SPECTRA 70

RADIO CORPORATION OF AMERICA • ELECTRONIC DATA PROCESSING

POS-TOS/TDOS ASSEMBLY SYSTEM REFERENCE MANUAL

RCA

PRIMARY OPERATING SYSTEM (POS)
TAPE OPERATING SYSTEM (TOS)
TAPE-DISC OPERATING SYSTEM (TDOS)

ASSEMBLY SYSTEM REFERENCE MANUAL

SPECTRA 70

RADIO CORPORATION OF AMERICA • ELECTRONIC DATA PROCESSING

PRIMARY OPERATING SYSTEM (POS)
TAPE OPERATING SYSTEM (TOS)
TAPE-DISC OPERATING SYSTEM (TDOS)

ASSEMBLY SYSTEM REFERENCE MANUAL

RADIO CORPORATION OF AMERICA

70-00-602
March 1968

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

First Printing: November, 1965
Reissued: February, 1966
Reissued: March, 1968

FOREWORD

◆ This publication is intended as a reference manual for the programmer using the assembly language. It contains all information necessary to program in assembly language when used with the Primary (POS), Tape (TOS), or Tape/Disc (TDOS) Operating System Reference Manuals.

The information in this publication is stated based on the assumption the reader knows basic programming concepts and has had programming experience with computer systems. It is assumed the reader understands the content of the Spectra 70/35-45-55 Programmers' Training Manual (70-35-801).

Macro definition language specifications are included in the latter sections of this Assembly System Reference Manual. RCA supplied macros are described in the appropriate Operating System Reference Manuals (POS 70-00-605 and TOS/TDOS 70-00-608). Spectra 70/25 Assembly Language exceptions are summarized in Appendix F.

CONTENTS

CONTENTS
(Cont'd)

CONTENTS
(Cont'd)

CONTENTS
(Cont'd)

CONTENTS
(Cont'd)

1. SPECTRA 70 ASSEMBLY SYSTEM

INTRODUCTION

◆ The Spectra 70 Assembly System is a machine-oriented, symbolic programming language which expedites the writing of programs for Spectra 70 Systems. Assembly language programs consist of four basic types of statements: machine instructions, assembly instructions, macro instructions, and comments statements.

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

Assembly instruction statements provide auxiliary functions that assist the programmer in checking and documenting his programs, in controlling the assignment of storage addresses, in program sectioning and linking, in defining data and storage fields, and in controlling the Assembly System itself. Assembly instruction statements specify these auxiliary functions to be performed by the assembly, and, with a few exceptions, do not result in the generation of any machine language code by the assembly.

Macro instruction statements enable the Assembly System to retrieve specially coded symbolic routines, modify these routines according to information supplied in the macro instruction, and insert the resultant generated source statements into the assembly process for translation into machine language.

The Assembly System resides on a systems tape and operates under control of a control system which provides input/output, library, and other services required in assembling a source program. Device interchangeability at assembly time also is provided to permit substitution of magnetic tape for source input, object program, and program listings.

FEATURES

Mnemonic Operation Codes

◆ Predefined mnemonic codes are provided in the assembly language for all machine instructions and assembly instructions. Additional extended mnemonics are provided for the various forms of the Branch-on-Condition instruction.

Symbolic Addressing

◆ The assembly language provides for the symbolic representation of addresses, machine components (such as registers), and actual values required in source statements.

Data Representation

◆ Decimal, binary, hexadecimal, and character representations of machine language values can be used by the programmer in writing source statements. The programmer selects the representation best suited to his purpose.

Program Sectioning and Linkage

◆ The assembly provides facilities for generating (optionally) multi-sectional programs, and for symbolically linking separately assembled programs or program sections.

The output of the assembly consists of the assembled control sections and an External Symbol Dictionary. The External Symbol Dictionary contains information that the Linkage Editor requires to complete cross-referencing between control sections as it combines these sections into a single object program.

Symbols can be defined in one assembly and referred to in another assembly, thus providing symbolic linkages between independent assemblies. Specifically, these symbols provide linkages between separately assembled control sections. The assembly places the required linkage information in the External Symbol Dictionary (ESD) on the basis of the linkage symbols identified by the ENTRY and EXTRN assembly instructions.

The ENTRY instruction identifies the symbol, within a given assembly, that is to be used as the name of the entry point from another program (or section). Similarly, the program that uses a symbol defined in some other assembly must identify it by use of the EXTRN instruction, which provides linkage to the point of the definition.

Base Register Calculation

◆ The base register addressing scheme requires the designation of a general register (containing a base address value) and a displacement value for specifying a storage location. The Assembly System assumes the clerical burden of calculating storage addresses in these terms for the symbolic address used by the programmer. The programmer retains control of general register usage and the values entered therein by means of the USING and DROP assembly instructions.

Relocatability

◆ Object programs produced by the Assembler are in a format that permits them to be relocated from the originally assigned areas to any other suitable area. It is also possible to produce object programs that are absolute (not relocatable).

Program Listings

◆ A listing of the source program statements and the resulting object program statements may be produced by the Assembly System for each source program it assembles. The programmer can partly control the format 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 Assembly language. Detected errors are indicated in the program listing. Up to six error flags are printed for each statement processed that has been found to contain errors.

MINIMUM EQUIPMENT REQUIREMENTS

◆ The minimum equipment configurations to operate the Assembly System under control of the POS, TOS, and TDOS operating systems are detailed below. In each case, additional memory over the stated minimum is used to allow more symbols and to process macro expansions more efficiently. (The maximum number of symbols permitted for each system is discussed in Appendix D.) In addition, it should be noted that the output device and listing device required for assembly output may be omitted if no output is desired. (See AOPTN control message, page 5-6.)

POS Equipment Requirements

◆ The Primary Operating System equipment requirements are as follows:

Processor	- Model 70/35D, 70/45D, or 70/55E.
Magnetic tape devices (four required)	- Includes three work tapes (capable of being read in reverse direction) and the system tape. (See Notes 2 and 4.)
Input Device	- Card reader or magnetic tape.
Output Device	- Card punch or magnetic tape. (See Note 2.)
Listing Device	- Printer or magnetic tape.

Notes

◆ 1. If the source input is contained on magnetic tape, it may be batched in blocks of one to five cards.

2. If UPSI switch 0 is set ON, the assembly uses only two work tapes (SYS001 and SYS002). SYS000 is not used which allows for object output to tape on a four-tape system.

3. Object programs are batched in blocks of one to five cards and may be stacked on the output tape.

4. The systems tape may be a seven-level tape.

TOS Equipment Requirements

◆ The Tape Operating System equipment requirements are as follows:

Processor	- Model 70/35E, 70/45E, or 70/55E.
Magnetic tape devices (five required)	- Includes three work tapes, the Call Library Tape (all capable of being read in reverse) and the nine-channel system tape. (See Note 3.)
Input Device	- Magnetic tape or card reader.
Output Device	- Magnetic tape or card punch. (See Note 6.)
Listing Device	- Printer or magnetic tape.

Notes

1. Batched assemblies are permitted by TOS Assembly. That is, between the END card of program N and the START card of program N + 1, no Monitor control cards are present. Object coding is batched in blocks of one to five cards and may be stacked on the output tape. Linkage Editor and other system utility routines require object module files to be in ascending sequence by program name.

2. If the source input is contained on magnetic tape, it may be batched in blocks of one to five cards.

3. If all macros used are submitted with the source program, or if no macros are used, the Call Library Tape is not required.

4. If the object coding and listing information are assigned to magnetic tape, they must be assigned to the same device.

5. The optional source language correction and update feature requires one or two additional tape devices. (See Appendix G.)

6. Generation of the object program may be omitted or it may be generated on:

   a. SYSOPT.

   b. SYSUT1 or alternate device.

   c. Both a and b above.

   If SYSUT1 already contains an object module or is to receive the object program, it is considered unavailable as a work tape. An alternate work tape (SYSUT4) can be specified, if available. If an alternate tape is not specified, the assembly operates with only two work tapes.

