N/8) [N/8 is rounded to the next highest integer]
Global SETB undimensioned	1 byte
Global SETC dimensioned	1 byte plus 9N
Global SETC undimensioned	9 bytes

Local Dictionary

Fixed Overhead	20 bytes plus word alignment for the F assembler 27 bytes plus word alignment for the D assembler
Sequence Symbols	5 bytes (when the size of the dictionary (see below) is less than 3000, only the first 40 symbols will require 5 bytes each)
Local SETA dimensioned	1 byte plus 4N
Local SETA undimensioned	4 bytes
Local SETB dimensioned	1 byte plus (N/8) [N/8 is rounded to the next highest integer]
Local SETB undimensioned	1 byte
Local SETC dimensioned	1 byte plus 9N
Local SETC undimensioned	9 bytes
Relevant ordinary symbols appearing in the main portion of the program (see Note)	5 bytes

N = dimension

Note: For the D assembler, only those ordinary symbols which appear in macro instruction operands are included in this table; for the F assembler, all ordinary symbols are included. As a result, the F assembler may overflow the Local Dictionary before the D assembler.

The restructured Global Dictionary and the restructured Local Dictionary for the main portion of the program must be resident in main storage.

In addition, if the program contains any macro instructions, main storage is required for the largest Local Dictionary of the macro definitions being processed. Furthermore, if any macro definitions contain inner macro instructions, main storage is required for all the restructured Local Dictionaries of all the macros in the nest.

In addition to those requirements specified above for the Local Dictionary of the main portion of the program, each macro definition Local Dictionary requires the following for the parameter table:

1. Fixed Overhead — 22 bytes

2. Table Entries
   a. Character string — 3 bytes plus L
   b. Hexadecimal, binary, decimal, and character self-defining values — 7 bytes plus L
   c. Symbol — 9 bytes plus L
   d. Sublist — 10 bytes plus 2N bytes plus Y

   L=Length of entry
   N=Number of entries in sublist
   Y=Total length of table entries of a., b., and c. formats

Each nested macro instruction also requires the following:
   Parameter pointer list — 2 bytes plus 2N (N = the number of operands)
   Pointers to list in table — 8 bytes plus word alignment

The size of the dictionary depends on the partition size and the assembler variant used. Maximum dictionary sizes for Assembler (D) variants in the smallest possible partitions are as follows (in bytes):

Partition size / DOS Assembler (D) variant	10K	12K	14K	16K
10K with tape work files	2050	3100	4150	5000
10K with disk work files	1500	2400	3600	4400
14K	-	-	2750	3800

PART 2: MACRO MNEMONIC TABLE (D ASSEMBLER ONLY)

As the source text is scanned, a table of macro mnemonics is constructed. There is an entry for each macro used or defined as a programmer macro in the program. The entries are made under the premise that every undefined operation is a system macro mnemonic. This table is then subsetted to locate and edit system macros from the library.

An entry in this subsetted table consists of 9 bytes. With 10,240 or 14,336 contiguous bytes of main storage available (see Machine Features Required), approximately 450 distinct macro mnemonics can be handled. When this table overflows, processing continues with only those macros defined at that point. If additional storage is available, this table is expanded accordingly.

PART 3: SOURCE STATEMENT COMPLEXITY - CONDITIONAL ASSEMBLY AND MACRO GENERATION

For any statement except macro prototype or macro instructions, a counter is increased by one for each literal occurrence of the following:

1. Ordinary Symbol

   a. Name, operation, or operand entry (when the operand count starts, the counter is decremented by one), or

   b. Operand of an EXTRN or WXTRN statement, or

   c. Operand of an attribute operator (L',T',I', etc. in a SETA, SETB or SETC expression, or

   d. Operand of a machine or assembler instruction (only if in the main portion of the program)

2. Variable Symbol

3. Sequence Symbol

Note 1: The maximum value the counter may attain is 35 for the D assembler and 50 for the F assembler.

Note 2: This restriction applies to the name and operation entry of a macro instruction or prototype taken as a unit. Each macro instruction or prototype operand (in sublist, each sublist operand) is also subject to the counter restriction.

Examples of counts:

```
1.  &B2 SETB (T'NAME EQ'W' OR '&C'.'A' EQ'AA')
        count=3
2.      EXTRN  A, B, C, &C
        count=4
```

PART 4: SOURCE STATEMENT COMPLEXITY - ASSEMBLER STATEMENTS

A. D Assembler

With 10,240 or 14,336 contiguous bytes of main storage available (see Machine Features Required), the size of any statement must be less than a certain limit. This limit is:

1. 727 bytes for DC or DS statements.
2. 743 bytes for all other statements.

There are two formulas used to estimate the size (in bytes) of a statement. The greater of the two calculated values (S_1 or S_2) determines whether the statement is less than the given limit. In general, all statements can be processed if they contain 50 or fewer terms. If a statement contains more than 50 terms, the formulas should be used to determine if the statement can be processed, or if the statement should be shortened using EQU assembler instructions. (In the example for S_1, if A+(B-C)*3 were equated to a symbol, that symbol could be used as the displacement field of the first operand.) The formulas for statement size, S_1 and S_2, follow.

$$S_1 = N_B+N_D+4(N_{LS}+N_{SD})+6(N_S+N_L)$$

N_B = the total number of bytes in name, operation, operand, and comments entries. (The maximum value of N_B is 187.)

N_D = the number of operators and delimiters in the operand entry [except equal (=), period (.), and apostrophe (')]

N_{LS}= the number of references to length attribute (L'SYMBOL),

N_{SD} = the number of self-defining terms,

N_S = the number of symbolic terms (including *),

N_L = the number of literal operands. (The maximum is 1.)

Example:

```
NAME MVC A+(B-C)*3(L'D,5),=15CL5'ABCDEFG'
        S1=39+9+4(1+4)+6(3+1)
          =92 bytes
```

$$S_2 = N_B + 9(W_1 + W_2 + \ldots\ldots + W_i + N_E) + N_{ED}$$

N_B = the total number of bytes in name, operation, operand, and comments entries. (The maximum value of N_B is 187).

$W_1+W_2\ldots\ldots+W_i$ = a weight associated with the 1st, 2nd, ..., i^{th} expression.

W_i = 1, if the expression is:
- a. absolute,
- b. simply relocatable, or
- c. in error.

If the expression is complexly relocatable, W_i depends on the number of unpaired control section numbers (N_{ESD}).

N_{ESD}	W_i
1	1
2, 3, 4, or 5	2
6, 7, 8, or 9	3
10, 11, 12, or 13	4
14, 15, or 16	5

N_E = the number of expressions.

N_{ED} = the number of expression delimiters.

The rules for counting the number of expressions (N_E) and the number of expression delimiters (N_{ED}) are:

1. Expression delimiters are commas and the terminating blank of an operand.

2. Left and right parentheses can be part of an expression or can be expression delimiters. A left or right parenthesis is an expression delimiter if it ends an expression. Otherwise, it is part of an expression.

Example 1: The operand is:

```
5,6,A+20*B(6,7)
```

The expression delimiters are the three commas, the left parenthesis [(], the right parenthesis [)], and the terminating blank.

The first, second, fourth, and fifth expressions all have a weight of 1. The third expression in the operand [A+20*B] has a weight of 1 (either B is absolute, making the result absolute or simply relocatable or, B is relocatable so the expression is in error.

$$S_2 = N_B + 9(W_1 + W_2 + W_3 + W_4 + W_5 + N_E) + N_{ED}$$

$$S_2 = N_B + 9(1 + 1 + 1 + 1 + 1 + 5) + 6$$

$$S_2 = N_B + 96 \text{ bytes}$$

Example 2: The operand is:

```
A+17*(C-D),(A+20)
```

The number of expressions (N_E) is 2. The first expression is A+17*(C-D). The second expression is (A+20).

The number of expression delimiters (N_{ED}) is 2 (the comma and the terminating blank).

Example 3: The operand is:

```
20(5,3),16(5)
```

There are 5 expressions and 7 expression delimiters.

```
Expression 1 = 20     Expression Delimiter 1 = (
Expression 2 =  5     Expression Delimiter 2 = ,
Expression 3 =  3     Expression Delimiter 3 = )
Expression 4 = 16     Expression Delimiter 4 = ,
Expression 5 =  5     Expression Delimiter 5 = (
                      Expression Delimiter 6 = )
                      Expression Delimiter 7 = blank
```

B. F Assembler

1. Generated statements may not exceed 272 characters. Statement length includes name, operation, operand, and comments. If a comments field exists, the blank separating the operand and the comments field is included in the statement length. The statement is truncated if it exceeds 272 characters.

2. DC, DS and literal DCs cannot contain more than 32 operands per statement.

PART 5: PRINT CONTROL STATEMENT LISTING RESTRICTIONS

TITLE, SPACE and EJECT statements will not appear in the source listings unless the statement is continued onto another card. Then the first card of the statement will be listed. If any of these three statements are generated by macro expansion, they will not be listed (regardless of continuation) if the current PRINT option is NOGEN.

APPENDIX I: SAMPLE PROGRAM AND ASSEMBLER LISTING DESCRIPTION

The assembler listing consists of five sections, ordered as follows: external symbol dictionary items; the source and object program statements; relocation dictionary items; symbol cross-reference table; and diagnostic messages.

The following sample program illustrates an actual assembler listing. Several errors have been included to show their effect on an assembly.

Given:

1. A TABLE with 15 entries, each 16 bytes long, having the following format:

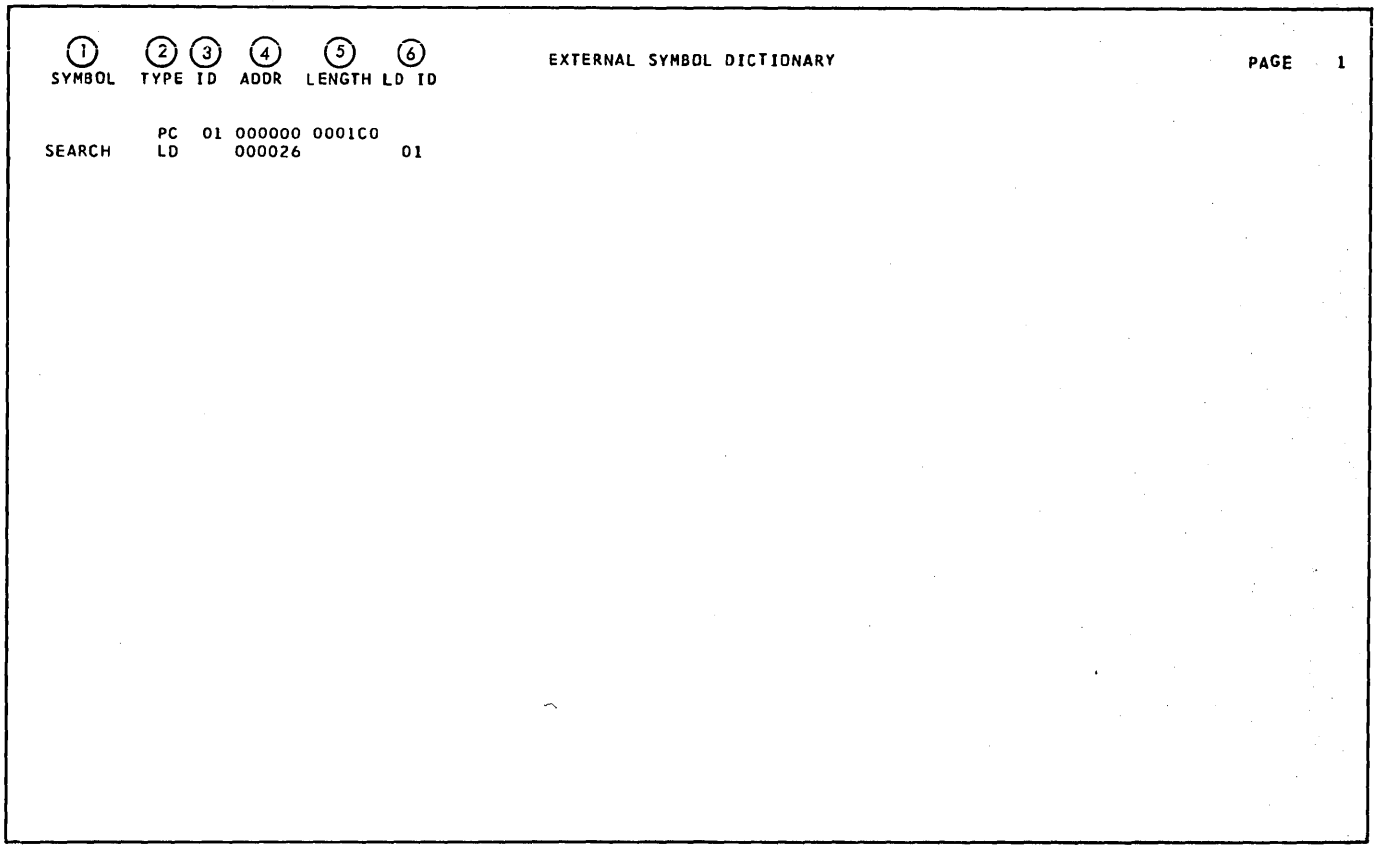
NUMBER of items	SWITCHes	ADDRESS	NAME
3 bytes	1 byte	4 bytes	8 bytes

2. A LIST of items, each 16 bytes long, having the following format:

NAME	SWITCHes	NUMBER of items	ADDRESS
8 bytes	1 byte	3 bytes	4 bytes

Find: Any of the items in the LIST which occur in the TABLE and put the SWITCHes, NUMBER of items, and ADDRESS from that LIST entry into the corresponding TABLE entry. If the LIST item does not occur in the TABLE, turn on the first bit in the SWITCHes byte of the LIST entry.

The TABLE entries have been sorted by their NAME.



EXTERNAL SYMBOL DICTIONARY (ESD)

This section of the listing contains the external symbol dictionary information passed to the linkage-editor in the object module. The entries describe the control sections, external references, and entry points in the assembled program. There are five types of entries, shown along with with their associated fields. The circled numbers refer to the corresponding heading in the sample listing.

(1) SYMBOL	(2) TYPE	(3) ID	(4) ADDR	(5) LENGTH	(6) LDID
X	SD	X	X	X	-
X	LD	-	X	-	X
X	ER	X	-	-	-
-	PC	X	X	X	-
-	CM	X	X	X	-
X	WX	X	-	-	-

The X indicates entries accompanying each type designation.

(1) This column contains symbols that appear in the name field of CSECT or START statements, as operands of ENTRY, EXTRN, and WXTRN statements, or in the operand field of V-type address constants.

(2) This column contains the type designator for the entry, as shown in the table. The type designators are defined as:

- SD -- names section definition. The symbol appeared in the name field of a CSECT or START statement.
- LD -- The symbol appeared as the operand of an ENTRY statement.
- ER -- external reference. The symbol appeared as the operand of an EXTRN statement, or was defined as a V-type address constant.
- PC -- unnamed control section definition.
- CM -- common control section definition.
- WX -- weak external reference. The symbol appeared as the operand of a WXTRN statement.