TDOS Equipment Requirements

The Tape/Disc Operating System equipment requirements are as follows:

Processor	- Model 70/35E, 70/45E or 70/55E.
Magnetic Tapes (three required)	- Work Tapes. Two of these tapes must be nine-level. If a seven-level tape is used, it must have the pack/unpack feature.
Disc Storage Unit or Drum Memory Unit	- Macro library and System library are on this device.
Input Device	- Magnetic tape or card reader.
Output Device	- Magnetic tape or card punch.
Listing Device	- Printer or magnetic tape.

Notes

◆ 1. Refer to Notes 1 through 6 under TOS Equipment Requirements.

2. The Macro library (if present) must reside on a random access device. This may be either the device containing the system library or a separate device.

3. Input and output devices must be assigned to card devices and/or magnetic tape devices. The Program Load Library produced by the Linkage Editor may be transcribed to a random access device or operated directly from tape.

2. ASSEMBLY LANGUAGE STRUCTURE

THE CODING FORM

♦ The coding associated with a statement line normally occupies columns 1 through 71 and, if needed, columns 16 through 71 of a *single* continuation line. A continuation line is designated by entering any nonblank character in column 72 of the statement line to be continued. Columns to the left of column 16 on the continuation line must be blank.

Note:

Only one continuation line is allowed for assembly instructions.

Source statements normally occupy columns 1 through 71 of the statement line and 16 through 71 of a continuation line. Therefore, columns 1, 71, and 16 are referred to as the begin, end, and continue columns, respectively. These standards may be altered by the use of the controlling code, Input Format Control. (See ICTL, page 5-8.)

Statements may consist of from two to four entries in the statement field. They are, from left-to-right: An 8-character Name entry, a 5-character Operation entry, and a 56-character Operand and/or Comments entry.

Name Field

♦ The Name entry is an optional symbol created by the programmer to identify the statement line. The symbol must consist of eight characters or less, and, if used, must start in the begin column of the statement line. If the begin column is blank, the Assembler assumes that the statement line is unnamed. Rules for proper symbol definition are listed on page 2-4.

Operation Field

♦ The Operation entry is a mandatory entry that begins at least one position to the right of the begin column, and specifies the machine mnemonic or assembly function desired. Valid operation codes consist of five characters or less, and may not contain embedded blanks.

Operand Field

♦ Depending on the requirements of the instruction specified in the operation entry, this entry contains coding that identifies and/or describes storage, masks, storage-area length, or types of data. One or more Operand entries may be needed to properly specify the instructions. Operand entries are separated by commas; blanks may not intervene between the operands and the commas that separate them.

Operand entries may not contain embedded blanks, except when the Operand entry is used to specify constants, literals, or immediate data, and the data string contains blanks. The Operand field must start at least one position to the right of the Operation field. In the absence of a Comments field, the operand field may extend through the "END" column.

Symbols appearing in operand entries must be defined only once in a program. A symbol is defined when it appears in the name field of a statement.

CHARGE NO. ____________
DATE REQ'D ____________

RCA

SPECTRA 70
ASSEMBLY PROGRAM FORM
FLOW CHART REFERENCE ____________

DATE ________ PAGE ____ OF ____
PROGRAM ____________
PROGRAMMER ____________

NAME		OPERATION		OPERAND COMMENTS		IDENTIFICATION
1 2 3 4 5 6 7 8	9	10 11 12 13 14	15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71	72	73 74 75 76 77 78 79 80
1 2 3 4 5 6 7 8	9	10 11 12 13 14	15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71	72	73 74 75 76 77 78 79 80

28-00-119

Figure 2-1. RCA Spectra 70 Assembly Program Form

Comments Field

◆ Comments are descriptive items of information that are to be included in the program listing. All valid characters, including blanks, can be used in writing comments. Comment entries may follow the last operand entry. A blank must separate it from the last operand. Comment entries may not extend beyond the end column, and a blank must be used to separate it from the operand.

One or more statement lines may be used entirely for comments by placing an asterisk (*) in the begin column of each statement line.

In statements where an optional Operand entry is omitted and comments are desired, the missing operand must be indicated by a comma followed by one or more blanks prior to writing comments.

Identification-Sequence Field

◆ An optional entry, when used, specifies program identification and/or statement sequence characters. If the entry or a portion of the entry is used for program identification, the identification is punched by the programmer in the statement cards, and reproduced by the assembly in the source program listing.

As an aid in keeping source statements in order, the programmer may code a sequence of characters in ascending order in this field, or a portion of this field, which will be checked by use of the input sequence check instruction. (See ISEQ, page 5-10.)

CHARACTER SET

◆ Assembly language statements are written using the following letters, numeric digits, and special characters:

Letters:	There are 29 characters classified as letters. These include the characters @, #, and $ as well as the alphabetic characters A through Z. The three additional characters are included so that the category can accommodate certain languages.
Numeric Digits:	0 through 9.
Special Characters:	+ - , = . * ( ) ' / & BLANK

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

TERMS

◆ All terms represent a value. This value may be assigned by the Assembly System (symbols, length attributes, Location Counter reference, and literals) or may be inherent in the term itself (self-defining terms).

Symbol Definition

◆ Symbols provide the most commonly used means of addressing instructions, constants, storage locations, and control sections. Symbols are normally defined in the Name field of a source statement line. A symbol that is defined in another assembly is specified as defined elsewhere by the EXTERN statement. (See EXTERN, page 4-16.) After a symbol has been defined, it can be referred to by other Operand entries.

The value assigned to the symbol is the address of the leftmost byte of the instruction, constant, storage location, or control section named by the symbol. Because the address of these items may change upon program relocation, the symbols naming them are considered relocatable terms. The value of a symbol may be equated to an absolute value. (See EQU, page 3-15.)

Symbol Table

◆ The Assembly System compiles a table containing all the symbols that appear in the Name field. A specific memory address and a length attribute are stored in the symbol table. The length attribute of a symbol is the size, in bytes, of the storage field named by the symbol. References to symbols cause the Assembly System to interrogate the symbol table for the address and the size of the field being referred to. This information is used by the Assembly System for instruction address generation. Correct symbol definition is dependent on the following rules:

1. A symbol can be a single character or group of characters created from the standard character set, not exceeding eight characters.
2. A symbol must begin with an alphabetic character other than the letter I. The remaining characters may be alphabetic, numeric or a combination thereof.
3. No special characters or embedded blanks may appear within a symbol.
4. A symbol may be defined only once in any single assembly. Thus, two or more control sections assembled together cannot define the same symbol. (Exception: The Name field of a control section used in the START, CSECT, or DSECT assembly statements. (See page 4-3.)
5. Symbol values may not exceed a value of $2^{19}-1$ (524,287).
6. The maximum number of symbols permitted in an assembly is dependent on the amount of memory available to the Assembler. See Appendix D for a complete summary of symbol limits.
7. Operand entries used within the instruction may contain addresses that are generated from other than symbol table references. These entries are classified as self-defining terms, literals, constants, or expressions.

Valid Symbols

◆ The following are examples of valid symbols:

READER

A2345678

N

X4F2

S4

$3

Symbol Length Attribute

◆ The length attribute of a symbol may be used as a term. Reference to the length attribute is made by writing the symbol preceded by the letter L and a single quotation mark, for example, L'BETA. The assembly substitutes the associated length for the symbol.

Chart 2-1 shows how a programmer might use a symbol length attribute. A1 names a storage location of eight bytes, and B2 names a character constant that is two bytes in length. The statement line named HIORD moves the contents of B2 into the leftmost two bytes of storage location A1. The term (L'B2) supplies the length specification required by the machine instruction MVC. Statement line named LOORD moves the contents of B2 into the rightmost two bytes of A1. The expression A1 + L'A1 - L'B2 results in a value equal to the seventh byte of field A1. Again, (L'B2) supplies the length specification needed by the instruction.

Chart 2-1. Use of Symbol Length Attribute

◆

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

Self-Defining Terms

◆ A self-defining term is one whose value is inherent in the term. The Assembler program does not assign a value to the term but uses the term itself as the value to be assembled. All self-defining terms are classified as being absolute, since the value of the term does not change when the program in which they appear is relocated.

Self-defining terms are the means of specifying immediate data, masks, addresses, registers, operand lengths, and I/O information to the Assembler. Self-defining terms differ from constants and literals when used in instructions in that the value of the term is assembled into the instruction. By contrast, a data constant or literal has the address of the data assembled into the instruction.

Four types of self-defining terms are available to the programmer: decimal, binary, hexadecimal, and character.

Decimal Self-Defining Terms

◆ A decimal self-defining term is an unsigned decimal number written as a series of decimal digits. Limitations as to the value of the term depends entirely on its use within the program. Decimal self-defining terms are assembled as binary equivalents, and must not exceed eight digits (2^{24}-1). (See chart 2-2.)

Chart 2-2. Example of Decimal Self-Defining Terms

◆

NAME	OPERATION	OPERAND	COMMENTS
	MVC	TO(225),FROM	MOVE 225 BYTES
	AP	SUM(12),ADD(3)	ADD THREE BYTES OF PACKED DECIMAL TO A 12 - BYTE FIELD
	BC	15,COMPUTE	UNCONDITIONAL BRANCH TO COMPUTE

Hexadecimal Self-Defining Terms

◆ A hexadecimal self-defining term is an unsigned hexadecimal number written as a series of hexadecimal digits. The digits must be enclosed in single quotation marks and be preceded by the letter X. Each hexadecimal digit is assembled as a four-bit binary value. The maximum hexadecimal term is X'FFFFFF'. (See chart 2-3.) The following is a summary of the hexadecimal bit patterns:

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

Binary Self-Defining Terms

◆ A binary self-defining term is an unsigned sequence of 0's and 1's enclosed in single quotation marks and preceded by the letter B; for example, B'1011'. The maximum binary self-defining term is 24 bits. These terms are used primarily in designating bit patterns for masks used in logical instructions. (See chart 2-4.)

Character Self-Defining Terms

◆ A character self-defining term consists of from one to three characters enclosed in single quotation marks and may be preceded by the letter C. All letters, decimal digits, and special characters may be used in this type of self-defining term. Any of the 256 punch combinations may be used to indicate the character that will be assembled in 8-bit code. (See chart 2-5.)

Note:

Care must be used when specifying the characters single quotation (') or ampersand (&) in character self-defining terms or character constants. The assembly itself uses these characters to denote special functions. When the programmer uses these characters, two quotation marks or ampersands must be indicated; for example, to specify the term A'# as a character constant, the programmer would write C'A''#'.

Chart 2-3. Example of Hexadecimal Self-Defining Terms

◆ NAME	OPERATION	OPERAND	COMMENTS
	BC	X'4',ABLE	BRANCH TO ABLE IF CONDITION CODE IS 0100 (CONDITION CODE 1)

Chart 2-4. Example of Binary Self-Defining Terms

◆ NAME	OPERATION	OPERAND	COMMENTS
	TM	CODE,B'10101010'	

Chart 2-5. Example of Character Self-Defining Terms

◆ NAME	OPERATION	OPERAND	COMMENTS
	TM	MEM,C'1'	THE CHARACTER 1 (11110001) IS USED AS MASK
	MVI	SWITCH,'1'	

Literals

◆ A literal term is a convenient way of entering data into a program. It is a constant preceded by an equal sign (=) coded as an operand in an instruction; for example: MVC FIELD(1),=C'A'. The constant itself is specified in the same manner as in a define constant (DC) statement. (See DC, page 3-16.)

Literals represent data, not references to where data is stored. The use of a literal in a statement line directs the Assembly System to place the value of the literal into a reserved portion of memory called a literal pool and to substitute this assigned address in place of the literal.

Defining Literals

◆ All types of address constants (except S-type) can be expressed as literals. A duplication factor of zero is not permitted in a literal.

Chart 2-6 shows the use of a literal as an Operand entry. The statement named Alpha is an AP instruction with the second Operand field containing a literal. When assembled, the literal is replaced with an address of the location in which the assembly has stored the binary value of P'1'.

Notes:

1. Only one literal may appear in a statement line.
2. Literals may not be combined in expressions.
3. Program instructions cannot alter literals.
4. Literals cannot be receiving fields.
5. Literals may not be used in address constants.
6. Literals are considered to be relocatable terms.

Literal Pool

◆ Literals collected by the assembly are placed in a special area called a literal pool. The positioning of the literal pool, if not controlled by the programmer, will be the end of the first control section. The programmer may create multiple literal pools and/or relocate the literal pool under control of the LTORG assembly instruction. (See page 4-10.)

Chart 2-6. Use of Literal as Operand Entry

◆

NAME	OPERATION	OPERAND
ALPHA	AP	COUNT, =P'1'

Location Counter Reference

◆ The Spectra 70 Assembly System maintains an internal Location Counter for each control section under assembly. This counter is similar to the Program Counter which contains the main memory address of the next instruction to be executed. The Location Counter in the assembly assigns storage addresses to program statements. Program statements for each section are assigned addresses from the Location Counter for that section.

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. Therefore, the Location Counter always points to the location of the next available storage location in memory after the instruction has been assembled.

The programmer may refer to the current setting of the Location Counter by inserting an asterisk (*) in the Operand field entry. This method of addressing is the same as assigning a name to the statement line and using the name as an Operand entry. The leftmost byte address is supplied when reference to the Location Counter is made within an instruction.

The location counter setting can be controlled by using the START and ORG Assembler instructions (see pages 4-3 and 3-11). The Counter affected by either of these instructions is the counter for the control section in which they appear. The maximum value of the location counters is $2^{24}-1$ on the 70/35-45-55 Processors.

EXPRESSIONS

◆ Operand entries written for the Spectra 70 Assembly System consist of either a single term or an arithmetic combination of terms and are referred to as expressions. An expression can be considered as being either simple or multiterm. Simple expressions are Operand entries containing symbols, self-defining terms, Location Counter references, literals, or length attributes. Multiterm expressions are simple expressions that have been combined by arithmetic operators for evaluation.

Terms may be combined by use of the following arithmetic operators:

+ Addition; that is, Alpha + 2

⊖ Subtraction; that is, Alpha ⊖ Beta

* Multiplication; that is, 5 * L'Beta

/ Division; that is, (Alpha - Beta)/2

Combining Terms

◆ The following rules describe the method by which terms can be combined; these rules must be followed if expressions are to be evaluated properly.

1. Terms may be grouped within parentheses to indicate the order in which they are to be evaluated. The terms within parentheses (grouped) are evaluated first; this value is then used to reduce the rest of the expression to another single value.
2. Expressions may not begin with an arithmetic operator, that is, (+ , ⊖ , * , /) .
3. Expressions may not contain two terms or two operators in succession.
4. Expressions may not contain more than three levels of parentheses, that is, nest of three.
5. Final values of expressions may not exceed a maximum value of $2^{19}-1$, or have an intermediate value greater than $2^{31}-1$.
6. Multiterm expressions may not contain Literals.

The following are examples of valid expressions:

Simple Expressions	Multiterm Expressions
FIELD	AREA+X'2D'
L'FIELD	*+32
B'101'	N-25
C'ABC'	FIELD+332
29	((EXIT - ENTRY)/8)
=C'ABC'	L'BETA*10
*	TEN/TWO

Expressions are evaluated in a definite order. The following rules define this method of evaluation:

1. Single expressions take on the value of the term involved, that is, BETA,X'123',*,L'TAG.
2. Multiterm expressions, are scanned from left-to-right, and each term is assigned a value.
3. The terms within the parentheses are evaluated first, with multiplication and division preceding addition and subtraction.
4. Division by zero is valid and produces a zero result.
5. Division yields an integer result; fractions are dropped.

Absolute Expressions

◆ Expressions can be further divided into two additional classifications namely, absolute and relocatable expressions. An expression is called absolute if its value is unaffected by program relocation. An absolute expression may be a single absolute term or an arithmetic combination of absolute terms. An absolute term may be an absolute symbol, self-defining term, or length attribute. All arithmetic operators are permitted between absolute terms.