(3) This column contains the external symbol dictionary identification number (ID). The number is a unique two digit hexadecimal number identifying the entry.

```
(7)                (8)                                                                    (9)
EXAM          SAMPLE PROGRAM                                                          PAGE    1
(10)  (11)              (12)    (13)      (14)                                    (15)      (16)
LOC   OBJECT CODE     ADDR1 ADDR2 STMT   SOURCE STATEMENT                        DOS CL3-0 09/16/67
                                                                                          (17)
                                    2 ********************************************************************** SAMPL001
                                    3 *        THIS IS THE MACRO DEFINITION                                * SAMPL002
                                    4 ********************************************************************** SAMPL003
                                    5          MACRO                                                       SAMPL004
                                    6          MOVE  &TO,&FROM                                             SAMPL005
                                    7 .*                                                                   SAMPL006
                                    8 .*       DEFINE SETC SYMBOL                                          SAMPL007
                                    9 .*                                                                   SAMPL008
                                   10          LCLC &TYPE                                                  SAMPL009
                                   11 .*                                                                   SAMPL010
                                   12 .*       CHECK NUMBER OF OPERANDS                                    SAMPL011
                                   13 .*                                                                   SAMPL012
                                   14          AIF   (N'&SYSLIST NE 2).ERROR1                              SAMPL013
                                   15 .*                                                                   SAMPL014
                                   16 .*       CHECK TYPE ATTRIBUTES OF OPERANDS                           SAMPL015
                                   17 .*                                                                   SAMPL016
                                   18          AIF   (T'&TO NE T'&FROM).ERROR2                             SAMPL017
                                   19          AIF   (T'&TO EQ 'C' OR T'&TO EQ 'G' OR T'&TO EQ'K').TYPECGK SAMPL018
                                   20          AIF   (T'&TO EQ 'D' OR T'&TO EQ 'E' OR T'&TO EQ'H').TYPEDEH SAMPL019
                                   21          AIF   (T'&TO EQ 'F').MOVE                                   SAMPL020
                                   22          AGO   .ERROR3                                               SAMPL021
                                   23 .TYPEDEH ANOP                                                       SAMPL022
                                   24 .*                                                                   SAMPL023
                                   25 .*       ASSIGN TYPE ATTRIBUTE TO SETC SYMBOL                        SAMPL024
                                   26 .*                                                                   SAMPL025
                                   27 &TYPE    SETC  T'&TO                                                 SAMPL026
                                   28 .MOVE    ANOP                                                        SAMPL027
                                   29 *        NEXT TWO STATEMENTS GENERATED FOR MOVE MACRO                SAMPL028
                                   30          L&TYPE    2,&FROM                                           SAMPL029
                                   31          ST&TYPE   2,&TO                                             SAMPL030
                                   32          MEXIT                                                       SAMPL031
                                   33 .*                                                                   SAMPL032
                                   34 .*       CHECK LENGTH ATTRIBUTES OF OPERANDS                         SAMPL033
                                   35 .*                                                                   SAMPL034
                                   36 TYPECGK  AIF   (L'&TO NE L'&FROM OR L'&TO GT 256).ERROR4             SAMPL035
      *** ERROR ***
                                   37 *       NEXT STATEMENT GENERATED FOR MOVE MACRO                     SAMPL036
                                   38          MVC   &TO,&FROM                                             SAMPL037
                                   39          MEXIT                                                       SAMPL038
                                   40 .*                                                                   SAMPL039
                                   41 .*       ERROR MESSAGES FOR INVALID MOVE MACRO INSTRUCTIONS          SAMPL040
                                   42 .*                                                                   SAMPL041
                                   43 .ERROR1  MNOTE 1,'IMPROPER NUMBER OF OPERANDS, NO STATEMENTS GENERATED' SAMPL042
                                   44          MEXIT                                                       SAMPL043
                                   45 .ERROR2  MNOTE 1,'OPERAND TYPES DIFFERENT, NO STATEMENTS GENERATED'  SAMPL044
                                   46          MEXIT                                                       SAMPL045
                                   47 .ERROR3  MNOTE 1,'IMPROPER OPERAND TYPES, NO STATEMENTS GENERATED'   SAMPL046
                                   48          MEXIT                                                       SAMPL047
                                   49 .ERROR4  MNOTE 1,'IMPROPER OPERAND LENGTHS, NO STATEMENTS GENERATED' SAMPL048
                                   50          MEND                                                        SAMPL049
```

It is used by the LD entry of the ESD and by the relocation dictionary to cross reference to the ESD.

(4) The column contains the address of the symbol (hexadecimal notation) for SD and LD type entries, and zeros for ER and WX type entries. For PC and CM type entries, it indicates the beginning address of the control section.

(5) This column contains the assembled length, in bytes, of the control section (hexadecimal notation).

(6) This column contains, for LD type entries, the identification (ID) number assigned to the ESD entry that identifies the control section in which the symbol was defined.

SOURCE AND OBJECT PROGRAM

This section of the listing documents the source statements and the resulting object program.

(7) This is the deck identification. It is the symbol that appears in the name field of the first TITLE statement.

(8) This is the information taken from the operand field of a TITLE statement.

(9) Listing page number.

(10) This column contains the assembled address (hexadecimal notation) of the object code.

(11) This column contains the object code produced by the source statement. The entries are always left-justified. The notation is hexadecimal. Entries are

```
(7)  EXAM        (8)  SAMPLE PROGRAM                                                                    (9)  PAGE    2
(10) LOC   (11) OBJECT CODE     (12) ADDR1 ADDR2  (13) STMT  (14) SOURCE STATEMENT                      (15) DOS CL3-0 09/16/67
                                                                                                            (16)
                                                                                                            (17)
                                                    52 ********************************************************************** SAMPL050
                                                    53 *         MAIN ROUTINE                                               * SAMPL051
                                                    54 ********************************************************************** SAMPL052
000000                                              55          CSECT                                                          SAMPL053
                                                    56          ENTRY SEARCH                                                   SAMPL054
000000 05C0                                         57 BEGIN    BALR  R12,0              ESTABLISH ADDRESSABILITY OF PROGRAM   SAMPL055
000002                                              58          USING *,R12              AND TELL THE ASSEMBLER                SAMPL056
000002 9857 C1A6              001A8                 59          LM    R5,R7,=A(LISTAREA,16,LISTEND)    LOAD LIST AREA PARAMS SAMPL057
000000                                              60          USING LIST,R5            REGISTER 5 POINTS TO THE LIST         SAMPL058
000006 45E0 C024              00026                 61 MORE     BAL   R14,SEARCH         FIND LIST ENTRY IN TABLE              SAMPL059
00000A 9180 C022       00024                        62          TM    SWITCH,NONE        CHECK TO SEE IF NAME WAS FOUND        SAMPL060
00000E 4710 C018              0001A                 63          BO    NOTTHERE           BRANCH IF NOT                         SAMPL061
000000                                              64          USING TABLE,R1           REGISTER 1 NOW POINTS TO TABLE ENTRY  SAMPL062
                                                    65          MOVE  TSWITCH,LSWITCH    MOVE FUNCTIONS                        SAMPL063
        *** ERROR ***
                                                    66                1,IMPROPER OPERAND TYPES, NO STATEMENTS GENERATED
                                                    67          MOVE  TNUMBER,LNUMBER    FROM LIST ENTRY
        *** ERROR ***
                                                    68          MOVE  TADDRESS,LADDRESS  TO TABLE ENTRY                        SAMPL069
                                                    69++        NEXT TWO STATEMENTS GENERATED FOR MOVE MACRO
000012 5820 500C              0000C                 70+         L     2,LADDRESS
000016 5020 1004              00004                 71+         ST    2,TADDRESS
00001A 9680 5008       00008                        72 NOTTHERE OI    LSWITCH,NONE       TURN ON SWITCH IN LIST ENTRY          SAMPL075
00001E 8756 C004              00006                 73          BXLE  R5,R6,MORE         LOOP THROUGH THE LIST                 SAMPL076
                                                    74          EOJ                      END OF PROGRAM, USER LIBRARY MACRO    SAMPL077
                                                    75++ 360N-CL-453 EOJ     CHANGE LEVEL 3-0
000022 0A0E                                         76+         SVC   14
000024                                              77 SWITCH   DS    X                                                        SAMPL078
000080                                              78 NONE     EQU   X'80'                                                    SAMPL079
                                                    79 ********************************************************************** SAMPL080
                                                    80 *         BINARY SEARCH ROUTINE                                      * SAMPL081
                                                    81 ********************************************************************** SAMPL082
000025 00
000026 947F C022       00024                        82 SEARCH   NI    SWITCH,255-NONE    TURN OFF NOT FOUND SWITCH             SAMPL083
00002A 9813 C1B2              001B4                 83          LM    R1,R3,=F'128,4,128'      LOAD TABLE PARAMETERS       SAMPL084
00002E 4111 C046              00048                 84          LA    R1,TABLAREA-16(R1) GET ADDRESS OF MIDDLE ENTRY        SAMPL085
000032 8830 0001              00001                 85 LOOP     SRL   R3,1               DIVIDE INCREMENT BY 2                 SAMPL086
000036 D507 5000 1008 00000 00008                   86          CLC   LNAME,TNAME        COMPARE LIST ENTRY WITH TABLE ENTRY   SAMPL087
00003C 4720 C04A              0004C                 87          BH    HIGHER             BRANCH IF SHOULD BE HIGHER IN TABLE   SAMPL088
000040 078E                                         88          BCR   8,R14             EXIT IF FOUND                         SAMPL089
                                                    89          SR    R1,R3              OTHERWISE IT IS LOWER IN THE TABLE  XSAMPL090
000042 1813                                                                              SO SUBTRACT INCREMENT                 SAMPL091
000044 4620 C030              00032                 90 MORE     BCT   R2,LOOP            LOOP 4 TIMES                          SAMPL092
        *** ERROR ***
000048 47F0 C050              00052                 91          B     NOTFOUND           ARGUMENT IS NOT IN THE TABLE          SAMPL093
00004C 1A13                                         92 HIGHER   AR    R1,R3              ADD INCREMENT                         SAMPL094
00004E 4620 C030              00032                 93          BCT   R2,LOOP            LOOP 4 TIMES                          SAMPL095
000052 9680 C022       00024                        94 NOTFOUND OI    SWITCH,NONE        TURN ON NOT FOUND SWITCH              SAMPL096
000056 07FE                                         95          BR    R14                EXIT                                  SAMPL097
```

machine instructions or assembled constants. Machine instructions are printed in full with a blank inserted after every four digits (two bytes). Constants may be only partially printed (see the PRINT assembler instruction in Assembler Instruction Statements).

(12) These two columns contain effective addresses (the result of adding together a base register value and displacement value):

1. The column headed ADDR1 contains the effective address for the first operand of an SS or an SI instruction.

2. The column headed ADDR2 contains the effective address of the second operand of any instruction referencing storage.

Both address fields contain six digits; however, if the high order digit is a zero, it is not printed.

(13) This column contains the statement number. A plus sign (+) to the right of the number indicates that the statement was generated as the result of macro-instruction processing. The maximum statement number is 65,535. If there are more than 65,535 statements, the statement number wraps-around.

(14) This column contains the source program statement. The following items apply to this section of the listing:

a. Source statements are listed, including those brought into the program by the COPY assembler instruction, and macro-definitions submitted with the main program for assembly. Listing control instructions are not printed, except for the following case: PRINT is listed when PRINT ON is in effect and a PRINT statement is encountered.

```
(7)                (8)                                                                                    (9)
EXAM          SAMPLE PROGRAM                                                                          PAGE    3
(10)          (11)              (12)          (13)          (14)                                    (15)          (16)
 LOC   OBJECT CODE     ADDR1 ADDR2   STMT    SOURCE STATEMENT                                  DOS CL3-0 09/16/67
                                                                                                                (17)
                                     97 *                                                                      SAMPL099
                                     98 *          THIS IS THE TABLE                                           SAMPL100
                                     99 *                                                                      SAMPL101
000058                              100           DS    0D                                                      SAMPL102
000058 0000000000000000             101 TABLAREA  DC    XL8'0'                                                  SAMPL103
000060 C1D3D7C8C1404040             102           DC    CL8'ALPHA'                                              SAMPL104
000068 0000000000000000             103           DC    XL8'0'                                                  SAMPL105
000070 C2C5E3C140404040             104           DC    CL8'BETA'                                               SAMPL106
000078 0000000000000000             105           DC    XL8'0'                                                  SAMPL107
000080 C4C5D3E3C1404040             106           DC    CL8'DELTA'                                              SAMPL108
000088 0000000000000000             107           DC    XL8'0'                                                  SAMPL109
000090 C5D7E2C9D3D6D540             108           DC    CL8'EPSILON'                                            SAMPL110
000098 0000000000000000             109           DC    XL8'0'                                                  SAMPL111
0000A0 C5E3C14040404040             110           DC    CL8'ETA'                                                SAMPL112
0000A8 0000000000000000             111           DC    XL8'0'                                                  SAMPL113
0000B0 C7C1D4D4C1404040             112           DC    CL8'GAMMA'                                              SAMPL114
0000B8 0000000000000000             113           DC    XL8'0'                                                  SAMPL115
0000C0 C9D6E3C140404040             114           DC    CL8'IOTA'                                               SAMPL116
0000C8 0000000000000000             115           DC    XL8'0'                                                  SAMPL117
0000D0 D2C1D7D7C1404040             116           DC    CL8'KAPPA'                                              SAMPL118
0000D8 0000000000000000             117           DC    XL8'0'                                                  SAMPL119
0000E0 D3C1D4C2C4C14040             118           DC    CL8'LAMBDA'                                             SAMPL120
0000E8 0000000000000000             119           DC    XL8'0'                                                  SAMPL121
0000F0 D4E4404040404040             120           DC    CL8'MU'                                                 SAMPL122
0000F8 0000000000000000             121           DC    XL8'0'                                                  SAMPL123
000100 D5E4404040404040             122           DC    CL8'NU'                                                 SAMPL124
000108 0000000000000000             123           DC    XL8'0'                                                  SAMPL125
000110 D6D4C9C3D9D6D540             124           DC    CL8'OMICRON'                                            SAMPL126
000118 F040404040404040             125           DC    CL8'0'                                                  SAMPL127
000120 D7C8C94040404040             126           DC    CL8'PHI'                                                SAMPL128
000128 0000000000000000             127           DC    XL8'0'                                                  SAMPL129
000130 E2C9C7D4C1404040             128           DC    CL8'SIGMA'                                              SAMPL130
000138 0000000000000000             129           DC    XL8'0'                                                  SAMPL131
000140 E9C5E3C140404040             130           DC    CL8'ZETA'                                               SAMPL132
                                    131 *                                                                      SAMPL133
                                    132 *          THIS IS THE LIST                                            SAMPL134
                                    133 *                                                                      SAMPL135
000148 D3C1D4C2C4C14040             134 LISTAREA  DC    CL8'LAMBDA'                                             SAMPL136
000150 0A                           135           DC    X'0A'                                                   SAMPL137
000151 00001D                       136           DC    FL3'29'                                                 SAMPL138
000154 00000000                     137           DC    A(BEGIN)                                                SAMPL139
000158 E9C5E3C140404040             138           DC    CL8'ZETA'                                               SAMPL140
000160 05                           139           DC    X'05'                                                   SAMPL141
000161 000005                       140           DC    FL3'5'                                                  SAMPL142
000164 00000032                     141           DC    A(LOOP)                                                 SAMPL143
000168 E3C8C5E3C1404040             142           DC    CL8'THETA'                                              SAMPL144
000170 02                           143           DC    X'02'                                                   SAMPL145
000171 00002D                       144           DC    FL3'45'                                                 SAMPL146
000174 00000000                     145           DC    A(BEGIN)                                                SAMPL147
000178 E3C1E44040404040             146           DC    CL8'TAU'                                                SAMPL148
000180 00                           147           DC    X'00'                                                   SAMPL149
```

b. Macro-definitions for system macro-instructions are not listed.

c. The statements generated as the result of a macro-instruction follow the macro-instruction in the listing.

d. Assembler or machine instructions in the source program that contain variable symbols are listed twice: as they appear in the source input, and with values substituted for the variable symbols.

e. Diagnostic messages are not listed in-line in the source and object program section. An error indicator, ***ERROR***, appears following the statement in error. The message appears in the diagnostic section of the listing.

f. MNOTE messages are listed in-line in the source and object program section. An MNOTE indicator appears in the diagnostic section of the listing. The MNOTE message format is: severity code, message text.

g. The MNOTE * form of the MNOTE statement results in an in-line message only. An MNOTE indicator does not appear in the diagnostic section of the listing.

h. When an error is found in a programmer macro-definition, it is treated like any other assembly error: the error indication appears after the statement in error, and a diagnostic is placed in the list of diagnostics. However, when an error is encountered during the expansion of a macro-instruction (system or programmer defined), the error indication appears in place of the erroneous statement, which is not listed. The error indication appears following the last statement listed before the erroneous statement was

```
(7)                (8)                                                                    (9)
EXAM          SAMPLE PROGRAM                                                          PAGE    4

 (10)   (11)              (12)        (13)      (14)                                  (15)      (16)
  LOC   OBJECT CODE      ADDR1 ADDR2  STMT    SOURCE STATEMENT                      DOS CL3-0 09/16/67
                                                                                              (17)
000181 000000                        148           DC    FL3'0'                                SAMPL150
000184 00000001                      149           DC    A(1)                                  SAMPL151
000188 D3C9E2E340404040              150           DC    CL8'LIST'                             SAMPL152
000190                               151           DC    X'1G'                                 SAMPL153
        *** ERROR ***
000190 0001C8                        152           DC    FL3'456'                              SAMPL154
000193 00
000194 00000000                      153           DC    A(0)                                  SAMPL155
000198 C1D3D7C8C1404040              154 LISTEND   DC    CL8'ALPHA'                            SAMPL156
0001A0 00                            155           DC    X'00'                                 SAMPL157
0001A1 000001                        156           DC    FL3'1'                                SAMPL158
0001A4 0000007B                      157           DC    A(123)                                SAMPL159
                                     158 *                                                     SAMPL160
                                     159 *         THESE ARE THE SYMBOLIC REGISTERS            SAMPL161
                                     160 *                                                     SAMPL162
000001                               161 R1        EQU   1                                     SAMPL163
000002                               162 R2        EQU   2                                     SAMPL164
000003                               163 R3        EQU   3                                     SAMPL165
000005                               164 R5        EQU   5                                     SAMPL166
000006                               165 R6        EQU   6                                     SAMPL167
000007                               166 R7        EQU   7                                     SAMPL168
00000C                               167 R12       EQU   12                                    SAMPL169
00000E                               168 R14       EQU   14                                    SAMPL170
                                     169 *                                                     SAMPL171
                                     170 *         THIS IS THE FORMAT DEFINITION OF LIST ENTRIES SAMPL172
                                     171 *                                                     SAMPL173
000000                               172 LIST      DSECT                                       SAMPL174
000000                               173 LNAME     DS    CL8                                   SAMPL175
000008                               174 LSWITCH   DS    C                                     SAMPL176
000009                               175 LNUMBER   DS    FL3                                   SAMPL177
00000C                               176 LADDRESS  DS    F                                     SAMPL178
                                     177 *                                                     SAMPL179
                                     178 *         THIS IS FORMAT DEFINITION OF TABLE ENTRIES  SAMPL180
                                     179 *                                                     SAMPL181
000000                               180 TABLE     DSECT                                       SAMPL182
000000                               181 TNUMBER   DS    FL3                                   SAMPL183
000003                               182 TSWITCH   DS    C                                     SAMPL184
000004                               183 TADDRESS  DS    F                                     SAMPL185
000008                               184 TNAME     DS    CL8                                   SAMPL186
000000                               185           END   BEGIN                                 SAMPL187
0001A8 0000014800000010              186                 =A(LISTAREA,16,LISTEND)
0001B4 0000008000000004              187                 =F'128,4,128'
```

encountered, and the associated diagnostic message is placed in the list of diagnostics.

i. Literals will appear in the listing following an LTORG or the END statement or both. Literals are identified by the equals (=) sign preceding them.

j. If the END statement contains an operand, the transfer address appears in the location column (LOC).

k. In the case of COM, CSECT, and DSECT statements, the location field contains the beginning address of these control sections i.e., the first occurrence.

l. For a USING statement, the location field contains the value of the first operand.

m. For LTORG and ORG statements, the location field contains the location assigned to the literal pool or the value of the ORG operand.