Paired Terms

An absolute expression may contain two relocatable terms (RT), along or in combination with an absolute term (AT) provided:

1. The relocatable terms are paired, that is, they must appear within the same control section and have opposite signs. The paired terms do not have to be contiguous, for example, RT+AT ⊖ RT.

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

The pairing of relocatable terms cancels the effect of relocation. Therefore, the value represented by the paired terms remains constant, regardless of program relocation.

The following combinations illustrate absolute expressions:

R1 ⊖ R2

R1 ⊖ R2 + A

A ⊖ R1 + R2

A * A

* ⊖ R1

where:

R1, R2 = Relocatable Terms from the same control section.

A = Absolute Terms.

Note

◆ A reference to the location counter must be paired with another relocatable term from the same control section.

Relocatable Expressions

◆ A relocatable expression is one whose value would change by n if the program in which it appears is relocated n bytes away from its originally assigned storage area. All relocatable expressions must have a positive value.

Relocation is needed to load the object program (control section) into storage locations other than those originally assigned by the Assembler. All addresses using the same base register may be relocated by simply changing the contents of that base register upon loading.

Relocatable Expressions (Cont'd)

A relocatable expression may contain relocatable terms, alone or in combination with, absolute terms under the following conditions:

1. Relocatable expressions must contain an odd number of relocatable terms. If a relocatable expression contains three relocatable terms, two of them must be paired. Pairing is described under Absolute Expressions, above.
2. A relocatable term may not enter into a multiply or divide operation.
3. 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: RT1 - RT2 + RT1, RT1 and RT2 are relocatable terms from the same control section. If RT1 equals 10 and RT2 equals 5, the value of the expression reduces to 15. However, if the program is relocated 100 bytes from its original location, the value of the expression becomes 115. The paired terms RT1 and RT2 remain constant at 5 regardless of the relocation factor. Thus, the result of the expression is the value of the unpaired term RT1 adjusted by the value of RT1 - RT2.

The following examples are valid relocatable expressions. A is an absolute term, RT1 and RT2 are relocatable terms from the same control section and Y is a relocatable term from a different control section.

Y - 32*A	=X'1234'
RT1 - RT1 + RT2	A*A + RT1
RT1 - RT2 + *	RT1 - RT2+Y
* (location counter reference)	RT1

A reference to the Location Counter in an expression must be paired to a relocatable term in the same control section as: *-TAG.

ADDRESSING

Base Register Calculation

◆ Spectra 70 addresses may range from zero through $2^{19}-1$. The final address is produced by adding the base address value in a general register and a displacement value. The final address may be produced by adding a third value (index factor) from another general register in certain instructions. The Assembler permits the programmer to specify the general register(s) and the displacement explicitly or to direct the Assembler to calculate the address from a symbolically stated address. The programmer can direct the Assembler to perform address calculation by specifying which general registers are available as base registers and what values each register is assumed to contain. (See USING, page 2-15.) Whenever the Assembler encounters a symbolic address in the operand field of an instruction it determines the base register and displacement value for this address by subtracting it from the value in each available register. The register producing the smallest displacement below 4,095 is selected. If two or more registers produce the same displacement, the highest numbered register is used.

Register Considerations

◆ Certain general registers have special uses in conjunction with the Operating System, particularly for input output functions. (See pertinent FCP Reference Manual.)

Values placed in general registers must be word-aligned. They are automatically aligned when expressed as address constants.

A register designated as containing an absolute value is available only for absolute addresses. If the absolute value is less than 4,096 and a base register has not been specified, the Assembler will select register 0.

Explicit Addressing

◆ The programmer may refer to an address explicitly in a given instruction by coding the base register and displacement as self-defining terms. (See Section 3 for correct coding options for each instruction class.) Chart 2-7 illustrates an explicitly coded instruction.

Chart 2-7. Example of Explicit Addressing

NAME	OPERATION	OPERAND
	BC	15,4(0,8)

Chart 2-8. Example of Implied Addressing

NAME	OPERATION	OPERAND	COMMENT
	MVC	ABLE,BAKER	(IMPLIED LENGTH AND REGISTER)

Implied Addressing

◆ In chart 2-8 it is assumed that the names ABLE and BAKER are assigned the addresses $3850_{(10)}$ and $8173_{(10)}$ in the symbol table and that General Registers 2 and 3 contain values of $0100_{(10)}$ and $4195_{(10)}$. In interpreting the Move (MVC) instruction, the Assembly System subtracts the base value from the address associated with the symbol. The difference is the displacement. The displacement may not be negative and may not exceed 4095.

The resulting machine instruction is:

OP	L	B1	D1	B2	D2
D2	00	2	3750	3	3978

Relative Addressing

◆ Relative addressing is a technique of addressing instructions and data areas by designating their location as relative to a symbolic location. The programmer can refer to any location to the right or left of a defined symbol by indicating a plus (+) or minus (-) value; for example, SYMBOL ± VALUE. The value specified is always in terms of bytes. (See chart 2-9.)

Chart 2-9. Relative Addressing

NAME	OPERATION	OPERAND
	MVI	PRINT,X'40'
	MVC	PRINT + 1(131),PRINT

Self-Relative Addressing

◆ Self-relative addressing allows the programmer to use the current value of the Location Counter plus or minus a value to refer to locations (in bytes) of various locations within the program. (See chart 2-10.)

Chart 2-10. Self-Relative Addressing

◆

NAME	OPERATION	OPERAND	
	BC	0,*+18	4 BYTES
	MVI	TABLE,X'00'	4 BYTES
	MVC	TABLE + 1(255),TABLE	6 BYTES
	MVI	* - 13,X'F0'	4 BYTES
	MVC	RECORD,WORK	6 BYTES

Further details on permissible instruction coding formats are found in Section 3.

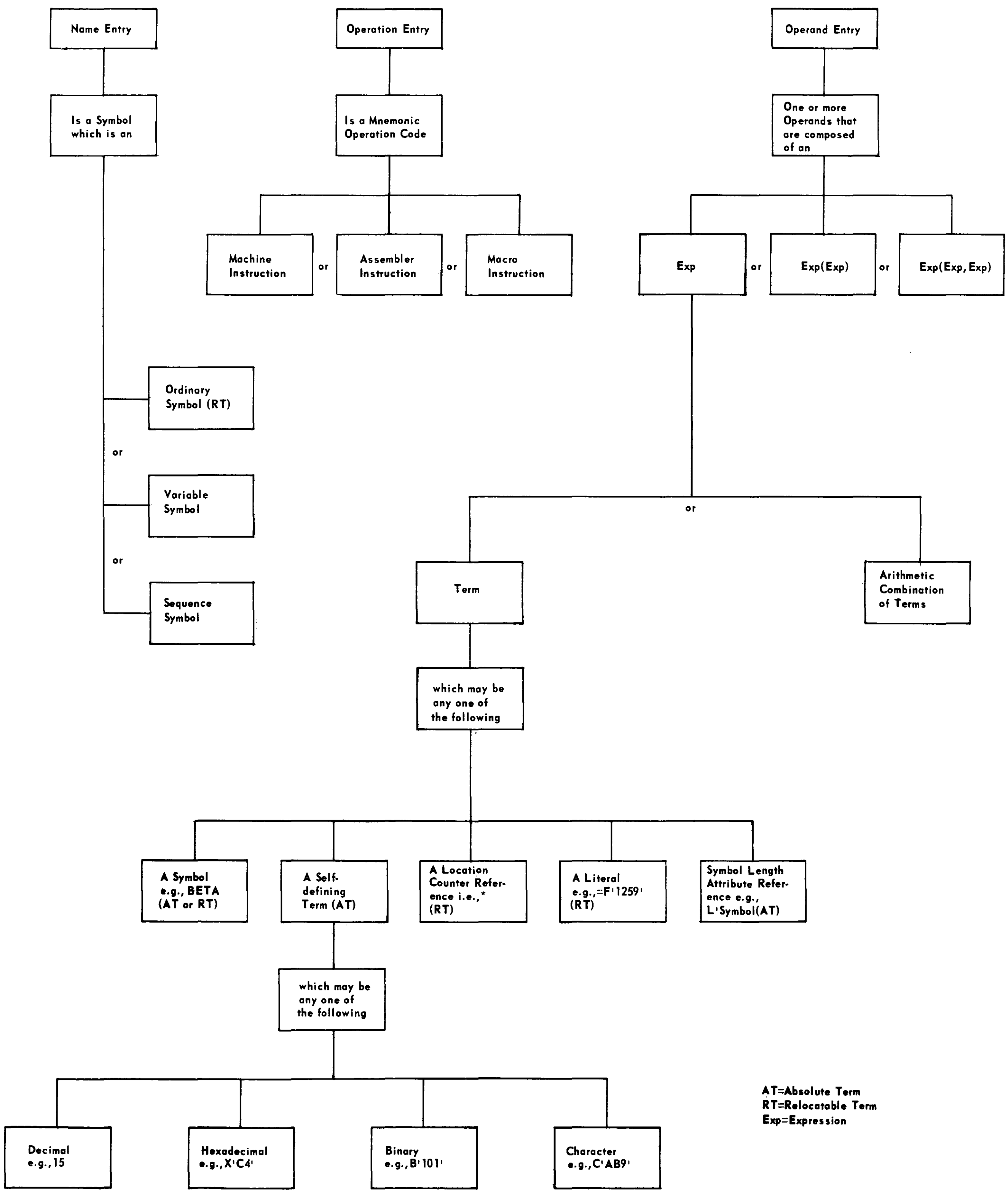


Figure 2-2. Assembler Language Structure - Machine and Assembler Instructions

USING
Use Base Register

General Description

◆ The USING instruction indicates to the Assembler the general register(s) that are available for use as base registers, and the value(s) that the register(s) are assumed to contain at object time.

Format

◆ The format of the USING instruction is as follows:

NAME	OPERATION	OPERAND
Not Used	USING	(Expressions of the form: V, r1, r2, r3, r4, r5 ---, rx)

Specification Rules

Name Field

◆ Not used.

Operation Field

◆ USING.

Operand Field

◆ Contains the base address value and the register(s) to be assigned. Operand V must be an absolute or relocatable expression. Literals are not permitted. The value that are assumed in the base registers r1 through rx will be in the form of V, V+4096, V+8192, V+12288, etc. The expressions used to indicate the registers r1 through rx must be between 1-15. Any number of registers may be specified in one USING statement.

Example

◆ (See chart 2-11.)

Notes

◆ 1. The USING instruction may be used as often as needed and at any point in the program to indicate to the Assembler changes in the register(s) or their value(s).

2. Since the USING instruction does *not* actually load the assigned general registers; it is the user's responsibility to ensure that the register(s) are loaded with the value(s) specified in the USING instruction.

3. General register 0 may not be used as a base register.

Chart 2-11. Example of USING Instruction

NAME (1–8)	OPERATION (10–14)	OPERAND (16–)	COMMENTS
	USING	*,8,9	(LOCATION COUNTER, LOCATION COUNTER+4Ø96 WILL BE
			ASSUMED LOADED IN GENERAL REGISTERS 8 AND 9

DROP
Drop Base Register

General Description

◆ The DROP instruction allows the user to eliminate a general register(s) previously assigned in a USING statement.

Format

◆ The format of the DROP instruction is as follows:

NAME	OPERATION	OPERAND
Not Used	DROP	Absolute Expression(s) of the form r1, r2, 43, r4, r5 --- rx

Specification Rules

Name Field

◆ Not used.

Operation Field

◆ DROP.

Operand Field

◆ Contains absolute expressions indicating the register(s) to be dropped. Any number of registers may be dropped with one DROP instruction. Only those registers specified will be dropped.

Example

◆ (See chart 2-12.)

Notes

◆ 1. It is not necessary to DROP a general register before changing its value with another USING instruction.

2. A dropped register may be made available again through another USING instruction.

Chart 2-12. Example of DROP Instruction

NAME	OPERATION	OPERAND	COMMENTS
	USING	*,3,4,5,6	(ASSIGNING REGISTERS)
	.		
	.		
	DROP	3,6	(DROP REGISTERS 3 AND 6. 4 AND 5
			STILL IN EFFECT)

Programming With the Using Instruction

◆ The USING (and DROP) instructions may be used anywhere in a program, as often as needed, to indicate the general registers that are available for use as base registers and the base address values the Assembler may assume each contains at execution time. Whenever an address is specified in a machine-instruction statement, the Assembler determines whether there is an available register containing a suitable base address. A register is considered available for a relocatable address if it was loaded with a relocatable value that is in the same control section as the address. A register with an absolute value is available only for absolute addresses. In either case, the base address is considered suitable only if it is less than or equal to the address of the item to which the reference is made. The difference between the two addresses may not exceed 4,095 bytes.

If two or more registers can be used to develop an address, the one yielding the smallest displacement is used. If two or more registers yield the same displacement, the highest numbered register is used. If an absolute address is less than 4,096, and if no base register has been specified, the Assembler will automatically select register 0.

Loading Register

◆ Using the BALR (Branch and Link Register) instruction and the USING instruction; chart 2-13 shows a possible method for loading a general register with the address of the first instruction of the program. The BALR loads General Register 2 with the address that is in the Program (P) register at object running time. The USING instruction notifies the Assembly that General Register 2 contains this value. When using this method, the USING instruction must immediately follow the BALR instruction and the last program statement line must be within the 4,095-byte range.

Expanding Assembly Addressing

◆ To expand the addressing capabilities of the assembly beyond 4,095 bytes with the LM (Load Multiple) instruction, the technique in chart 2-14 (on page 2-18) can be used.

In chart 2-14, the BALR instruction initially loads General Register 2 with the current value in the P register and proceeds to the next instruction. The USING instruction notifies the Assembly System that General Registers 2, 3, 4, and 5 are available for base addressing and contain the relocatable value of HERE. Here, being the name of the Load Multiple statement line, loads General Registers 3, 4, and 5 with address constants of HERE + 4096, 8192, and 12288. By increasing the number of general registers and constants, the number of addressable bytes can be increased.

Chart 2-13. Loading a General Register by way of BALR Instruction

◆

NAME	OPERATION	OPERAND
BEGIN	BALR	2,0
	USING	*,2
FIRST	⋮	

Chart 2-14. Expanding the Addressing Capabilities of the Assembly System

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

3. BASIC PROGRAM ELEMENTS

ASSEMBLY OF MACHINE INSTRUCTIONS

Machine Format

◆ Machine instructions are coded symbolically as assembly language statements. Instructions that require base-displacement format may be coded using implied addressing or explicit addressing.

The assembly language coding format varies for each class of machine instruction: RR, RX, RS, SI, and SS. Further coding variations are permitted within an instruction class. The assembly coding sequence that represents a machine instruction is:

1. Mnemonic operation code.
2. Operand operated upon.
3. Additional operand.

Any assembly instruction may be symbolically named such that any other assembly instruction may reference it by name as an operand. The symbol refers to the address of the leftmost byte of the instruction. The symbol is given the length attribute of the instruction being referenced. This length attribute is:

2 for RR instructions

4 for RX, RS, and SI instructions

6 for SS instructions

Instruction Alignment and Checking

◆ All generated instructions are properly aligned by the Assembler on half-word boundaries. Instruction alignment may cause the Assembler to skip bytes. These bytes are filled with hexadecimal zeroes.

Storage addresses are checked for boundary alignment appropriate for the instruction in which they occur. Similarly, instructions that require an even-numbered register designation are checked. They are: Multiply or Divide (word), Double Shift, and all Floating-point instructions.

For example, assume that FIELD is a relocatable symbol that has been assigned a value of 7400. Assume also that the assembly has been notified (by a USING instruction) that General Register 8 currently contains a relocatable value of 4096 and is available as a base register. The example in chart 3-1 shows a machine instruction statement as it would be written in assembly language and chart 3-2 shows the instruction as it would be assembled. The assembled instruction is presented in decimal.