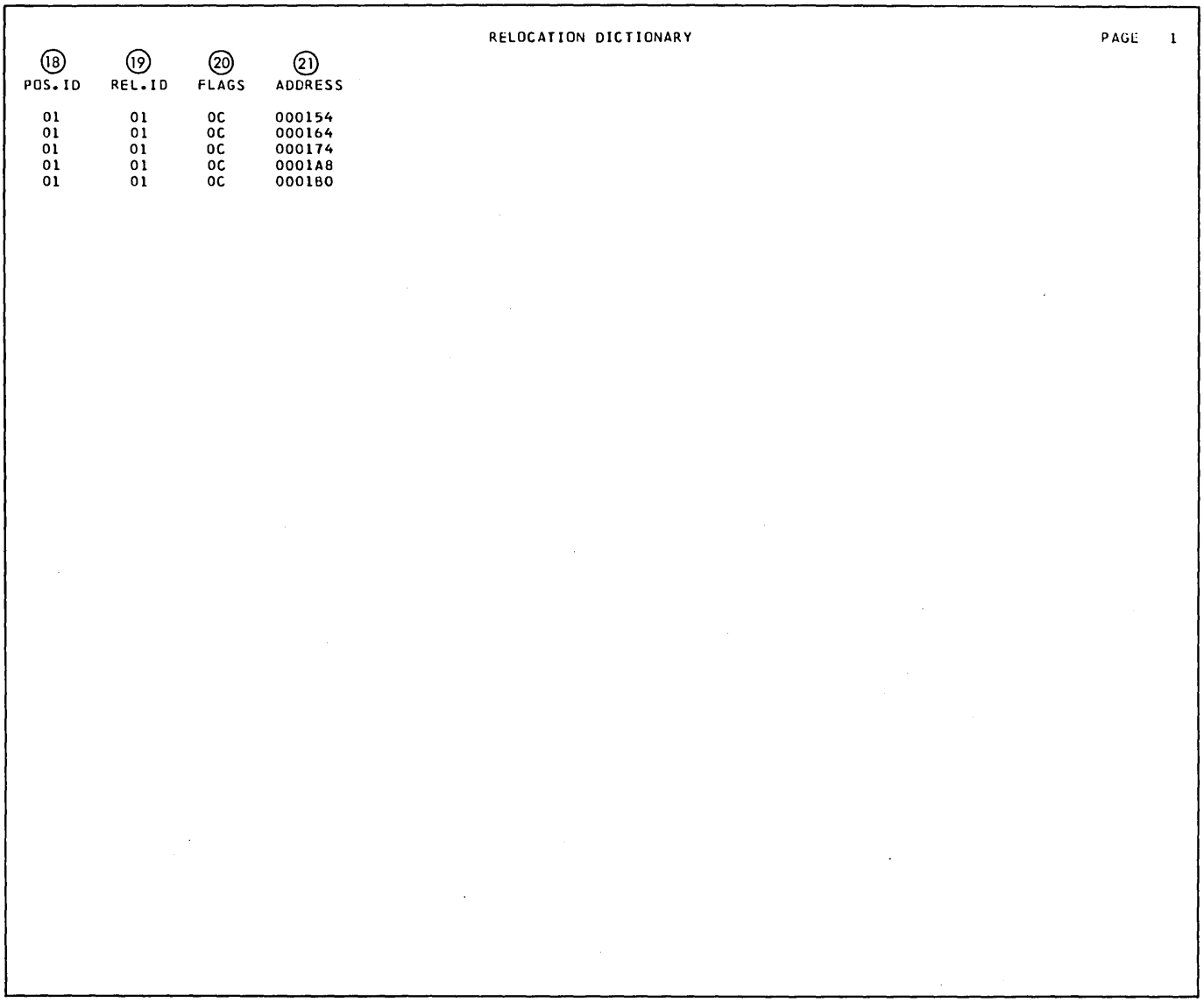
n. For an EQU statement the location field contains the value assigned.

o. Generated statements always print in normal statement format. Because of this, it is possible for a generated statement to occupy two or more continuation lines on the listing. This is unlike source statements which are restricted to one continuation line.

(15) This field indicates the assembler level and version number, e.g., DOS CL2-1 reads as DOS assembler level 2, version 1.

(16) Current date obtained from SET card.

(17) Identification-sequence field from the source statement.



RELOCATION DICTIONARY PAGE 1

(18) POS.ID	(19) REL.ID	(20) FLAGS	(21) ADDRESS
01	01	0C	000154
01	01	0C	000164
01	01	0C	000174
01	01	0C	0001A8
01	01	0C	0001B0

RELOCATION DICTIONARY

This section of the listing contains the relocation dictionary information passed to the linkage editor in the object module. The entries describe the address constants in the assembled program that are affected by relocation.

(18) This column contains the external symbol dictionary ID number assigned to the ESD entry that describes the control section in which the address constant is used as an operand.

(19) This column contains the external symbol dictionary ID number assigned to the ESD entry that describes the control section in which the referenced symbol is defined.

(20) The two-digit hexadecimal number in this column is interpreted as follows:

First Digit -- a zero indicates that the entry describes an A-type, a Y-type, or a CCW address constant.
-- a one indicates that the entry describes a V-type address constant.

Second Digit -- the first three bits of this digit indicate the length and sign of the address constant as follows:

Bits 0 and 1	Bit 2
00 = 1 byte	0 = +
01 = 2 bytes	1 = -
10 = 3 bytes	
11 = 4 bytes	

(21) This column contains the assembled address of the field where the address constant is stored.

```
                                        CROSS-REFERENCE                                  PAGE    1

 (22)       (23)  (24)                      (26)
SYMBOL     LEN   VALUE  DEFN

BEGIN      00002 000000 00057   0137  0145  0185
HIGHER     00002 00004C 00092   0087
LADDRESS   00004 00000C 00176   0070
LIST       00001 000000 00172   0060
LISTAREA   00008 000148 00134   0059  0186
LISTEND    00008 000198 00154   0059  0186
LNAME      00008 000000 00173   0086
LNUMBER    00003 000009 00175
LOOP       00004 000032 00085   0090  0093  0141
LSWITCH    00001 000008 00174   0072
MORE       00004 000006 00061   0073
MORE       00004 000006 00090
NONE       00001 000080 00078   0062  0072  0082  0094
NOTFOUND   00004 000052 00094   0091
NOTTHERE   00004 00001A 00072   0063
R1         00001 000001 00161   0064  0083  0084  0084  0089  0092
R12        00001 00000C 00167   0057  0058
R14        00001 00000E 00168   0061  0088  0095
R2         00001 000002 00162   0090  0093
R3         00001 000003 00163   0083  0085  0089  0092
R5         00001 000005 00164   0059  0060  0073
R6         00001 000006 00165   0073
R7         00001 000007 00166   0059
SEARCH     00004 000026 00082   0056  0061
SWITCH     00001 000024 00077   0062  0082  0094
TABLAREA   00008 000058 00101   0084
TABLE      00001 000000 00180   0064
TADDRESS   00004 000004 00183   0071
TNAME      00008 000008 00184   0086
TNUMBER    00003 000000 00181
TSWITCH    00001 000003 00182
```

CROSS-REFERENCE

This section of the listing information concerns symbols -- where they are defined and used in the program.

(22) This column contains the symbols.

(23) This column states the length (decimal notation), in bytes, of the field occupied by the symbol value.

(24) This column contains either the address the symbol represents, or a value to which the symbol is equated.

(25) This column contains the statement number of the statement in which the symbol was defined.

(26) This column contains the statement numbers of statements in which the symbol appears as an operand.

The following notes apply to the cross-referencing section:

- Symbols appearing in V-type address constants do not appear in the cross-reference listing.
- A PRINT OFF listing control instruction does not affect the production of the cross-reference section of the listing.
- Undefined symbols appear in the cross-reference section. However, only the symbol column and the reference column have entries.

```
EXAM                                                    DIAGNOSTICS                                    PAGE   1
 (27)       (28)        (29)
STMT  ERROR CODE    MESSAGE

  36  IJQ073        ILLEGAL NAME FIELD
  65  IJQ059        UNDEFINED SEQUENCE SYMBOL
  66  IJQ037        MNOTE STATEMENT
  67  IJQ088        UNDEFINED OPERATION CODE
  90  IJQ023        PREVIOUSLY DEFINED NAME
 151  IJQ039        INVALID DELIMITER


   6 STATEMENTS FLAGGED IN THIS ASSEMBLY
```

DIAGNOSTICS

This section contains the diagnostic messages issued as a result of error conditions encountered in the program. Explanatory notes for each message are contained in Appendix N.

(27) This column contains the number of the statement in error.

(28) This column contains the message identifier.

(29) This column contains the message.

The following notes apply to the diagnostics section:

- An MNOTE indicator of the form MNOTE STATEMENT appears in the diagnostic section, if an MNOTE statement is issued by a macro-instruction. The MNOTE statement itself is in-line in the source and object program section of the listing.

- A message identifier consists of six characters and is of the form:

IJQxxx

IJYxxx

IJQ
identifies the issuing agent as DOS/TOS D assembler.

IJY
identifies the issuing agent as DOS F assembler.

xxx
is a unique number assigned to the message.

- Two statistical messages may appear in the listing. They are:

1. A message indicating the total number of statements in error. If no statements are in error, the message

NO STATEMENTS FLAGGED IN THIS ASSEMBLY

is printed following the Cross-Reference section and no diagnostic section is printed.

2. A message if one or more Y-type address constants appear in the program.

AT LEAST ONE RELOCATABLE Y-TYPE CONSTANT IN ASSEMBLY.

This message if issued, appears before the diagnostic section.

APPENDIX J: ASSEMBLER LANGUAGES--FEATURES COMPARISON CHART

Features not shown below are common to all assemblers. In the chart:
Dash = Not allowed.
X = As defined in Operating System/360 Assembler Language Manual.

Feature	Model 20 Basic Assembler	Basic Programming Support/360: Basic Assembler	7090/7094 Support Package Assembler	BPS 8K Tape, BOS 8K Disk Assemblers	DOS, TOS D Assemblers	OS/360 Assembler DOS F Assembler
No. of Continuation Cards/Statement (exclusive of macro-instructions)	0	0	0	1	1	2
Input Character Code	EBCDIC	EBCDIC	BCD & EBCDIC	EBCDIC	EBCDIC	EBCDIC
ELEMENTS:						
Maximum Characters per symbol	4	6	6	8	8	8
Character self-defining terms	1 Char. only	1 Char. only	X	X	X	X
Binary self-defining terms	- -	- -	- -	X	X	X
Length attribute reference	- -	- -	- -	X	X	X
Literals	- -	- -	- -	X	X	X
Extended mnemonics	- -	- -	X	X	X	X
Maximum Location Counter value	$2^{14}-1$	$2^{16}-1$	$2^{24}-1$	$2^{24}-1$	$2^{24}-1$	$2^{24}-1$
Multiple Control Sections per assembly	- -	- -	- -	X	X	X
EXPRESSIONS:						
Operators	+ -	+ -*	+ -*/	+ -*/	+ -*/	+ -*/
Number of terms	3	3	16	3	16	16
Levels of parentheses	- -	- -	- -	1	5	5
Complex relocatability	- -	- -	- -	X	X	X
ASSEMBLER INSTRUCTIONS:						
DC and DS						
Expressions allowed as modifiers	- -	- -	- -	- -	X	X
Multiple operands	- -	- -	- -	- -	- -	Less than 33
Multiple constants in an operand	- -	- -	- -	Except Address Consts.	X	X
Bit length specifications	- -	- -	- -	- -	- -	X
Scale modifier	- -	- -	- -	X	X	X
Exponent Modifier	- -	- -	- -	X	X	X
DC types	Only C, X, H, Y, Q	Except B, P, Z, V, Y, S, Q	Except B, V, Q	Except Q	Except Q	X*
DC duplication factor	Except Y	Except A	X	Except S	X	X

*The DOS F Assembler does not allow Q-type address constants.

Feature	Model 20 Basic Assembler	Basic Programming Support/360: Basic Assembler	7090/7094 Support Package Assembler	BPS 8K Tape, BOS 8K Disk Assemblers	DOS, TOS D Assemblers	OS/360 Assembler DOS F Assembler
DC duplication factor of zero	Except Y	- -	- -	Except S	X	X
DC length modifier	Except H, Y	Except H, E, D	X	X	X	X
DS types	Only H, C	Only C, H, F, D	Only C, H, F, D	X	X	X
DS length modifer	Only C	Only C	Only C	X	X	X
DS maximum length modifier	256	256	256	256	65,535	65,535
DS constant subfield permitted	- -	- -	- -	X	X	X
COPY	- -	- -	- -	- -	X	X
CSECT	- -	- -	- -	X	X	X
DSECT	--	--	--	X	X	X
ISEQ	- -	- -	- -	X	X	X
LTORG	- -	- -	- -	X	X	X
PRINT	- -	- -	- -	X	X	X
TITLE	- -	- -	X	X	X	X
COM	- -	- -	- -	- -	X	X
ICTL	- -	1 operand (1 or 25 only)	1 operand	X	X	X
USING	2 operands (operand 1 relocatable only)	2 operands (operand 1 relocatable only)	2-17 operands (operand 1 relocatable only)	6 operands	X	X
DROP	1 operand only	1 operand only	X	5 operands	X	X
CCW	- -	operand 2 (relocatable only)	X	X	X	X
ORG	no blank operand	no blank operand	no blank operand	X	X	X
ENTRY	1 operand only	1 operand only	1 operand only	1 operand only	X	X
EXTRN	1 operand only	1 operand only (max 14)	1 operand only	1 operand only	X	X
WXTRN	- -	- -	- -	- -	DOS 14K D only	- -
CNOP	- -	2 decimal digits	2 decimal digits	2 decimal digits	X	X
PUNCH	- -	- -	- -	X	X	X
REPRO	- -	- -	- -	X	X	X
Macro Instructions	S/360 Model 20 IOCS only	- -	- -	X	X	X

Macro Facility Features	BPS 8K Tape, BOS 8K Disk Assemblers	DOS, TOS Assemblers	OS/360 Assembler
Operand Sublists	- -	X	X
Attributes of macro-instruction operands inside macro definitions and symbols used in conditional assembly instructions outside macro definitions.	- -	X	X
Subscripted SET symbols	- -	X	X
Maximum number of operands	49	100**	200
Conditional assembly instructions outside macro definitions	- -	X	X
Maximum number of SET symbols			
global SETA	16	*	*
global SETB	128	*	*
global SETC	16	*	*
local SETA	16	*	*
local SETB	128	*	*
local SETC	0	*	*

*The number of SET symbols permitted by the Disk and Tape Operating Systems Assemblers and the Operating System Assembler is variable, dependent upon the available main storage.