Chart 3-1. Assembly Statement

◆

NAME	OPERATION	OPERAND
	STM	4,4,FIELD

Chart 3-2. Assembled Instruction

◆

OP	LI	L2	B1	D1
90	4	4	8	3304

Operand Formats

◆ An address may be specified explicitly as a base register and displacement by the formats shown in the second column of table 3-1. The address may be specified as an implied address by the formats shown in the third column.

Table 3-1. Explicit and Implied Operand Formats

◆

Type	Explicit Address	Implied Address
RX	D2(X2,B2)	S2(X2)
	D2(O,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)

Subfields

◆ A comma must be written to separate operand entries. 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. When two subfields are separated by a comma and enclosed by parentheses, the following rules apply:

1. If both subfields are omitted, the separating comma and parentheses must be omitted.
2. If the first subfield in the sequence is omitted, the comma that separates it from the second subfield must *not* be omitted. The parentheses must also be written. (See chart 3-3.)
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. (See chart 3-4.)

Chart 3-3. Separation of Operands

NAME	OPERATION	OPERAND
	MVC	32(16,5),FIELD2
	MVC	BETA(,5),FIELD2 IMPLIED LENGTH

Chart 3-4. Separation of Operands, Omitted Commas

NAME	OPERATION	OPERAND
INST1	MVC	32(16,5),FIELD2
INST2	MVC	FIELD1(16),FIELD2 IMPLIED ADDRESS

Subfields *(Cont'd)*

Fields and subfields in a symbolic operand are represented either by absolute or by relocatable expressions, depending on the requirements of the field. (An expression has been defined as consisting of one term or a series of arithmetically combined terms.)

Note:

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

The length field in certain 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 item 1 or 2, the length attribute for an expression is the length of the leftmost term in the expression. By contrast, an explicit length is written by the programmer in the operand as an absolute expression. The explicit length overrides any implied length.

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

Note:

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

To summarize, the length required in certain instructions can be specified explicitly by the formats shown in the first column of table 3-2, or can be implied by the formats shown in the second column. Observe that the two lengths required in one of the instruction formats are presented separately. An implied length is used for one and an explicit length is used for the other.

Table 3-2. Expressing Field Lengths

◆

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

Mnemonic Operation Codes

◆ The mnemonic operation codes are constructed so that they indicate the functions of the machine instruction. A modifier is appended as the last character to distinguish the function further. For example, the function of addition is designated by the mnemonic A (fixed-point arithmetic additions). This is distinguished from other arithmetic additions by appending another character, for instance:

AP	Add Packed-Decimal
AL	Add Logical
AH	Add Halfword
AE	Add Normalized (word) "Exponent"
AU	Add Unnormalized (word)
AD	Add Double word (normalized)
AW	Add Double word (Unnormalized)

Operand Fields

◆ An operand that represents an address in base-displacement form may be symbolically coded in implied or explicit form. If explicitly coded the Assembler requires the address to be expressed in the sequence D(B) in contrast to the machine-instruction format. Explicit addresses must be represented by absolute expressions.

An operand that represents a register may be coded as a self-defining (absolute) term or a symbol equated to an absolute term. (See EQU, page 3-15.)

Instructions of the RR format, where each operand is expressed as a single field without subfields, are coded in the form: operation, operand 1, operand 2. For example:

```
BALR 14,15
```

Instructions of the RS format that refer to a base-displacement address implicitly may also be coded explicitly. For example, either:

```
LM 3, 5, BASEVALU
     or
LM 3, 5, POINTER (2)
     or
LM 3, 5, 8 (2)
```

are acceptable assembly formats. Note that BASEVALU implies the base register and displacement; POINTER (2) states the base register explicitly, but implies the displacement; and that 8(2) states both base register and displacement explicitly. An implied address may be represented by either a relocatable or absolute expression.

Operand Fields (Cont'd)

The Shift instructions (RS) have several coding options. For example:

SLL 5,4(0)

and

SLL 5,4

will use the low-order six bits of the displacement as the shift count, but

SLL 5,0 (4)

will add the value of the displacement to the contents of register 4. The low-order six bits of the resulting sum will be used for the shift count.

Implied addresses are permitted provided the programmer specifies the base-register(s) and base value(s) with a USING statement and omits the base register. Explicit coding of the base register will override implied addressing. Omitting the base register reference permits the Assembler to select a suitable base register.

Instructions of the RX format reference an index register as well as the base register and displacement. Indexing is specified by appending the designated index register to the implied address. For example:

L 6,TABLE(8)

When no indexing is needed the appendage is omitted and register 0 is generated for the index register. An instruction which specifies index register 0 results in only the base register and displacement being used to form the effective address. For example:

L 6,VALUE

would generate a hexadecimal 60 in the second byte.

The explicit form may be used to form an address with indexing. For example:

CL 6,8(7,3)

forms the address of the second operand by adding the index value to the base and displacement value.

However, note that the explicit operand address has the form D(X,B). The indexing factor may not be omitted when the operand is coded explicitly. When the explicit form is used and indexing is not required, index register zero must be specified. For example:

ST 6,80(0,3)

results in storing the contents of register 6 without indexing.

Operand Fields (Cont'd)

A comma must be used to separate the index register from the base register. Both must be enclosed within parentheses. However, the base register and the comma may be implied by omitting both.

Note

◆ The value of a general register may be incremented by the value of the displacement when the LA (Load Address) instruction is coded explicitly, such as:

```
LA 6,100(0,6)
```

The instruction will take the value in register 6, add the displacement value 100 to it and then store it back in register 6. Reversing the base and index registers in the above example produces the same result. Register 0 may <u>not</u> be designated as the first operand for this purpose.

Instructions of the SS format are coded with the length subfield being implied or explicitly stated as:

```
MVC SAVE (256), WORK

    or

MVC SAVE, WORK
```

Further, packed decimal instructions with two length factors may be coded with implied or explicit lengths with either operand as:

```
SP BALANCE (6), AMOUNT (3)

    or

SP BALANCE, AMOUNT
```

Various combinations other than those above may be used such as:

```
MVC 48(L'ITEM,BR4),ITEM
```

Instructions of the SI format are coded as illustrated below.

```
TM CODE, B'10101000'

    or

OI DATA+6, X'F0'

    or

MVI FIELD-1, '$'
```

EXTENDED MNEMONIC CODES

◆ For the convenience of the programmer, the Assembly System 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 set of machine instructions, but are translated by the assembly into the corresponding operation and condition combinations. The allowable extended mnemonic codes are shown in table 3-3.

Table 3-3. Extended Mnemonic Codes

Extended Codes	Meaning	Extended Format	Machine Instruction
B	Branch Unconditional	D2(X2,B2)	BC 15,D2(X2,B2)
BR	Branch Unconditional	R2	BCR 15,R2
NOP	No Operation	D2(X2,B2)	BC 0,(X2,B2)
NOPR	No Operation (RR Format)	R2	BCR 0,RR
	Used After Compare Instructions		
BH	Branch on High	D2(X2,B2)	BC 2,D2(X2,B2)
BL	Branch on Low	D2(X2,B2)	BC 4,D2(X2,B2)
BE	Branch on Equal	D2(X2,B2)	BC 8,D2(X2,B2)
BNH	Branch on Not High	D2(X2,B2)	BC 13,D2(X2,B2)
BNL	Branch on Not Low	D2(X2,B2)	BC 11,D2(X2,B2)
BNE	Branch on Not Equal	D2(X2,B2)	BC 7,D2(X2,B2)
	Used After Arithmetic Instructions		
BO	Branch on Overflow	D2(X2,B2)	BC 1,D2(X2,B2)
BP	Branch on Plus	D2(X2,B2)	BC 2,D2(X2,B2)
BM	Branch on Minus	D2(X2,B2)	BC 4,D2(X2,B2)
BZ	Branch on Zero	D2(X2,B2)	BC 8,D2(X2,B2)
	Used After Test Under Mask Instructions		
BO	Branch if Ones	D2(X2,B2)	BC 1,D2(X2,B2)
BM	Branch if Mixed	D2(X2,B2)	BC 4,D2(X2,B2)
BZ	Branch if Zeros	D2(X2,B2)	BC 8,D2(X2,B2)

DEFINING STORAGE

◆ The DS instruction allows the programmer to reserve areas of memory for the storage of data and to assign names to those areas. Input/output areas and working storage can be classified as contiguous and non-contiguous storage. The Location Counter, which is used by the assembly to allocate storage, can be set and reset to any desired value through use of the ORG instruction. The setting and resetting of the Location Counter enables the programmer to define and redefine the allocated areas of memory.