**Maximum number of operands in DOS assembler F is 200.

Note: The maximum size of a character expression is 127 in DOS (assembler D) and TOS and 255 characters in OS and DOS (assembler F).

Figure K-1 lists the control cards necessary to assemble a program. The card groups are listed in the order in which they must appear. All job control cards enter the system via SYSRDR, all others via SYSIPT. The same device may be assigned for both SYSRDR and SYSIPT. If this device is a disk file, the combined file must be designated as SYSIN. Job Control statements are described in the publications: IBM System/360 Disk Operating System, System Control and System Service Programs or IBM System/360 Tape Operating System, System Control and System Service Programs. The form numbers are listed in the preface.

Card Group	Card Arrangement	Comments
Job Control	// JOB	First card in group, always required.
	// ASSGN SYSSLB,..	Used when the source statement library is on a separate (private) file.[1]
	// ASSGN SYSIPT,..	Source program input
	// ASSGN SYSLST,..	Program listing

	// ASSGN SYS001,...
	// ASSGN SYS002,...
	// ASSGN SYS003,...
		} Work files

	// ASSGN SYSPCH,..	Required when DECK option is specified.
	// ASSGN SYSLNK,..	Required when assemble-and-execute is specified.
	// OPTION DECK,...	Optional. Used to indicate desired assembler functions.
	// EXEC ASSEMBLY	Required.
Assembler Input	Source Deck	Source statements (machine, assembler, and macro instructions).
	/*	Indicates end-of-data set
Job Control	/&	End of job statement

[1]SYSSLB is assigned as follows:

For DOS -- SYSSLB cannot be assigned for the 10K assembler. If SYSSLB is assigned for the 14K D assembler or the F assembler, it is concatenated with the source statement library on SYSRES. (The assembler searches first SYSSLB and then the SYSRES library.) The 10K D assembler and -- if SYSSLB is not assigned -- the 14K D and F assemblers use only the source statement library of SYSRES.

For TOS -- Both TOS assembler variants use either SYSSLB or the source statement library on SYSRES. They use SYSSLB if it is assigned. If it is not assigned, they use the SYSRES library.

Note 1: Only those assignments and options not already in effect are required.

Note 2: Assignments for SYSIN and/or SYSOUT must be accomplished by permanent assignments. For details see the publications for DOS and TOS system control and system service programs (see preface).

Figure K-1. Card Input for an Assembly

Symbolic Unit	Function and Device
SYSRDR (Required if the SYSIN option is not used)	Job control statement input device. May be the same device as SYSIPT except for combined input from IBM 2311 or 2314 Disk Extent (see SYSIN). IBM 1442, 2520, or 2540 Card Read Punch, IBM 2501 Card Reader, IBM 2400-series Magnetic Tape Unit, or IBM 2311 or 2314 Disk Extent for the disk system.
SYSIPT (Required if the SYSIN option is not used)	Source program input device. May be the same device as SYSRDR except for combined input from IBM 2311 or 2314 Disk Extent (see SYSIN). IBM 1442, 2520, or 2540 Card Read Punch, IBM 2501 Card Reader, IBM 2400-series Magnetic Tape Unit (7- or 9-track), or IBM 2311 or 2314 Disk Extent for the disk system. If the Data Conversion feature was used to prepare the 7-track tape, it must also be used to read the tape. The tape or disk records must be 80-byte unblocked records.
SYSIN (Required for combined disk input. Optional for combined card or tape input)	Used for a combined input file for SYSRDR and SYSIPT. IBM 1442, 2520, or 2540 Card Read Punch, IBM 2501 Card Reader, IBM 2400-series Magnetic Tape Unit, or IBM 2311 or 2314 Disk Extent for the disk system. SYSIN can be used in lieu of the SYSRDR and SYSIPT designation when the file is card or tape input. It must be used when the file is disk input (disk system only).
SYSLST (Required if the SYSOUT option is not used)	Program listing device. IBM 1403, 1404 (continuous forms only), or 1443 Printer. IBM 2400-series Magnetic Tape Unit (9-track, or 7-track with or without the Data Conversion feature) or IBM 2311 or 2314 Disk Extent for the disk system. Listing on tape or disk appears as 121-character print images (a single forms-control byte followed by a 120-character line image).
SYSPCH (Optional)	Object program output device. IBM 1442, 2520, or 2540 Card Read Punch. IBM 2400-series Magnetic Tape Unit (9-track, or 7-track with the Data Conversion feature), or IBM 2311 or 2314 Disk Extent for the disk system. Output on tape or disk is in 81-byte unblocked records. Not used when the Assemble-and-Execute or the NODECK option is specified.
SYSOUT (Optional)	Used for a combined output file for SYSLST and SYSPCH to a single tape unit. IBM 2400-series Magnetic Tape Unit (9-track, or 7-track with the Data Conversion feature).
SYSLNK (Optional)	Used for temporary storage of assembler output. Required only when the Assemble-and-Execute option is specified. IBM 2400-series Magnetic Tape Unit (9-track, or 7-track with the Data Conversion feature) for the tape system or IBM 2311 or 2314 Disk Extent for the disk system. This extent may be on the same device that contains the DOS resident system.
SYS001 SYS002 SYS003 (Required)	Used for temporary work area during assembly. IBM 2400-series Magnetic Tape Unit (9-track, or 7-track with the Data Conversion feature) for either the tape or disk systems or three IBM 2311 or 2314 Disk Extents for the disk system. These extents may be on the same device that contains the DOS resident system. For details of work file assignment see the publication for DOS system generation (see preface).
SYSSLB (Optional or 2314)	Must be same device type as SYSRES. See Appendix K, Figure K-1.

NOTE: The 2311 or 2314 can be used for one or more of the symbolic units SYSRDR, SYSIPT, SYSIN, SYSPCH, or SYSLST only if a supervisor has been SYSGEN'd that can accomodate input from disk storage or output to disk storage for these units. For details see the DOS system generation manual (see preface)

Figure K-2. Device Assignments

Input and Output Using an IBM 1442 or 2520 Card Read Punch: Whenever an IBM 1442 or 2520 Card Read Punch is assigned to SYSRDR, SYSIPT, or SYSIN and also to SYSPCH, a number of blank cards sufficient for punching the output deck must follow the /* card which follows the assembler END statement in the source deck. This is to prevent erroneously punching the cards of a following job step. Any extra cards that are not needed are automatically bypassed.

Figure K-3. Operating Considerations

Card Group	Card Arrangement	Comments
Job Control	// JOB ...	First card in group, always required.
	// ASSGN SYSRLB...	Used by the Linkage editor when the relocatable library is on a separate (private) file and previously assembled modules are to be included.
	// ASSGN SYSSLB...	Used when the source statement library is on a separate (private) file.[1]
	// ASSGN SYSIPT...	Source program input
	// ASSGN SYSLST...	Program listing
	// ASSGN SYSLNK...	Required for assemble-and-execute.

	// ASSGN SYS001...
	// ASSGN SYS002...
	// ASSGN SYS003...
		} Work files

	// OPTION LINK....	Required. Used to indicate LINK option and any additional assembler functions desired.
	// EXEC ASSEMBLY	Required
Assembler Input Source Deck		Source statements (machine-, assembler- and macro-instructions). NOTE: If the operand of the END statement is omitted, a PHASE card must precede the // EXEC ASSEMBLY card or an ENTRY card must follow the END statement (tape system only).
	/*	Indicates end-of-data set
Job Control	ENTRY ...	
	// EXEC LNKEDT	Calls the Linkage Editor
	// EXEC	
Data	Data, if any	
	/*	End-of-data set indicator
Job Control	/&	End of job statement

[1] See Figure K-1

Note 1: Only those assignments and options not already in effect are required.

Note 2: Assignments for SYSIN and/or SYSOUT must be accomplished by permanent assignments. For details see the publications for DOS and TOS system control and system service programs (see preface).

Figure K-4. Card Input for Assembly, Linkage Editing, and Execution

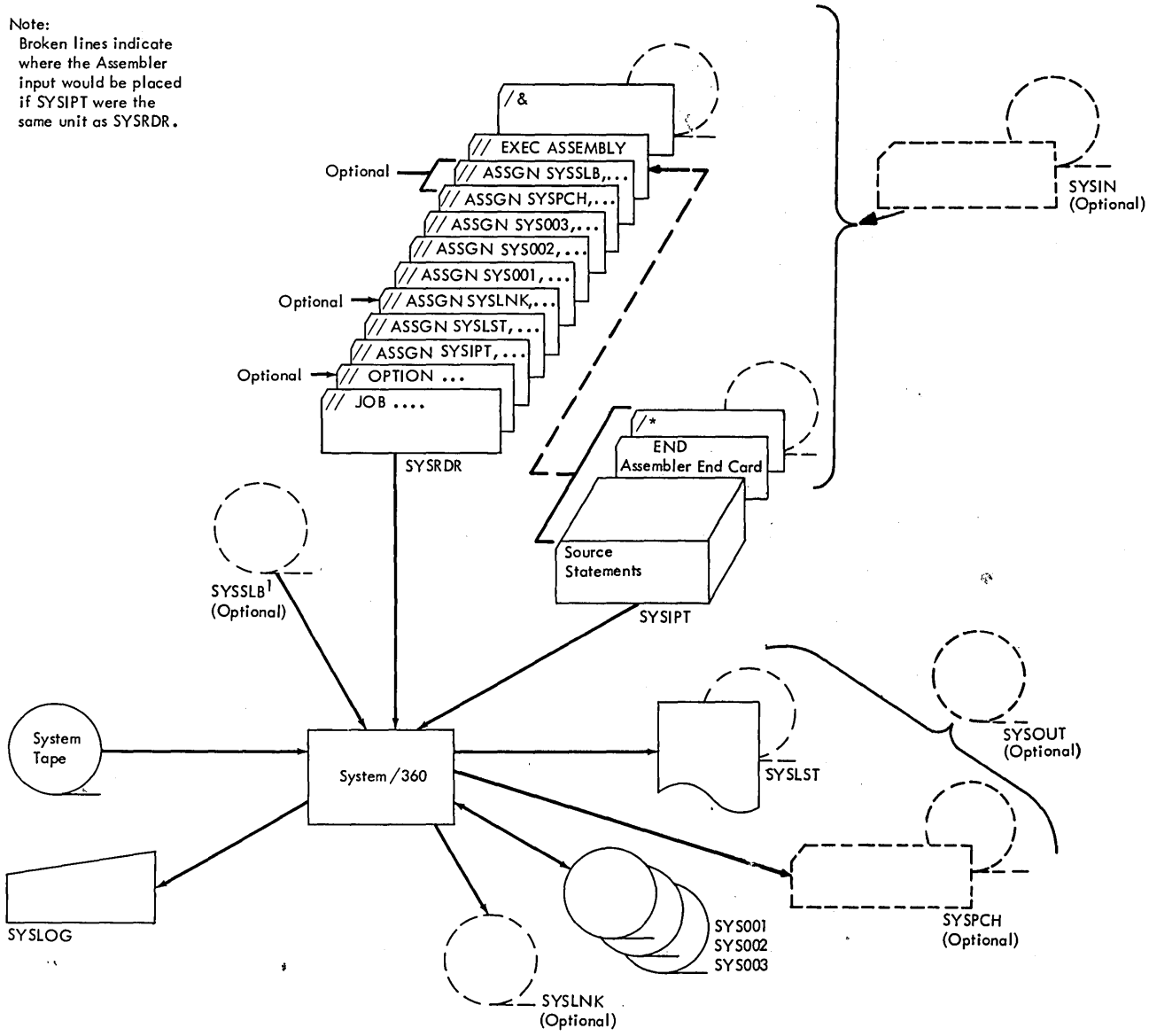


Figure K-5. I/O Units Used by the Tape Assembler

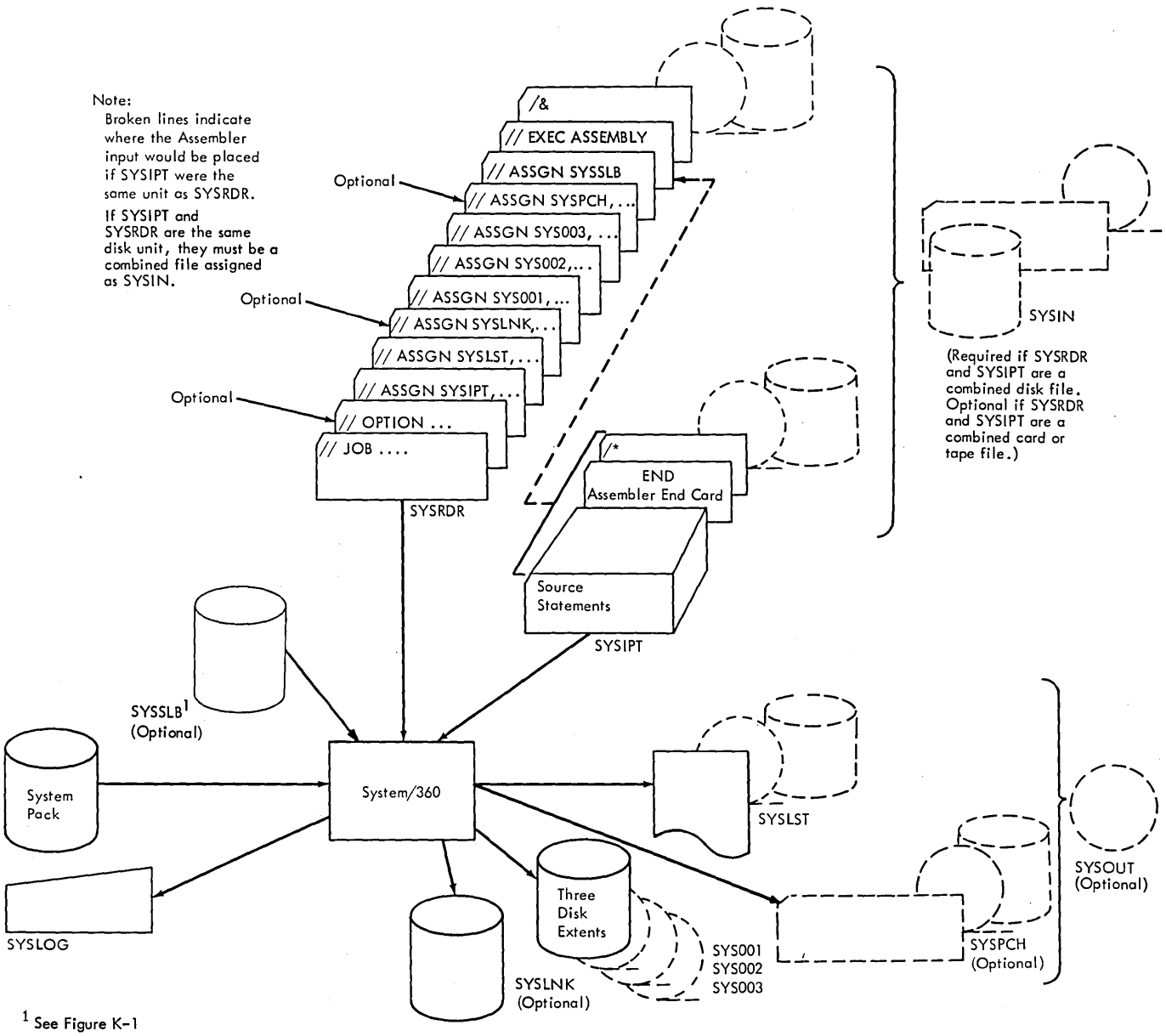


Figure K-6. I/O Units Used by the Disk Assembler

Assemble-and-execute	Assemble-and-execute (Include object routines from the relocatable library)	Assemble-and-execute (Include object routines from cards)	Assemble-and-execute (Include object routines from the relocatable library and from cards)
// JOB ...	// JOB ...	// JOB ...	// JOB ...
// ASSGN SYSIPT,..	// ASSGN SYSIPT,..	// ASSGN SYSIPT,..	// ASSGN SYSIPT,...
// ASSGN SYSLST,...	// ASSGN SYSLST,...	// ASSGN SYSLST,..	// ASSGN SYSLST,...
// ASSGN SYS001,..	// ASSGN SYS001,..	// ASSGN SYS001,..	// ASSGN SYS001,..
// ASSGN SYS002,..	// ASSGN SYS002,..	// ASSGN SYS002,..	// ASSGN SYS002,..
// ASSGN SYS003,..	// ASSGN SYS003,..	// ASSGN SYS003,..	// ASSGN SYS003,..
// ASSGN SYSLNK,...	// ASSGN SYSLNK,...	// ASSGN SYSLNK,..	// ASSGN SYSLNK,..
// OPTION LINK,...	// OPTION LINK,..	// OPTION LINK,..	// OPTION LINK,...
// EXEC ASSEMBLY	// EXEC ASSEMBLY	// EXEC ASSEMBLY	// EXEC ASSEMBLY
Source Deck /*	Source deck /*	Source deck /*	Source deck /*
	INCLUDE SUBR1 INCLUDE SUBR2	INCLUDE Object deck(s) /*	INCLUDE SUBR1 INCLUDE Object deck(s) /* INCLUDE SUBR2 INCLUDE SUBRT
ENTRY	ENTRY	ENTRY	ENTRY .....
// EXEC LNKEDT	// EXEC LNKEDT	// EXEC LNKEDT	// EXEC LNKEDT
Any Job Control cards needed for the programs to be executed.			
// EXEC	// EXEC	// EXEC	// EXEC
Data, if any /*	Data, if any /*	Data, if any /*	Data, if any /*
/&	/&	/&	/&
If SYSRDR and SYSIPT are different units, a /& card must be placed after the last EXEC card in SYSRDR, and should be placed after the last /* in SYSIPT.			