DS
Define Storage

General Description

◆ The DS (Define Storage) instruction reserves working storage and input/output areas in memory. Names can be assigned to refer to these reserved areas symbolically.

Format

◆ The format for the DS instruction is as follows:

NAME	OPERATION	OPERAND
A symbol or blank.	DS	DTXn ['constant'] or DT

Specification Rules

Name Field

◆ Any symbol or blank.

Operation Field

◆ DS.

Operand Field

◆ One operand expression in the following format DTXn 'constant' where

D = the duplication factor.

T = the type of unit to be allocated halfword (H), fullword (F) double word (D) or byte (C).

Xn = the length of the field type to be reserved.

'constant' = a map of the actual data to be stored. (The data shown is used by the assembly for size calculation only. The constant shown is not stored in the allocated area.) This subfield is optional.

Chart 3-5. Example of DS Instruction

◆

NAME	OPERATION	OPERAND
READIN	DS	80C
AREA	DS	CL100
SOC#	DS	C'182243291'

H. F. D. Type Operands

◆ A DS (define storage) operand may have the format dt, where

d = a duplication factor.
t = a type code as follows:

TYPE	ADDRESS ALIGNMENT	IMPLIED LENGTH
H	Halfword	2 bytes
F	Word	4 bytes
D	Double Word	8 bytes

Additional examples of DS are given in chart 3-6.

Notes

◆ 1. The symbol in the Name field is assigned a left-hand byte address of the area allocated.

2. The length attribute is the length of the data type specified.

3. Skipped locations are not zeroed when proper positioning is necessary.

4. Packed (P), zoned (Z), character (C), hexadecimal (X), and binary (B) fields have an implied length of one byte. If more than one byte is to be reserved, the length modifier must be specified.

5. To reserve areas of storage greater than 256, a duplication factor must be used.

Forcing Alignment

◆ The Location Counter can be forced to a double-word, full word, or halfword boundary by using one of the three special field types shown in chart 3-7 with a duplication factor of zero.

The zero duplication factor in chart 3-7 can be used to assign a name to a field without actually reserving storage. Additional DS instructions can then be used to name the individual fields (see chart 3-8).

Chart 3-6. Additional Examples of DS Instruction

NAME	OPERATION	OPERAND	COMMENTS
ONE	DS	CL8Ø	ONE 8Ø-BYTE FIELD
TWO	DS	8ØC	8Ø 1-BYTE FIELDS
THREE	DS	6F	SIX FULL WORDS, ATTRIBUTE OF 4
FOUR	DS	D	ONE DOUBLE WORD, ATTRIBUTE OF 8
FIVE	DS	4H	FOUR HALFWORDS, ATTRIBUTE OF 2

Chart 3-7. Examples of DS Instruction Using Zero Duplication Factor

NAME	OPERATION	OPERAND	COMMENTS
	DS	ØD	DOUBLE-WORD ALIGNMENT
	DS	ØF	FULL-WORD ALIGNMENT
	DS	ØH	HALFWORD ALIGNMENT
	DS	ØF	
MASTER	DS	8ØC	ASSIGN ONE AREA 8Ø BYTES LONG
	ORG	MASTER	RESET LOCATION CTR TO LEFT BYTE OF MASTER
ITEM1	DS	CL1Ø	REDEFINE ALLOCATED 8Ø BYTE AREA
ITEM2	DS	CL2Ø	" " " " "
ITEM3	DS	CL1Ø	" " " " "
	ORG	MASTER+8Ø	RESTORE LOCATION CTR TO NEXT AVAILABLE LOC.

Chart 3-8. Examples of DS Instruction Naming Fields

NAME	OPERATION	OPERAND	COMMENTS
READA	DS	ØCL8Ø	NAME AND LENGTH ASSIGNED, NO STORAGE RESERVED.
ITEM1	DS	CL2Ø	READA DEFINED-BY THE FOLLOWING ITEMS.
ITEM2	DS	CL2Ø	
ITEM3	DS	CL2Ø	
ITEM4	DS	CL2Ø	
	ORG	ITEM4	REDEFINE ITEM 4
SUBITEM1		CL5	SUB 1
SUBITEM2		CL5	SUB 2
SUBITEM3		CL1Ø	SUB 3
	ORG	READA	
MASTER	DS	ØCL8Ø	RENAME READA TO MASTER

ORG
Set Location Counter

General Description

◆ The ORG instruction alters the setting of the Location Counter for the current control section.

Format

◆ The format of the ORG instruction is as follows:

NAME	OPERATION	OPERAND
Not Used	ORG	A relocatable expression or blank.

Specification Rules

Name Field

◆ Not used.

OperationField

◆ ORG.

Operand Field

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

Chart 3-9. Example of ORG Instruction

◆

NAME	OPERATION	OPERAND
	ORG	*+500
	ORG	START

Notes

◆ 1. The Location Counter is set to the value of the expression in the Operand Field.

2. Omission of an Operand entry causes the Location Counter to be set one byte higher than the maximum location assigned for the control section up to this point.

3. An ORG statement must not specify a location below the beginning of the current control section.

Contiguous Assignments in Allocating Storage

◆ Contiguous memory, such as input/output areas, may be allocated in units of bytes (C), halfwords (H), words (F), and double words (D). The Location Counter is positioned to the proper boundary before the desired storage area is allocated. The area allocated will not be filled with zeros.

To redefine the areas allocated in chart 3-10, the programmer can, through use of the ORG instruction, reset the Location Counter to the lefthand value originally used by INPUT. (See chart 3-11.)

To reset the Location Counter to the next available location for storage assignment, a relative address of INPUT + 80 is used.

Chart 3-10. Example of Contiguous Area Assignment

◆

NAME	OPERATION	OPERAND
INPUT	DS	80C
	ORG	INPUT
NUMB	DS	10C
CODE	DS	10C
TYPE	DS	10C
SIZE	DS	10C
COLOR	DS	10C
AMT1	DS	10C
AMT2	DS	10C
EM	DS	10C

Chart 3-11. Redefining Areas Using ORG Instruction

◆

NAME	OPERATION	OPERAND
	ORG	INPUT
ITEM1	DS	40C
ITEM2	DS	40C

Noncontiguous Assignments in Allocating Storage

◆ Through the use of the ORG and the unit of assignment options, the programmer can allocate areas of storage that are not contiguous, but are allocated separately and positioned on the proper halfword, word or double-word boundary. (See chart 3-12.)

Chart 3-12. Examples of Noncontiguous Assignments

◆

NAME	OPERATION	OPERAND
WORK1	DS	10C
WORK2	DS	40H
WORK3	DS	15F
	ORG	WORK1 + 100

CNOP Conditional No Operation

General Description

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

Format

◆ The format of the CNOP instruction is as follows:

NAME	OPERATION	OPERAND
Not Used	CNOP	Two decimal terms of the form b,w.

Specification Rules

Name Field

◆ Not used.

Operation Field

◆ CNOP.

Operand Field

◆ Two operands in the form of b,w where:

b specifies the byte in a word or double word in which the Location Counter is to be set. Values of 0,2,4, or 6 may be specified.

w specifies whether the byte b is a word (four bytes) or double word (eight bytes). The following pairs are valid combinations:

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

Assuming that the Location Counter is currently aligned at a double-word boundary, then the CNOP instruction in sequence given in chart 3-13 has no effect; it is merely printed in the assembly listing. However, the sequence given in chart 3-14 causes three branch-on-conditions (no-operations) to be generated, thus aligning the BALR instruction at the last halfword in a double word as given in chart 3-15.