Figure K-7. Card Input for Different Variations of Assembly, Linkage Editing and Execution.

The EXEC ASSEMBLY statement causes the job control program to look for a phase with the name ASSEMBLY in the Core Image Library and load it into main storage. Since duplicate names cannot appear in a library, and every version of the assembler processor has the same phase name, only one of them can be in the Core Image Library at a time. Therefore the variant best suited for the particular installation is normally included in the Core Image Library. The programmer can, however, select another variant from the Relocatable Library and include it in the Core Image Library instead of the variant that is already there.

Figure L-1 shows the Job Control cards required to bring a particular assembler variant from the Relocatable Library into the Core Image Library, and Figure L-2 shows the valid assembler names (the names under which the variants would be cataloged in the Relocatable Library). After the variant has been included in the Core Image Library, it can be loaded and executed through the EXEC ASSEMBLY statement.

```
// JOB CONDENSE          }
// EXEC MAINT            }
   DELETC ASSE.ALL       } This job not needed in TOS
   CONDS CL              }
/&                       }
// JOB LINKASM
// OPTION CATAL
   INCLUDE name*
// EXEC LNKEDT
/&
```

* 'name' selected from those listed in Figure K-5

Figure L-1. Card Input for Selecting Different Assembler Variants

Variants IJQT16, IJQD16TW, and IJQD16DW must be used if the assembler is to be run in less than 14K of available core. Variants IJQT32 and IJQD32 may be used if available core is never less than 14K. The IJQ variants are D assemblers. Variant IJYASM (the DOS F assembler) may be used if available core is never less than 45,056 bytes.

IJQT32 and IJQD32 are generally faster than LJQT16 and IJQD16DW or IJQD16TW, respectively, because they have test I/O buffering and can use the additional core to build larger symbol tables. The difference in speed varies with the amount of additional core and the number of symbols in the assembly.

Thus, if the assembly has few symbols or if only a small amount of additional core is available to a larger variant, the larger and smaller variants will be nearly equal in speed.

For comparable assemblies, DOS assembler F (IJYASM) is up to 45% faster than DOS assembler D.

Note 1: The descriptions 16K and 32K refer to the machine size required to run the 10K and 14K variants respectively (except that the 14K variant can run on a System/360 Model 30 with 24K of core).

Note 2: Some installations have two or more assemblers in the Core Image Library. In such instances, the phase names have been changed to avoid duplicate names in the library. (Refer to IBM System/360 Disk Operating System, System Generation and Maintenance, Form C24-5033 and IBM System/360 Tape Operating System, System Generation and Maintenance, Form C24-5015).

Name	System	Work Files[1]	Minimum Core[2]
IJQD16DW	DOS	Disk	10,240
IJQD16TW	DOS	Tape	10,240
IJQD32	DOS	Mixed	14,336
IJYASM	DOS	Mixed	45,056
IJQT16	TOS	Tape	10,240
IJQT32	TOS	Tape	14,336

1. Mixed work files mean any combination of 2400-series tapes and/or 2311 and/or 2314 disk extents for SYS001, SYS002, and SYS003. In general, the assembler uses SYS001 and SYS002 as serial files and SYS003 as a random access file.
2. Minimum core refers to the minimum number of contiguous bytes necessary for the particular assembler variant to function correctly.

Figure L-2. Assembler Variants

APPENDIX M: OBJECT DECK OUTPUT

Figure M-1 lists the card groups that make up the output deck produced by the assembler. The groups are listed in the order in which they appear in the output deck.

Note: No output deck will be produced when NODECK appears in the OPTION card.

The formats of the ESD, TXT, RLD, END, and SYM cards are shown in Figures M-2 and M-3.

Card Group	Remarks
Reproduced Cards	These reproduced cards result from REPRO or PUNCH instructions located before START.
Symbol Table (SYM)	Produced when SYM appears in the OPTION card.
External Symbol Dictionary (ESD)	
Problem Program	Consists of text (TXT) and reproduced cards. The reproduced cards result from REPRO or PUNCH instructions located after START.
Relocation Dictionary (RLD)	Produced if relocatable constants are present.
END Card	Produced as the last card of the output deck.
Object Deck Identification	
The 4-character assembly identification label punched into the name entry of the first TITLE card in the source program is punched into columns 73-76 of each record in the object deck. If there is no label, these columns are left blank.	
Object Deck Sequencing Numbering	
An assembler-generated sequence number is punched into columns 77-80 of each card in the object deck.	

Figure M-1. Assembler Output Deck

The information in each card is in Extended Binary Coded Decimal Interchange Code.

Columns	Punched
ESD Card	
1	Multiple punch (12-2-9). Identifies this as a loader card.
2-4	ESD--External Symbol Dictionary card.
11-12	Number of bytes of information contained in this card.
15-16	External symbol identification number (ESID) of the first SD, PC, WX, or ER on this card. Relates the SD, PC, WX, or ER to a particular control section.
17-72	Variable information. 8 positions. Name. 1 position. Type code to indicate SD, PC, LD, WX, or ER. 3 positions. Assembled origin. 1 position. Blank. 3 positions. Control section length, if an SD-type or a PC-type. If an LD-type, this field contains the external symbol identification number (ESID) of the SD or PC containing the label.
73-76	Program identification taken from the name field of the first TITLE statement.
77-80	Sequence number.
TXT Card	
1	Multiple punch (12-2-9). Identifies this as a loader card.
2-4	TXT--Text card.
6-8	Assembled origin (address of first byte to be loaded from this card).
11-12	Number of bytes of text to be loaded.
15-16	External symbol identification number (ESID) of the control section (SD) containing the text.
17-72	Up to 56 bytes of text--data or instructions to be loaded.
73-76	Program identification taken from the name field of the first TITLE statement.
77-80	Sequence number.
RLD Card	
1	Multiple punch (12-2-9).
2-4	RLD--Relocation Dictionary card.
11-12	Number of bytes of information contained in the card.
17-72	Variable information (multiple items). 2 positions. Pointer to the relocation factor of the contents of the load constant. 2 positions. Pointer to the relocation factor of the control sections in which the load constant occurs. 1 position. Flag indicating type of constant. 3 positions. Assembled address of load constant.
73-76	Program identification taken from the name field of the first TITLE statement.
77-80	Sequence number.
END Card	
1	Multiple punch (12-2-9).
2-4	END
6-8	Assembled origin of the label supplied to the Assembler in the END card (optional).
15-16	ESID number of the control section to which this END card refers.
17-22	Symbolic label supplied to the Assembler if this label was not defined within the assembly.
73-76	Program identification taken from the name field of the first TITLE statement.
77-80	Sequence number.

Figure M-2. Format of ESD, TXT, RLD, and END Cards.

Columns	Contents
1	12-2-9 punch
2-4	SYM
5-10	Blank
11-12	Number of bytes in the Variable Field
13-14	Blank
15-16	ESID
17-72	The Variable Field (see below)
73-76	Deck ID (from the first TITLE card) or blank
77-80	Card sequence number

The variable field (columns 17-72) contains up to 56 bytes of AUTOTEST text. The items making up the text are packed together; consequently the last card may contain less than 56 bytes of text in the variable field. The contents of the fields within an individual text item are, as follows:

1. Organization (1 byte):

   Bit 0: 0
   Bit 1: 1
   Bit 2: 1
   Bit 3: 1
   Bit 4: 1 = DC or DS
          0 = not DC or DS
   Bits 5-7: Length of name minus one.

2. Address (3 bytes): displacement from base of control section.

3. Symbol Name (1-8 bytes): symbolic name of the particular item.

The following fields are present only for data-type items:

4. Data Type (1 byte):

   X'00' character
   X'04' hexadecimal
   X'08' binary
   X'10' full word, fixed point
   X'14' half word, fixed point
   X'18' single precision floating point
   X'1C' double precision floating point
   X'20' A-type or Q-type address constant
   X'24' Y-type address constant
   X'28' S-type address constant
   X'2C' V-type address constant
   X'30' packed decimal
   X'34' zoned decimal

5. Length (1-2 bytes): length of data item minus one. Occupies two bytes for character, hexadecimal and binary items; otherwise one.

6. Multiplicity (1 byte): always X'01'.

Figure M-3. Format of the SYM card.

New text can be substituted for assembled text using the REP card. Each REP card must contain the assembled address of the first byte to be replaced and the identification of the control section to which it refers, and may contain from two to 22 bytes of text. The text is substituted, byte for byte, for the original text, beginning at the address specified. The address, the control section reference, and the new text must be stated in hexadecimal. The REP card must be placed after the TXT cards in the object module that it modifies. Its format is shown in Figure M-4.

Columns	Contents
1	Multiple punch (12-2-9). Identifies this as a loader card.
2-4	REP -- Replace text card.
5-6	Blank.
7-12	Assembled address of the first byte to be replaced (hexadecimal). Must be right justified with leading zeros if needed to fill the field.
13	Blank.
14-16	External symbol identification number (ESID) of the control section (SD) containing the text (hexadecimal). Must be right justified with leading zeros if needed to fill the field.
17-70	From one to eleven 4-digit hexadecimal fields separated by commas, each replacing two bytes. A blank indicates the end of information in this card.
71-72	Blank.
73-80	May be used for program identification.

Figure M-4. Format of the REP card.

APPENDIX N: DIAGNOSTIC ERROR MESSAGES

Diagnostic error messages are printed following the cross reference listing, in statement number order. The message code has the form IJQnnn for the D assembler and IJYnnn for the F assembler. Figure N-1 lists the diagnostic messages and their message codes. If errors are encountered while editing library macros, the statement number referenced will be that of the "END" statement.

Message code	Message	Meaning
IJQ001 IJY	DUPLICATION FACTOR ERROR	Duplication factor: a. is zero in a literal. b. is not a positive absolute expression.
IJQ002 IJY	RELOCATABLE DUPLICATION FACTOR	Duplication factor is relocatable.
IJQ003 IJY	LENGTH ERROR	1. Out of permissible range. 2. Invalid specification.
IJQ004 IJY	RELOCATABLE LENGTH	Length is relocatable.
IJQ005 IJY	S-TYPE CONSTANT IN LITERAL	S-type constant in literal.
IJQ006 IJY	INVALID ORIGIN	Location counter has been reset to a value less than the starting address of the control section.
IJQ007 IJY	LOCATION COUNTER ERROR	Location Counter has exceeded $2^{24}-1$.
IJQ008 IJY	INVALID DISPLACEMENT	Displacement in an explicit address is not within 0-4095.
IJQ009 IJY	MISSING OPERAND	Operand is missing.
IJQ010 IJY	INCORRECT REGISTER SPECIFICATION	1. Specification of register is greater than 15. 2. Odd register is specified where an even register is required.
IJQ011 IJY	SCALE MODIFIER ERROR	Scale modifier is: a. too large. b. not an absolute expression.
IJQ012 IJY	RELOCATABLE SCALE MODIFIER	Scale modifier is not allowed to be relocatable.
IJQ013 IJY	EXPONENT MODIFIER ERROR	Exponent is: a. out of range. b. not specified as an absolute expression.

Figure N-1. Assembler Diagnostic Error Messages (Part 1 of 14)

Message Code	Message	Meaning
IJQ014 IJY	RELOCATABLE EXPONENT MODIFIER	Exponent modifier is not allowed to be relocatable.
IJQ015 IJY	INVALID LITERAL USAGE	A valid literal is used illegally, e.g., specifies a receiving field or a register.
IJQ016 IJY	INVALID NAME	Name entry incorrectly specified -- a. contains more than 8 characters. b. does not begin with a letter. c. has a special character embedded.
IJQ017 IJY	DATA ITEM TOO LARGE	The constant is too large for: a. the data type. b. the explicit length.
IJQ018 IJY	INVALID SYMBOL	The symbol specification is invalid, e.g., longer than 8 characters, embedded special character.
IJQ019 IJY	EXTERNAL NAME ERROR	1. Identical name entry in a CSECT and a DSECT statement. 2. Identical operands in one or more EXTRN/WXTRN statements.
IJQ020 IJY	INVALID IMMEDIATE FIELD	1. The immediate field is not an absolute expression. 2. The value of the immediate field is not in the range 0-255.
IJQ021 IJY	SYMBOL NOT PREVIOUSLY DEFINED	A statement requiring predefined symbols contains a symbol not predefined.
IJQ022 IJY	ESD TABLE OVERFLOW	The total number of control sections, dummy sections, and unique symbols in EXTRN statements and V-type constants exceeds 255.
IJQ023 IJY	PREVIOUSLY DEFINED NAME	The symbol in the name entry has appeared in the name entry of a previous statement.

Figure N-1. Assembler Diagnostic Error Messages (Part 2 of 14)

Message Code	Message	Meaning
IJQ024 IJY	UNDEFINED SYMBOL	A symbol being referenced has not been defined in the program.
IJQ025 IJY	RELOCATABILITY ERROR	1. A relocatable or complex relocatable expression is specified where an absolute expression is required.
IJQ026 IJY	TOO MANY LEVELS OF PARENTHESES	Expression specifies more than 5 levels of parentheses.
IJQ027 IJY	TOO MANY TERMS	More than 16 terms specified in an expression.
IJQ028 IJY	REGISTER NOT USED	A register specified in a DROP statement is not currently in use.
IJQ029 IJY	CCW ERROR	Bits 37-39 of the Channel Command Word are set to nonzero.
IJQ030 IJY	INVALID CNOP	Invalid range
IJQ031 IJY	UNKNOWN TYPE	Incorrect type designation in a DC, DS, or literal.
IJQ032 IJY	OP-CODE NOT ALLOWED TO BE GENERATED	Operation code allowed only in source statement has been obtained through substitution of a value for a variable symbol.
IJQ033 IJY	ALIGNMENT ERROR	Referenced address is not aligned to the proper boundary for this instruction.
IJQ034 IJY	INVALID OP-CODE	Invalid operation code: a. More than eight characters. b. Operation entry not followed by a blank on same card.
IJQ035 IJY	ADDRESSABILITY ERROR	The referenced address is not within the range of a USING instruction.
IJQ036 IJY	OPERAND FIELD MUST BE BLANK	Operand found for an operation code which does not allow operands. (This message may be produced by the assembler if an operand is present in a COM, EJECT, or LTORG statement when the operation field has been created by variable symbol substitution. Operands in these statements are not used but are not in error.)
IJQ037 IJY	MNOTE STATEMENT	An MNOTE statement has been generated from a macro definition. The text and severity code of the MNOTE statement is in-line in the listing.

Figure N-1. Assembler Diagnostic Error Messages (Part 3 of 14)

Message Code	Message	Meaning
IJQ|IJY 038	ENTRY ERROR	1. More than 100 ENTRY operands in this program. 2. A symbol in the ENTRY operand: a. appears in more than one ENTRY statement. b. is undefined. c. is defined in a dummy section. d. is defined in blank common. e. is equated to a symbol defined by an EXTRN or WXTRN statement.
IJQ|IJY 039	INVALID DELIMITER	Any syntax error: 1. A symbol has other than alphameric characters. 2. A symbol begins with other than alpha characters. 3. Excessive right parenthesis. 4. Equal sign encountered in a sublist. 5. Any terminating character encountered in an unexpected place. 6. Mispunched op code causes unexpected syntax scan. 7. A missing delimiter. 8. A special character that is not a valid delimiter but is used as a delimiter. 9. A delimiter used illegally. 10. A missing operand; nothing appearing between delimiters. 11. Unpaired parenthesis. 12. An embedded blank.
IJQ|IJY 040	GENERATED RECORD TOO LONG	Record has more than 187 characters
IJQ|IJY 041	UNDECLARED VARIABLE SYMBOL	Variable symbol is not declared in a define SET symbol statement or in a macro prototype.
IJQ|IJY 042	SINGLE TERM LOGICAL EXPRESSION IS NOT A SETB SYMBOL	Single term logical expression is only valid for a SETB symbol.
IJQ|IJY 043	SET SYMBOL PREVIOUSLY DEFINED	SET symbol previously defined.
IJQ|IJY 044	SET SYMBOL USAGE INCONSISTENT WITH DECLARATION	A SET symbol has been declared as: 1. undimensioned but it is subscripted. 2. subscripted but it is undimensioned.
IJQ|IJY 045	ILLEGAL SYMBOLIC PARAMETER	Attribute requested for a variable symbol which is not a symbolic parameter
IJQ|IJY 046	AT LEAST 1 RELOCATABLE Y-TYPE CONSTANT IN ASSEMBLY	One or more relocatable Y-type constants in assembly; relocation may result in address greater than 2 bytes in length.
IJQ|IJY 047	SEQUENCE SYMBOL PREVIOUSLY DEFINED	Sequence symbol previously defined.

Figure N-1. Assembler Diagnostic Error Messages (Part 4 of 14)

Message Code	Message	Meaning
IJQ048 IJY	SYMBOLIC PARAMETER PREVIOUSLY DEFINED OR SYSTEM VARIABLE SYMBOL DECLARED AS SYMBOLIC PARAMETER	1. Symbolic parameter previously defined. 2. System variable symbol declared as a symbolic parameter.
IJQ049 IJY	VARIABLE SYMBOL MATCHES A PARAMETER	Variable symbol matches a parameter.
IJQ050 IJY	INCONSISTENT GLOBAL DECLARATIONS	A global SET variable that is defined in more than one macro definition, or in a macro definition and in the source program, is inconsistent in SET type or dimension.
IJQ051 IJY	PROGRAMMER MACRO DEFINITION PREVIOUSLY DEFINED	Programmer macro prototype operation entry is identical to a: a. machine instruction. b. assembler instruction. c. previous programmer macro prototype. This message is not produced when a programmer macro matches a system macro. The programmer macro will be assembled with no indication of the corresponding system macro.
IJQ052 IJY	NAME FIELD CONTAINS ILLEGAL SET SYMBOL	SET symbol in name entry does not correspond to SET statement type.
IJQ053 IJY	GLOBAL DICATIONARY FULL	Global dictionary is full. Assembly is terminated. See Appendix H for dictionary size limits.
IJQ054 IJY	LOCAL DICTIONARY FULL	Local dictionary is full. Assembly is terminated. See Appendix H for dictionary size limits.
IJQ056 IJY	ARITHMETIC OVERFLOW	Intermediate or final result of an arithmetic operation is less than -2^{31} or greater than $2^{31}-1$.
IJQ057 IJY	SUBSCRIPT EXCEEDS MAXIMUM DIMENSION	1. &SYSLST or symbolic parameter subscript: a. exceeds 100 (200 for F assembler). b. is negative. 2 Symbolic parameter subscript is zero. 3. SET symbol subscript exceeds dimension.
IJQ059 IJY	UNDEFINED SEQUENCE SYMBOL	Operand sequence symbol does not appear as a sequence symbol in a name field.

Figure N-1. Assembler Diagnostic Error Messages (Part 5 of 14)

Message Code	Message	Meaning
IJQ060 IJY	ILLEGAL ATTRIBUTE NOTATION	L', S', or I' requested for a parameter whose type attribute does not allow these attributes to be requested.
IJQ061 IJY	ACTR COUNTER EXCEEDED	Conditional assembly loop counter exceeded -- conditional assembly terminated.
IJQ062 IJY	GENERATED STRING GREATER THAN 127 [255] CHARACTERS	Generated string is greater than 127 characters for D assembler or 255 characters for F assembler.
IJQ063 IJY	EXPRESSION 1 OF SUBSTRING IS ZERO OR MINUS	Expression 1 of substring is not allowed to be zero or minus.
IJQ064 IJY	EXPRESSION 2 OF SUBSTRING IS ZERO OR MINUS	Expression 2 of substring is not allowed to be zero or minus.
IJQ065 IJY	INVALID OR ILLEGAL TERM IN ARITHMETIC RELATIONAL EXPRESSION	1. The parameter is not a self-defining term. 2. The value of the SETC symbol used in the arithmetic expression is not composed of decimal digits.
IJQ066 IJY	UNDEFINED OR DUPLICATE KEYWORD OPERAND	1. A keyword operand occurs more than once in a macro instruction. 2. Keyword is not defined in prototype.
IJQ067 IJY	EXPRESSION 1 OF SUBSTRING GREATER THAN LENGTH OF CHARACTER EXPRESSION	Expression 1 of the substring is not allowed to be greater than the length of the character expression to which it refers.
IJQ068 IJY	GENERATION TIME DICTIONARY AREA OVERFLOWED	See Appendix H for dictionary size limits.
IJQ069 IJY	EXPRESSION 2 OF SUBSTRING GREATER THAN 8 CHARACTERS	Expression 2 of substring is not allowed to be greater than 8.
IJQ070 IJY	FLOATING POINT CHARACTERISTIC OUT OF RANGE	Exponent too large for length of defining field, exponent modifier has caused loss of all significant digits.
IJQ071 IJY	ILLEGAL OCCURRENCE OF LCL, GBL, OR ACTR STATEMENT	Local or Global declaration, or ACTR statement is out of proper sequence.
IJQ072 IJY	ILLEGAL RANGE ON ISEQ STATEMENT	Operand of ISEQ statement has an illegal range.

Figure N-1. Assembler Diagnostic Error Messages (Part 6 of 14)

Message Code	Message	Meaning
IJQ073 IJY	ILLEGAL NAME FIELD	1. Name entry required to be blank is not blank. 2. Required name entry is missing. 3. Name entry required to be a sequence symbol is not a sequence symbol.
IJQ074 IJY	ILLEGAL STATEMENT IN COPY CODE OR SYSTEM MACRO	1. Statement encountered in COPY code is not legal in COPY code. 2. Statement encountered in system macro is not legal in system macro.
IJQ075 IJY	ILLEGAL STATEMENT OUTSIDE OF A MACRO DEFINITION	Statement allowed only in a macro definition encountered in OPEN code, e.g., period asterisk (.*), mnote statement.
IJQ076 IJY	SEQUENCE ERROR	Statement not in sequence specified by ISEQ instruction.
IJQ077 IJY	ILLEGAL CONTINUATION CARD	1. Too many continuation cards. 2. Non blanks occur between the begin and continue columns of the continuation card. 3. Card not intended as continuation was treated as such because of punch in continue column of preceding card.
IJQ078 IJY	MACRO MNEMONIC OP-CODE TABLE OVERFLOW	Macro mnemonic operation code table has an overflow. See Appendix H.
IJQ079 IJY	ILLEGAL STATEMENT IN MACRO DEFINITION	This operation is not allowed within a macro definition.
IJQ080 IJY	ILLEGAL START CARD	Statements affecting, or depending upon, the location counter have been encountered before a START statement.
IJQ081 IJY	ILLEGAL FORMAT IN GBL OR LCL STATEMENT	An operand is not a variable symbol.
IJQ082 IJY	ILLEGAL DIMENSION SPECIFICATION IN GBL OR LCL STATEMENT	Dimension is other than 1-255.
IJQ083 IJY	SET STATEMENT NAME FIELD NOT VARIABLE SYMBOL	The name entry of a SET statement must be a variable symbol.
IJQ084 IJY	ILLEGAL OPERAND FIELD FORMAT IN CONDITIONAL ASSEMBLY STATEMENT	Syntax invalid, e.g., AIF statement operand does not start with a left parenthesis or, sequence symbol missing in operand field of AIF or AGO statement.

Figure N-1. Assembler Diagnostic Error Messages (Part 7 of 14)

Message Code	Message	Meaning
IJQ085 IJY	INVALID SYNTAX IN EXPRESSION	1. Invalid delimiter. 2. Too many terms in expression. 3. Too many levels of parentheses. 4. Two operators in succession.
IJQ086 IJY	ILLEGAL USAGE OF SYSTEM VARIABLE SYMBOL	1. System variable symbol appears in: a. the name entry of a SET statement. b. a mixed-mode macro definition. c. a keyword macro definition. d. a GBL or LCL statement. 2. &SYSLIST in context other than N'&SYSLIST.
IJQ087 IJY	NO ENDING APOSTROPHE	End of card encountered before an ending apostrophe.
IJQ088 IJY	UNDEFINED OPERATION CODE	Symbol in operation code field does not correspond to a valid machine or assembler operation code or to any operation code in a macro prototype statement.
IJQ089 IJY	INVALID ATTRIBUTE NOTATION	The argument of the attribute reference must be a symbolic parameter and the statement must be within a macro definition.
IJQ090 IJY	INVALID SUBSCRIPT	Syntax error, e.g., no right parenthesis after subscript, double subscript where single subscript is required, or single subscript where double subscript is required.
IJQ091 IJY	INVALID SELF-DEFINING TERM	1. Value is too large. 2. Value is inconsistent with the data type, e.g., hex for decimal, etc.
IJQ092 IJY	INVALID FORMAT FOR VARIABLE SYMBOL	1. Variable symbol is longer than 8 characters. 2. First character after the ampersand is not alphabetic. 3. Failure to use double ampersand in TITLE card or character self-defining term.
IJQ093 IJY	UNBALANCED PARENTHESES OR EXCESSIVE LEFT PARENTHESES	End of statement or card encountered before all parenthesis levels are satisfied. May be caused by embedded blank or other unexpected terminator, or failure to have a punch in continuation column.
IJQ094 IJY	INVALID OR ILLEGAL NAME OR OPERATION IN PROTOTYPE STATEMENT	1. Name not blank or variable symbol. 2. Variable symbol in name field is subscripted. 3. Violation of rules for forming variable symbol, (must begin with ampersand (&) followed by 1-7 letters and/or numbers first of which must be a letter). 4. Statement following 'MACRO' is not a valid prototype statement.

Figure N-1 . Assembler Diagnostic Error Messages (Part 8 of 14)

Message Code	Message	Meaning
IJY095	ENTRY TABLE OVERFLOW	Number of ENTRY symbols, i.e., ENTRY instruction operands, exceeds 100.
IJQ096 IJY	MACRO INSTRUCTION OR PROTOTYPE OPERAND EXCEEDS 127 [225] CHARACTERS IN LENGTH	Macro instruction or prototype operand length exceeds 127 characters for D assembler and 255 characters for F assembler
IJQ097 IJY	INVALID FORMAT IN MACRO INSTRUCTION OPERAND OR PROTOTYPE PARAMETER	1. Illegal equal sign (=). 2. A single ampersand (&) appears somewhere in the standard value assigned to a prototype keyword parameter. 3. First character of a prototype parameter is not an ampersand. 4. Prototype parameter is a subscripted variable symbol. 5. Invalid usage of alternate format in prototype (see example) 6. Nonsense prototype parameter, e.g., &A* or &A&&. Note: Occurrence of this error will cause only syntax to be checked for the remainder of the macro definition.

Example:

Name	Operation	Operand	Continuation Column
	PROTO	&A,&B, or	
	PROTO	&A,&B, &C	X

IJQ098 IJY	EXCESSIVE NUMBER OF OPERANDS OR PARAMETERS	1. The prototype has more than 100 (200 for F assembler) parameters. 2. The macro instruction has more than 100 (200 for F assembler) operands.
IJQ099 IJY	POSITIONAL MACRO INSTRUCTION OPERAND, PROTOTYPE PARAMETER, OR EXTRA COMMA FOLLOWS KEYWORD	Positional macro instruction operand, prototype parameter, or extra comma follows keyword.
IJQ100 IJY	STATEMENT COMPLEXITY EXCEEDED	See Appendix H for statement complexity limits.
IJQ101 IJY	EOD on SYSIN or SYSIPT	End-of-data reached before an END statement was encountered.

Figure N-1. Assembler Diagnostic Error Messages (Part 9 of 14)

Message Code	Message	Meaning
IJQ102I IJY	INVALID OR ILLEGAL ICTL	1. Operands of ICTL statement are out of range. 2. ICTL is not the first statement in the input deck.
IJQ103I IJY	ILLEGAL NAME IN OPERAND FIELD OF COPY CARD	Syntax error, e.g., symbol has an illegal character or has more than 8 characters.
IJQ104I IJY	COPY CODE NOT FOUND	The operand of a COPY statement specified COPY text which cannot be found in the library.
IJQ105I IJY	EOD ON SOURCE STATEMENT LIBRARY	1. MEND statement missing from macro definition. 2. While editing a macro, COPY code not found. Macro definition truncated. 3. End-of-file encountered while reading a macro or copy code.
IJY106I	NOT NAME OF DSECT	Referenced symbol expected to be DSECT name, but it is not.
IJQ107I IJY	INVALID OPERAND	Operand unrecognizable, contains invalid value, or incorrectly specified.
IJQ108I IJY	PREMATURE EOD	Indicates a machine error or an internal assembler error.
IJQ109I IJY	PRECISION LOST	High order information lost by attempting to express constant in a field not long enough to contain it.
IJY110I	EXPRESSION VALUE TOO LARGE	Value of expression greater than -16777216 to +16777215. Expressions in EQU and ORG statements are flagged if (1) they include terms previously defined as negative values, or (2) positive terms give a result of more than three bytes in magnatude. The error indication may be erroneous due to (1) the treatment of negative values as three-byte positive values, or (2) the effect of large positive values on the location counter if a control section begins with a START statement having an operand greater than zero, or a control section is divided into subsections.
	NOTE: Messages ending with an I are printed on both SYSLST and SYSLOG unless one of the messages indicates that SYSLST or an unidentifiable unit is defective, in which case they will appear on SYSLOG only. The messages appearing on SYSLOG will be prefaced by "A" regardless of which assembler produced them. 110I and 111I errors can be detected at any point during assembly -- amount of assembly listing printed is unpredictable. 112I through 115I errors are detected immediately upon assembly attempt -- no assembly listing is printed. In either case the assembly is terminated, the source is bypassed to a /* or EOF, and control is returned to the supervisor via EOJ. The subsequent steps of a multiple step JOB are not bypassed unless they are also defective.	

Figure N-1. Assembler Diagnostic Error Messages (Part 10 of 14)

Message Code	Message	Meaning
IJQ110I IJY	ABORT--PERM I/O ERROR ON SYSxxx	Cause: An unrecoverable error on the named file prevents further processing: If the named file is SYSxxx, the unit code of the DTF that caused the error does not match any valid unit. This is usually the result of an accidental overlap that destroys the DTF. This is probably a hardware error. System Action: The job step is terminated. Programmer Action: Rerun the job using another disk pack or tape reel, or use another unit for the disk pack or tape reel. If the problem recurs, do the following to complete your problem determination action: 1. Execute the ROD command and EREP, and retain the output. 2. Have the job stream and system log available. Operator Action: Execute the LISTIO command for SYSxxx to determine the physical unit to which it is assigned. Move the disk pack or tape reel to another physical device and reassign SYSxxx to that unit, or mount another disk pack or tape reel and rerun the job.
IJQ111I IJY	ABORT--UNEXPECTED EOF ON SYSxxx	Cause: EOF has occurred on an assembler work file that does not support multi-volume files. It usually results from a short tape, or a tape indicate reflective marker was read. This is probably a user error. System Action: The job step is terminated. Programmer Action: If the problem recurs, have the system log, printer output, and the job stream available to complete your problem determination action. Operator Action: If SYSxxx is assigned to a tape, mount a longer tape, or use a 1600 BPI tape drive instead of an 800 BPI drive, or reassign the work files to disk and rerun the job, or if SYSxxx is assigned to a disk, submit larger extents and rerun the job.

•Figure N-1. Assembler Diagnostic Error Messages (Part 11 of 14)

Message Code	Message	Meaning
IJQ 112I IJY	ABORT--INADEQUATE CORE FOR 32L [44K] ASSEMBLER	Cause: An attempt was made to execute the [32K] D assembler in less than 14K, or the F assembler in less than 44K. This is probably a user error. System Action: The job step is terminated. Programmer Action: If there is insufficient main storage available, you must link edit a smaller assembler. If the problem recurs, do the following to complete your problem determination action: 1. Execute the MAP command and retain the output. 2. Have the printer output available. Operator Action: Execute the MAP command to determine the partition size. Then allocate a larger partition for the assembly.
IJQ 113I	ABORT--INVALID PHYSICAL UNIT FOR SYSxxx	Cause: The assignments for a work file(s) are not valid: • The device type is not valid, or the assembler is link edited for different devices than those assigned. • The UA (unassign) or IGN (ignore) option was specified for the D assembler. • The specified mode setting is not valid. • For the D assembler, the work file device types are not consistent. (SYS003 is correct.) Only the first invalid unit is named in the message. This is probably a user error. System Action: The job step is terminated. Programmer Action: Use the LISTIO output to determine the cause for the message. Use CSERV to display the phase named "ASSEMBLY" and check byte X'1C', bits 5, 6, and 7 for the device type specified at link edit time as work files. Bit 5: 1=2400 Bit 6: 1=2314 Bit 7: 1=2311 Correct the assignments and resubmit the job. If the problem recurs, do the following to complete your problem determination action: 1. Have the LISTIO and CSREV output available. 2. Have the job stream and system output available. Operator Action: Issue the LISTIO command to check the assignments and enter the correct work file assignments if possible.

•Figure N-1. Assembler Diagnostic Error Messages (Part 12 of 14)

Message Code	Message	Meaning
IJQ 114I	ABORT--NO UNIT ASSIGNED FOR SYSPCH (for D assembler)	Cause: For the D assembler, the OPTION [DECK] is in effect and SYSPCH is not assigned. This is probably a user error. System Action: The job step is terminated. Programmer Action: Submit an assign for SYSPCH, or specify OPTION [NODECK] and resubmit the job. If the problem recurs, do the following to complete your problem determination action: 1. Retain the LISTIO listing. 2. Have the job stream, program listing, and system log available. Operator Action: Execute the LISTIO command and verify assignments. Submit an assign for SYSPCH and rerun the job.
IJY 114I	ABORT--NO UNIT ASSIGNED FOR SYSxxx (OPTION SYM) (for F assembler)	Cause: For the F assembler, a required unit (SYS001-SYS003 or a device required by an OPTION statement) is unassigned, or the IGN option is specified for the device. The IGN (ignore) option is valid for SYSPCH and SYSLST. This is probably a user error. System Action: The system terminates the job step. Programmer Action: Submit an assignment for the indicated logical unit, or correct the OPTION statement to eliminate the requirement and resubmit the job. If the problem recurs, have the LISTIO listing, the system log, the job stream, and the printer output available to complete your problem determination action. Operator Action: Execute the LISTIO command and verify the assignments. Submit an assign for the indicated logical unit and rerun the job.
IJY 115I	ABORT--INVALID DUAL ASSGN SYSPCH-SYSIPT [SYSLST]	Cause: SYSPCH and SYSIPT are both assigned to the same unit, which is not a 1442N1 or 2520B1 card reader, or SYSPCH and SYSLST are both assigned to the same unit, which is not a disk.

● Figure N-1. Assembler Diagnostic Error Messages (Part 13 of 14)

Message Code	Message	Meaning
		This is probably a user error. System Action: The job step is terminated. Programmer Action: Check the LISTIO listing to determine the dual assignments. Reassign the indicated logical units to separate devices, or the required device type. If the problem recurs, retain the LISTIO output, the job stream, system log, and supervisor listing to complete your problem determination action. Operator Action: Execute LISTIO to determine the current assignments. Reassign the two indicated logical units to separate devices or to the required device type.

• Figure N-1. Assembler Diagnostic Error Messages (Part 14 of 14)

APPENDIX O: SELF-RELOCATING PROGRAM TECHNIQUES

Self-relocating programs are executed in a multiprogramming environment and at any location in main storage. These programs may be located in either foreground area of main storage. A program that is self-relocating must initialize its address constants, including Channel Command Words (CCWs), at execution time. The user must code his own self-relocating routine for execution after it is linkage edited and loaded into main storage.

When coding a self-relocating program, the programmer should take these points into consideration:

1. All A-type address constants must be relocated.
2. The I/O area addresses in all CCWs must be relocated.
3. Address constants generated by Physical IOCS macros (EXCP, WAIT, etc.) must be relocated.
4. Logical IOCS macros can be self-relocated using the OPENR macro.

The following example program shows how a user may code a self-relocating program. This example uses the A-type constant and registers 1 and 2 although the user may use any of the other available registers if he chooses.

This program contains six address constants. Two are A-type and two each are contained in the Command Control Block (CCB) and the Channel Command Word (CCW) macros. This procedure is used:

1. The absolute addresses of the contents of the two A-type constants (EOFTAPE and CHA12) and the CCW for each CCB (PRINTCCW and TAPECCW) are loaded into a work register (Register 1).
2. The work register is stored in the address constants [A(EOFTAPE) and A(CHA12)] and in their respective CCBs (PRINTCCB+8 and TAPECCB+8).
3. The command code for the CCWs shares a full word with the I/O area address and must be reset after the I/O area address has been stored. This is done here by two methods: (a) saving the command code for the PRINTCCW in Register 2 and then restoring it; (b) using the Move Immediate (MVI) instruction for the TAPECCW to set the command code.

In the main routine of this program, note that register notation has been used with the EXCP and WAIT macros to avoid the generation of address constants by the macros themselves. The example of a self-relocating program follows:

```
  SOURCE STATEMENT

          PRINT NOGEN
PROGRAM   START 0
          BALR  15,0
          USING *,15
*     ROUTINE TO RELOCATE ADDRESS CONSTANTS
          LA    1,PRINTCCW                RELOCATE CCW ADDRESS
          ST    1,PRINTCCB+8               IN CCB FOR PRINTER
          LA    1,TAPECCW                 RELOCATE CCW ADDRESS
          ST    1,TAPECCB+8                IN CCB FOR INPUT TAPE
          LA    1,EOFTAPE                *RELOCATE*****
          ST    1,AEOFTAPE               * PROGRAM    *
          LA    1,CHA12                  *  ADDRESS   *
          ST    1,ACHA12                 ****CONSTANTS*
          IC    2,PRINTCCW                SAVE PRINT CCW OP CODE
          LA    1,OUTAREA                 RELOCATE OUTPUT AREA ADDRESS
          ST    1,PRINTCCW                 IN PRINTER CCW
          STC   2,PRINTCCW                RESTORE PRINT CCW OP CODE
          LA    1,INAREA                  RELOCATE INPUT AREA ADDRESS
          ST    1,TAPECCW                  IN TAPE CCW
          MVI   TAPECCW,2                 SET TAPE CCW OP CODE TO READ
*     MAIN ROUTINE...READ TAPE AND PRINT RECORDS
READTAPE  LA    1,TAPECCB                 GET CCB ADDRESS
          EXCP  (1)                       READ ONE RECORD FROM TAPE
          WAIT  (1)                       WAIT FOR COMPL. OF I/O
          L     10,AEOFTAPE               GET ADDRESS OF TAPE EOF ROUTINE
          BAL   14,CHECK                  GO TO UNIT EXCEPTION SUBROUTINE
          MVC   OUTAREA(10),INAREA        EDIT RECORD
          MVC   OUTAREA+15(70),INAREA+10   IN
          MVC   OUTAREA+90(20),INAREA+80   OUTPUT AREA
          LA    1,PRINTCCB                GET CCB ADDRESS
          EXCP  (1)                       PRINT EDITED RECORD
          WAIT  (1)                       WAIT FOR COMPL. OF I/O
          L     10,ACHA12                 GET ADDRESS OF CHAN 12 ROUTINE
          BAL   14,CHECK                  GO TO UNIT EXCEPTION SUBROUTINE
          B     READTAPE
CHECK     TM    4(1),1                    CHECK FOR UNIT EXC. IN CCB
          BCR   1,10                       YES-GO TO PROPER ROUTINE
          BR    14                         NO-RETURN TO MAINLINE
CHA12     MVI   PRINTCCW,X'8B'            SET SK TO CHAN 1 OP CODE
          EXCP  (1)                       SK TO CHAN 1 IMMEDIATELY
          WAIT  (1)                       WAIT FOR COMPL. OF I/O
          MVI   PRINTCCW,9                SET PRINTER OP CODE TO WRITE
          BR    14                        RETURN TO MAINLINE
EOFTAPE   EOJ                             END OF JOB
          CNOP  0,4                       ALIGN CCB'S TO FULL WORD
PRINTCCB  CCB   SYS004,PRINTCCW,X'0400'
TAPECCB   CCB   SYS001,TAPECCW
PRINTCCW  CCW   9,OUTAREA,X'20',110
TAPECCW   CCW   2,INAREA,X'20',100
AEOFTAPE  DC    A(EOFTAPE)
ACHA12    DC    A(CHA12)
OUTAREA   DC    CL110' '
INAREA    DC    CL100' '
          END   PROGRAM
```

The macro definitions in this appendix are typical applications of the macro language and conditional assembly. Another macro definition is included as part of Appendix I. The definitions are presented along with statements generated from typical corresponding macro instructions.

The first macro definition is NOTE -- a DOS system macro taken from the source statement library of the DOS assembler.

The second macro definition is MOVE. This macro is recursive; i.e., it calls itself as an inner macro. Compare this macro definition with MOVE in Appendix I. MOVE in Appendix I has more statements, however it functions differently and includes error checking facilities.

```
         MACRO
&LABEL   NOTE  &FILEN
.* IBM SYSTEM/360 TAPE/DISK OPERATING SYSTEM
* CHANGE LEVEL 2-0
         AIF   (T'&FILEN NE 'O').ONE
         MNOTE 0,'NO FILENAME SPECIFIED.SET TO ''*'''
&LABEL   L     1,=A(*)              *****ERROR-PATCH DTF TABLE ADDRESS
         AGO   .THREE
.ONE     AIF   ('&FILEN'(1,1) NE '(').TWO
         AIF   ('&FILEN(1)' EQ '1').FOUR
&LABEL   LR    1,&FILEN(1)                GET DTF TABLE ADDRESS
         AGO   .THREE
.TWO     ANOP
&LABEL   L     1,=A(&FILEN)               GET DTF TABLE ADDRESS
.THREE   L     15,16(1)                   GET LOGIC MODULE ADDRESS
         AGO   .FIVE
.FOUR    ANOP
&LABEL   L     15,16(1)                   GET LOGIC MODULE ADDRESS
.FIVE    BAL   14,12(15)                  BRANCH TO NOTE ROUTINE
         MEND


 *
 *              STATEMENTS GENERATED FROM NOTE MACRO INSTRUCTIONS
 *


 *                        SYMBOL AS OPERAND

 NNAME    NOTE  INFILE
+** CHANGE LEVEL 2-0
+NNAME    L     1,=A(INFILE) GET DTF TABLE ADDRESS
+         L     15,16(1) GET LOGIC MODULE ADDRESS
+         BAL   14,12(15) BRANCH TO NOTE ROUTINE


 *                        REGISTER 1 AS OPERAND

         NOTE  (1)
+** CHANGE LEVEL 2-0
+         L     15,16(1) GET LOGIC MODULE ADDRESS
+         BAL   14,12(15) BRANCH TO NOTE ROUTINE


 *                        OTHER REGISTER AS OPERAND

         NOTE  (5)
+** CHANGE LEVEL 2-0
+         LR    1,5 GET DTF TABLE ADDRESS
+         L     15,16(1) GET LOGIC MODULE ADDRESS
+         BAL   14,12(15) BRANCH TO NOTE ROUTINE
```

```
*                       OMITTED OPERAND

          NOTE
++ CHANGE LEVEL 2-0
                0,NO FILENAME SPECIFIED.SET TO '*'
+         L     1,=A(*) *****ERROR-PATCH DTF TABLE ADDRESS
+         L     15,16(1) GET LOGIC MODULE ADDRESS
+         BAL   14,12(15) BRANCH TO NOTE ROUTINE



          MACRO
 &NAME    MOVE  &FROM,&TO,&COUNT
 .*
 .*                RECURSIVE GENERAL PURPOSE MOVE MACRO
 .*
          GBLA  &A
          LCLA  &B
 &B       SETA  &COUNT
 .PN      AIF   (&B LE 256).DO          MOVE LESS THAN 256 BYTES
 &NAME    MVC   &TO+&A.(256),&FROM+&A   256 BYTE MOVE
 &A       SETA  &A+256
 &B       SETA  &B-256
          MOVE       &FROM,&TO,&B       CALL THYSELF
          MEXIT
 .DO      ANOP
 &NAME    MVC   &TO+&A.(&B),&FROM+&A    LESS THAN 256 BYTE MOVE
 &A       SETA  0                        SET BACK TO 0 FOR NEXT CALL
          MEND


 *
 *              STATEMENTS GENERATED FROM MOVE MACRO INSTRUCTIONS
 *


 *                      MOVE LESS THAN 256 BYTES

 MNAME    MOVE FRMAD,TOAD,150
+MNAME    MVC   TOAD+0(150),FRMAD+0 LESS THAN 256 BYTE MOVE


 *                      MOVE 256 BYTES

          MOVE  FRMAD,TOAD,256
+         MVC   TOAD+0(256),FRMAD+0 LESS THAN 256 BYTE MOVE


 *                      MOVE MORE THAN 256 BYTES

 MNAME2   MOVE  FRMAD,TOAD,400
+MNAME2   MVC   TOAD+0(256),FRMAD+0 256 BYTE MOVE
+         MVC   TOAD+256(144),FRMAD+256 LESS THAN 256 BYTE MOVE


 *                      MOVE MORE THAN 512 BYTES

          MOVE  FRMAD,TOAD,520
+         MVC   TOAD+0(256),FRMAD+0 256 BYTE MOVE
+         MVC   TOAD+256(256),FRMAD+256 256 BYTE MOVE
+         MVC   TOAD+512(8),FRMAD+512 LESS THAN 256 BYTE MOVE
```

Indexes to systems reference library manuals are consolidated in IBM System/360 Disk Operating System Master Index (GC24-5036) and in IBM System/360 Tape Operating System Programming Index (GC24-5064). For additional information about any subject listed below, refer to other publications listed for the same subject in the consolidated index.

&SYS, restrictions on use 63, 75, 88
&SYSECT (see Current control section name)
&SYSLIST (see macro instruction operand)
&SYSNDX (see macro instruction index)
7090/7094 Support Package Assembler 8, 140
Absolute terms 15
ACTR instruction 84
Address constants 47
  A-type 47
  Complex relocatable expressions 47
  Literals not allowed 20
  S-type 48
  V-type 48
  Y-type 47
Address specification 34
Addressing 24
  Dummy sections 29
  Explicit 24
  External control sections, 32
  Implied 24
  Relative 26
AGO instruction 83
  Example 83
  Form of 83
  Inside macro definitions 83
  Operand field of 83
  Outside macro definitions 83
  Sequence symbol in 83
  Use of 83
AIF instruction 82
  Example of 82
  Form of 82
  Inside macro definitions 82
  Invalid operand fields of 83
  Logical expression in 82
  Operand field of 82
  Outside macro definitions 82
  Sequence symbols in 82
  Use of 82
  Valid operand fields of 83
Alignment, boundary
  CNOP instruction for 55
  Machine instruction 33
Ampersands in
  Character expressions 79
  Macro-instruction operands 66
  MNOTE instruction 87
  Symbolic parameters 63
  Variable symbols 59
ANOP instruction 84
  Example of 84
  Form of 84
  Sequence symbol in 84
  Use of 84
Apostrophes in
  Character expressions, 78
  Macro instruction operands 66
Apostrophes (continued)
  MNOTE instruction 87
Arithmetic expressions
  Arithmetic relations 81
  Evaluation procedure 76
  Invalid examples of 76
  Operand sublists 77
  Operators allowed 75
  Parenthesized terms in 76
    evaluation of 76
    examples of 76
  SETA instruction 75
  SETB instruction 80
  Substring notation 78
  Terms allowed 75
  Valid examples of 75
Arithmetic relations 81
Arithmetic variable 91
Assembler instructions
  Statement 38
  Table 117
  (see specific instructions)
Assembler language 8
  Basic Programming Support 8, 140
  Coding conventions 11
  Comparison chart 140
  Macro facilities relation to 58
  Statement format 13
  Structure 15, 16
Assembler program
  Basic functions 9
  DOS/TOS relationship 10
  Listing 131
  Output 27, 151
  Variants 149
Assembler relationships 10
Assembling a Program 143
  Assemble-and-execute 8, 145, 148
  Card Input 143, 145, 148
  Device Assignments 144
  Diagnostic Error Messages 152
  I/O Units Used 146, 147
  Operating Considerations 144
  Output 150
Assembly, terminating an 57
Assembly no operation (see ANOP instruction)
Attributes 71
  How referred to 72
  Inner macro instruction operands 71
  Kinds of 71
  Notations 71
  Operand sublists 71
  Outer macro instruction operands 71
  Summary chart of 124

Attributes (continued)
Use of 71
(see also specific attributes)
Basic Programming Support Assembler 8, 140
Base registers
Address calculation 10, 32, 34
DROP instructions 24
Loading of 24
USING instructions 24
Binary constant 44
Binary self-defining term 18
Binary variable 91
Bit length specification 40.1
Multiple constants 40.1
Multiple operands 41
Duplicated constants 41
Blanks
Logical expressions 80
Macro instruction operands 67

CCW instruction 50
Channel command word, defining 50
Character codes 98
Character constant 42
Character expressions 78
Ampersands in 78
Character relations 81
Concatenating 79
Examples of 78
Periods and 78
Apostrophes in 78
SETB instructions 80
SETC instructions 77
Character relations 81
Character self-defining term 19
Character set 14, 15, 98
Character variable 91
CNOP instruction 55
Coding form 12
COM instruction 30
Commas, macro instruction operands 67
Comments entries 13
Comments statements
Examples of 14, 65
Model statements 65
Not generated 65
Comparison chart 140
Compatibility
Assembler language 8
Macro-definitions 97
Complex relocatable expressions 47
Concatenation
Character expressions 78, 79
Defined 64
Examples of, 64
Substring notations 78
Conditional assembly elements, summary charts of 85, 123
Conditional assembly instructions
How to write 70
Summary of 85
Use of 70
(see also specific instructions)
Conditional branch (see AIF instruction)
Constants (see also specific types)
Defining (see DC instructions)
Summary of 120
Continuation lines 11
Conditional branch instruction 36
Operand format 37
Control section location assignment 28
Control sections
Blank common 30
CSECT instruction 28
Defined 27
DSECT instruction 29
First control section, properties of 28
START instruction 28
Unnamed 29
COPY instruction 56
COPY statements in macro definitions
Form of 65
Model statements, contrasted 65
Operand field of 65
Use of 65
Count attribute
Defined 73
Notation 71
Operand sublists 73
Use of 73
Variable symbols 73
CSECT instruction, symbol in, length attribute of 28
Current control section name (&SYSECT)
Affected by CSECT, DSECT, START 93
Example of 93
Use of 93
Data definition instructions 39
Channel command words 50
Constants 39
Storages 48
Data representation 10
DC instruction 39
Duplication factor operand subfield 40
Operand subfield modifiers 40
Type operand subfield 40
Length modifier 40
Scale modifier 41
Exponent modifier 42
Constant operand subfield 42
Address-constants (see address constants)
Binary constant 44
Character constant 42
Decimal-constants 46
Fixed-point constants, 44
Floating-point constants. 45
Hexadecimal constant 43
Type codes for 41
Decimal constants 46
Length modifier 46
Length, maximum 46
Packed 45
Zoned 45
Decimal field, integer attribute of 74
Decimal self-defining terms 78
Defining constants (see DC instruction)
Defining storage (see DC instruction; DS instruction)
Defining symbols 17, 70
Diagnostic Error Messages 152
Dimension, subscripted SET symbols 91
Displacements 34
Double-shift instruction 33
DROP instruction 25, 33

DS instruction 48
Defining areas 49
Forcing alignment 49
DSECT instruction 29
Dummy section location assignment 29, 31
Duplication factor 40
Forcing alignment 49

Effective address length 35
EJECT instruction 51
END instruction 57
ENTRY instruction 31
Entry point symbol, identification of 31
EQU instruction 38
Equal signs, as macro instruction operands 66
Error message (see MNOTE instruction)
Error Messages 152
After END statement 57
Explicit addressing 24, 34
Length 34
Exponent modifiers 42
Expressions 21, 31
Absolute 34
Character 78
Evaluation 22
Logical 81
Relocatable 34
Summary chart of 124
Extended mnemonic codes 36
Operand format 37
Table 108
External control section, addressing of 31
External symbol, identification of 31
EXTRN instruction 31

First control section 28
Fixed-point constants 44
Format 44
Positioning of 44
Scaling 44
Values minimum and maximum 44, 45
Fixed-point field, integer attribute of 74
Floating-point constants 45
Alignment 46
Format 45
Scale modifiers 45
Floating-point field, integer attribute of 74
Format control, input 53

GBLA instruction
Form of 88
Inside macro definitions 88
Outside, macro definitions 88
Use of 88
GBLB instruction
Form of 88
Inside macro definitions 88
Outside macro definitions 88
Use of 88
GBLC instruction
Form of 88
GBLC instruction (continued)
Inside macro definitions 88
Outside, macro definitions 88
Use of 88
General register zero, base register usage 25
Generated statements, examples of 64
Global SET symbols
Defining 88
Examples of 88, 90
Local SET symbols, compared 87
Using 88
Global variable symbols
Types of 87
(see also global SET symbols, subscripted SET symbols)

Hexadecimal constants 43
Hexadecimal-decimal conversion chart 98
Hexadecimal self-defining terms 18

I' (see Integer attribute)
ICTL instruction 52
Identification-sequence field 14
Identifying blank common control section 30
Identifying assembly output 51
Identify dummy section 29
Implied addressing 24, 34
Length 34
Implied length specification 34
Inner macro instruction
Defined 68
Example of 69
Symbolic parameters in 69
Instruction alignment 33
Instruction format 14
Integer attributed
Decimal fields 74
Examples of 74
Fixed-point fields 73
Floating-point fields 74
How to compute 74
Notation 71
Restrictions on use 74
Use of 74
ISEQ instruction 53

K' (see Count attribute)
Keyword
Defined 94
Keyword macro instruction 94
Symbolic parameter and 93.1
Keyword, inner macro instructions used in 95
Keyword macro definition
Positional macro definitions compared 93.1
Use 94
Keyword macro instruction
Example of 95
Format of 94
Keywords in 94
Operands 58, 94
Invalid examples 95
Valid examples 95

Keyword macro instruction (continued)
Operand sublists in 95
Keyword prototype statement
Examples of 94
Format of 94
Operands 94
Invalid examples 94
Valid examples 94
Standard values 94

L' (see Length attribute)
LCLA instruction
Form of 75
Use of 75
LCLB instruction
Form of 75
Use of 75
LCLC instruction
Form of 75
Use of 75
Length modifier 40
Bit length specification 40.1
Lengths explicit and implied 34, 35
Length attribute
Defined 34, 72
Examples 73
Notation 71
Restrictions on use 73
Symbols 17, 73
Use of 73
Length modifier 40
Length subfield 33
Level of parentheses 21
Library, copying coding form 56
Linkage symbols (see also ENTRY instruction; EXTRN instruction)
Entry point symbol 31
External symbol 31
Linkage editor and use of 31
Listing, spacing 52
Listing control instructions 51
Literal pools 20, 54, 55
Segments one to four 55
Literals 20
Character 34
DC instruction, used in 20
Duplicate 21
Format 20
Literal pool, beginning 55
Literal pools, multiple 20
Treatment of self-defining term 20
Local SET symbols
Defining 88
Examples of 88-90
Global SET symbols, compared 87
Using 88
Local variable symbols
Types of 87
(see also local SET symbols)
(see also subscripted SET symbols
Location counter 38, 42, 47
Predefined symbols 19
References to 19
Setting 54
Logical expressions
AIF instructions 82

Logical expressions (continued)
Arithmetic relations 81
Blanks in 81
Character relations 81
Evaluation of 81
Invalid examples of 81
Logical operators in 81
Parenthesized terms in
Evaluation of 81
Examples of 81
Relation operators in 81
SETB instructions 80
Terms allowed in 81
Valid examples of 81
LTORG instruction 55
Machine features required 7
Machine instructions 33
Alignment and checking 33
Length 34
Literals, limits on 20
Mnemonic operation codes 35
Operand fields and subfields 33
Symbolic operand formats 35
Machine-instruction mnemonic codes 35
Alphabetical listing 108
MACRO
Form of 61
Use 61
Macro definition
Compatibiltiy 97
Defined 61
Example of 63
How to prepare 61
Keyword (see Keyword macro definition)
Mixed-mode (see Mixed-mode macro definition)
Placement in source program 61
Sample 133, 165, 166
Use 61
Macro definition exit (see MEXIT instruction)
Machine instruction examples and format
RR 33, 35
RX 33, 36
RS 33, 36
SI 33, 36
SS 33, 36
Summary table 106
Macro definition header statement (see MACRO)
Macro definition trailer statement (see MEND)
Macro facility
Additional features 86
Comparison chart 142
Relation to assembler language 58
Summary 85, 121
Macro instruction
Defined 58
Example of 67
Form of 66
How to write 66
Levels of 69
Inner 68
Mnemonic operation code 66
Name entry of 66
Omitted operands 67
Example of 67

Macro instruction (continued)
Operand entry of 66
Operands
Ampersands 66
Blanks 67
Commas 67
Equal signs 66
Paired parentheses 66
Paired apostrophes 66
Operand sublists 67
Operation entry of 66
Outer 68
Statement form 67
Types of 58
Used as model statement 68
Macro instruction index (&SYSNDX)
AIF instruction 91
Arithmetic expressions 91
Character relation 91
Example 92
MNOTE instruction 91
SETB instruction 91
SETC instruction 91
Use of 91
Macro instruction operand (&SYSLIST)
Attributes of 93
Use of 93, 96
(see also symbolic parameters)
Macro instruction prototype statement (see prototype statement)
Macro instruction statement (see macro instruction)
MEND
Form of 61
MEXIT instruction, contrasted 86
Use of 61
MEXIT instruction
Example of 86
Form of 86
MEND, contrasted 86
Use of 86
Mixed-mode macro definitions
Positional macro definitions contrasted 96
Use 96
Mixed-mode macro instruction
Example of 96
Form of 96
Operand field of 58, 96
Mixed-mode prototype statement
Example of 96
Form of 96
Operands of 96
Mnemonic operation codes 35
Extended 37
Machine instruction 35
Macro instruction 61
MNOTE instruction
Ampersands in 86
Error message 86
Example of 86
Operand entry of 86
Apostrophes in 86
Severity code 86
Use of 86
Model statements
Comments field of 63
Comments statements 65
Model statements (continued)
Defined 62
Name field of 62
Operation field of 62
Operand field of 63
Use of 62
N' (see Number attribute)
Name entries 13
Number attribute
Defined 73
Notation 73
Operand sublist 73

Object deck 150
Object module 27
Operands
Entries 13
Fields 33
Subfields 33, 34
Symbolic 31, 33, 35
Operand Sublist
Alternate statement form 67
Defined 67
Example of 68
Use of 67
Operation codes
Assembler 9
Machine 9
Operations entries 13
Operation field 33
Ordinary symbol 17
ORG instruction 54
Outer macro instruction defined 68
Paired parentheses 66
Paired apostrophes 66
Parentheses in
Arithmetic expressions 76
Logical expressions 81
Macro instruction operands 66
Operand fields and subfields 34
Paired 66
Period in
Character expressions 78
Comments statements 65
Concatenation 65
Sequence symbols 74
Positional macro definition (see macro definition)
Positional macro instruction (see macro instruction) 58
Previously defined symbols 18
PRINT instruction 52
Program control instructions 53
Program listings 10
Program sectioning and linking 16, 24, 27
Prototype statement
Alternate 62
Example of 62
Form of 61
Keyword (see keyword prototype statement)
Mixed-mode (see mixed-mode prototype statement)
Name entry of 61
Operand entry of 61
Operation entry of 61
Statement form 61

Prototype statement (continued)
Symbolic parameters in 61
Use of 61
PUNCH instruction 54

Relational operators 81
Relative addressing 26
Relocatability 10, 15
Attributes 31
Program, general register zero 25
Relocatable expressions 23, 33
In USING instructions 25
Relocatable symbols 17
Relocatable terms
Pairing of 22
In relocatable expressions 23
REP card 151.2
REPRO instruction 54
RR machine instruction format 33
Length attribute 33
Symbolic operands 35
RS machine instruction format 33
Address specification 34
Length attribute 33
Symbolic operands 35
RX machine instruction format 33
Address specification 34
Length attribute 33
Symbolic operands 35

S' (see scaling attribute)
Sample macro definitions 133, 165, 166
Sample program 131
Scale modifier
Fixed-point constants 41
Floating-point constants 41
Scaling attribute
Decimal fields 73
Defined 73
Fixed-point fields 73
Floating-point fields 73
Notation 89
Restrictions on use 73
Symbols 73
Use of 73
Self-defining terms 18, 20
Binary 18
Character 19
Decimal 18
Hexadecimal 18
Using 18
(see also specific terms)
Sequence checking 53
Sequence symbols 17, 74
AGO instruction 83
AIF instruction 82
ANOP instruction 84
How to write 73
Invalid examples of 74
Macro instruction 74
Use of 74
Valid examples of 74
Set symbols
Assigning values to 70
Defining 70
Symbolic parameters, contrasted 70
Use 70

Set symbols (continued)
(see also local SET symbols)
(see also global SET symbols)
(see also subscripted SET symbols)
SET variable 90
SETA instruction
Examples of 76, 77
Form of 75
Operand entry of 75
Evaluation procedure 76
Operators allowed 75
Parenthesized terms 76
Terms allowed 75
Valid examples of 75
Operand sublist 77
Example 77
SETB instruction
Example of 82
Form of 80
Logical expression in 81
Arithmetic relations 81
Blanks in 81
Character relations 81
Evaluation of 81
Operators allowed 81
Operand entry of 80
Invalid examples of 81
Valid examples of 81
SETC instruction
Apostrophes 78
Character expressions in 78
Ampersands 78
Periods 78
Concatenation in
Character expressions 79
Substring notations 79
Examples of 77-80
Form of 77
Operand entry of 77
Substring notations in 78
Arithmetic expressions in 79
Character expressions in 79
Invalid examples of 79
Valid examples of 79
Type attribute in 77
Example of 78
SETA symbol
Assigning values to 70
Defining 70
SETA instruction 76
Using 76
SETB symbol
AIF instruction 82
Assigning values to 70
Defining 70
SETA instruction 82
SETB instruction 82
SETC instruction 82
Using 82
SETC symbol
Assigning values to 70
Defining 70
SETA instruction 81
Using 80
Severity Code 144
Severity code in MNOTE instruction 87
SI machine instruction format 39
Address specification 34

SI machine instruction format (continued)
Length attribute 33
Symbolic operands 35
Source module 27
Source statement library defined 59
SPACE instruction 52
SS machine instruction format 33
Address specification 34
Length attribute 33
Length field 34
Symbolic operands 35
START instruction
Positioning of 27
Unnamed control sections 28
Statements 11, 13
Boundaries 11
Examples 13
Macro instructions 66
Prototype 61
Summary of 119
Storage, defining (see DS instruction)
S-type address constant 48
Sublist (see operand sublist)
Subscripted SET symbols
Defining 90
Dimension of 91
Examples 91
How to write 90
Invalid examples of 90
Subscript of 91
Using 91
Examples 91
Valid examples of 90
Substring notation
Arithmetic expressions in 79
Character expression in 79
Concatenating 79
How to write 79
Invalid example of 79
SETB instruction 81
SETC instruction 79
Valid examples of 79
SYM card 151.1
Symbol definition, EQU instruction for 38
Symbols
Absolute 17
Defining 17
Length attributes 33
Referring to 21
Length, maximum 18
Ordinary 17
Previously defined 18
Relocatable 17
Restrictions 18
Sequence 17, 74
Symbol table capacity 126
Types of 17
Value attributes 33
Variable 17
Symbolic linkages 31
ENTRY instruction 31
EXTRN instruction 31
Symbolic operands formats 35
Symbolic parameter
Comments field 63
Concatenation of 64
Defined 63
How to write 63
Invalid examples of 63
Model statements 63
Prototype statement 62
Replaced by 63
Valid example of 63
System variable symbols
Assigned values by assembler 91
Defined 91
(see also specific system variable symbols)

T' (see Type attribute)

Tables, internal, capacity of 126
Terms
Expressions composed of 15
Pairing of 22
TITLE instruction 51
Type attribute
Defined 72
Literals 72
Macro instruction operands 72
Notation 71
SETC instruction 77
Use 72

Unconditional branch (see AGO instruction)
Unnamed control section 28
USING instruction 24, 33

Variable symbols 17
Assigning values to 59
Defined 59
How to write 59
Restrictions in use to generate operation codes 62
Summary chart of 125
System 92, 93
Summary chart of 125
Types of 59
Use 59
(see also specific variable symbols)
V-type address constant 48

WXTRN 32

XFR instruction 8

Y-type address constant 47

IBM System/360 (S360-21) Printed in U. S. A. GC24-3414-7

IBM

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

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]