Operand Field (Cont'd)

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

Figure 3-1. CNOP Alignment of Double Word Using (0,4,2,4)

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

Note

◆ The CNOP instruction ensures the alignment of the Location Counter setting to a halfword, 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.

Chart 3-13. Effect of CNOP Sequence

◆

NAME	OPERATION	OPERAND
	CNOP	0,8
	BALR	2,14

Chart 3-14. Effect of CNOP Sequence

◆

NAME	OPERATION	OPERAND
	CNOP	6,8
	BALR	2,14

Chart 3-15. CNOP Sequence Causing Branch-on-Condition

◆

NAME	OPERATION	OPERAND
	BCR	0,0
	BCR	0,0
	BCR	0,0
	BALR	2,14

EQU
Equate

General Description

◆ The EQU instruction is used to define a symbol by assigning to it the attributes of an expression specified in the Operand field.

Format

◆ The format of the EQU instruction is as follows:

NAME	OPERATION	OPERAND
A symbol	EQU	An expression.

Specification Rules

Name Field

◆ Any valid symbol.

Operation Field

◆ EQU.

Operand Field

◆ The expression may be absolute or relocatable. The symbols used in the expression must be previously defined.

In chart 3-16, the programmer chooses to equate General Register 2 to the symbol REG2 and to equate the hexadecimal term X'3F' to the symbol TEST. The expression ALPHA ⊖ BETA + GAMMA is computed by the Assembler and the value of the expression is assigned to the symbol FIELD.

Note

◆ Both name and operand entries are mandatory. The symbol used in the Name field is assigned the calculated value of the expression used in the Operand field and is assigned the length attribute of the leftmost (or only) term of the expression. The EQU controlling code is the only means of making a symbol absolute.

Chart 3-16. Example of EQU Instruction

NAME	OPERATION	OPERAND	COMMENTS
REG2	EQU	2	(GENERAL REGISTER)
TEST	EQU	X'3F'	(IMMEDIATE DATA)
FIELD	EQU	ALPHA-BETA+GAMMA	(VALUE OF FIELD WILL BE ALPHA+2.
	.		ADDRESS OF ALPHA ⊖ ADDRESS OF BETA +
	.		ADDRESS OF GAMMA. LENGTH OF ALPHA IS
ALPHA	DC	C'100'	GIVEN TO FIELD)
BETA	DC	C'20'	
GAMMA	DC	C'30'	

DEFINING CONSTANTS

◆ Data in the form of a character, hexadecimal, binary, decimal, fixed-point, or floating-point constant can be entered into a program through the use of the DC (Define Constant) instruction.

The format of the DC instruction is:

NAME	OPERATION	OPERAND
Symbol or blank.	DC	A single operand describing a constant or set of constants

These constants are classified as data constants or address constants. Data constants are enclosed in quote marks while address constants are enclosed in parentheses. Data constants are described in this section prior to address constants. Fixed- and floating-point data constants are described after the Character (C), Hexadecimal (X), Binary (B), and Decimal (P,Z) constants but before the address constants (A, Y, S, V).

Literals follow the same rules as constants; however, they may not be used with S-type address constants.

The following chart lists the types of constants that may be defined by the DC instruction.

CODE	USED TO GENERATE
C	Eight-bit code for each CHARACTER.
X	Four-bit code for each HEXADECIMAL digit.
B	One or more binary digits (BIT).
P	PACKED decimal digit, signed.
Z	ZONED decimal digit, unpacked.
F	Fixed-point binary value, signed 32-bit FULLWORD, implied.
H	Fixed-point binary value, signed 16-bit HALFWORD, implied.
E	Floating-point, Single precision 24-bit mantissa, 8-bit EXPONENT.
D	Floating-point, DOUBLE precision 56-bit mantissa, 8-bit exponent.
A	Binary address, fullword.
Y	Binary address, halfword.
S	Base-displacement address, halfword.
V	External symbol address, fullword reserved.

DC
Define Constant

General Description

◆ The DC (Define Constant) instruction is used to provide constant data in storage. One or more of a variety of constants may be specified in a single DC instruction.

Format

◆ The format of the DC instruction is as follows:

NAME	OPERATION	OPERAND
A symbol or blank.	DC	[D] [T] [X_n] 'constant'
		(constant)
		'constant, . . . , constant

Specification Rules

Name Field

◆ Contains a symbol or is left blank. If a symbol is used to name the constant it is assigned the leftmost byte address and the value attribute of the first, or only constant specified.

Operation Field

◆ DC.

Operand Filed

◆ Consists of three optional subfields preceding the constant subfield.

CONSTANT - enclosed by quotes or parentheses: 'constant' used with all data constants; (constant) used only with address constants; 'constant, . . . , constant' multiple data constants. The last form may not be used with C, X, or B type constants.

T = TYPE - specifies type of constant to be generated. If omitted, Character is assumed.

D = DUPLICATION - when specified, it causes the constant(s) to be duplicated D times after the constant has been generated. D must be specified as an unsigned decimal number.

X = L, S, or E - a Length, Scale, or Exponent modifier followed by a decimal number where:

Ln = defines explicitly the number of bytes assigned to the constant.

Sn = defines the scaling applicable to F, H, or E, D constants (see pages 3-22 and 3-25).

En = defines the preadjustment to F, H, or E, D constants (see pages 3-24 and 3-26).

Alignment of Constants

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

The total storage requirement of an operand is the product of the length times the number of constants in the operand times the duplication factor, plus any bytes skipped for alignment reasons.

Types of Constants

◆ The following description denotes the various types of constants, their descriptive features, and positioning within the Spectra 70 Assembly System.

Character Constants (C-Type)

◆ Any of the 256 punch combinations can be used in defining a character constant. Character constants may not exceed 256 bytes, are enclosed in single quotation marks, and are preceded by a letter C. Special attention should be given to the constant that requires the use of the quotation mark and ampersand. Only one character constant may be specified per operand, and no boundary alignment is performed on the assembled bytes. (See chart 3-17.)

If a length modifier is not specified, the length of the constant is implied by the constant itself. Each character is converted into an eight-bit byte and assigned a left-hand byte address to the symbol naming it. If a length modifier is specified that is less than or exceeds the stated constant, truncation or padding with blanks is performed starting at the *rightmost* end of the constant generated. (See chart 3-18.)

Chart 3-17 Constant Generation

◆

NAME	OPERATION	OPERAND	COMMENTS
K1	DC	C'TITLE PAGE'	Generates - TITLE PAGE
K2	DC	'CREDIT'	C-Type Code implied
K3	DC	C'O''CLOCK'	Generates - O'CLOCK

Chart 3-18. Constant Generation

◆

NAME	OPERATION	OPERAND	COMMENTS
K4	DC	CL5'TRUNCATE'	Generates - TRUNC
K5	DC	CL5'PAD'	Generates -PAD__

Chart 3-19. Defining Character Constants

NAME	OPERATION	OPERAND	COMMENTS
EOF	DC	C'END OF RUN'	Generates - END OF RUN
CON1	DC	3C'ABC'	Generates - ABCABCABC
CON2	DC	2CL5'AD'	Generates - AD_ _ _ AD_ _ _
CON3	DC	3CL4'ABCDEF'	Generates - ABCDABCDABCD

Hexadecimal Constants (X-Type)

◆ Hexadecimal constants are normally used in place of the character constant when one or more of the bytes cannot be expressed by a character value. Hexadecimal digits 0-9 and A-F are used to denote the constant. The constant is written as a series of hexadecimal digits, is enclosed in single quotation marks, and is preceded by an X.

The maximum number of hexadecimal digits may not exceed 512 (256 bytes). The hexadecimal digits, starting at the leftmost end of the constant are paired and used to generate the byte. If an odd-number of hexadecimal digits is specified, the leftmost byte has its leftmost bits filled with a hexadecimal zero and the rightmost byte contains the first digit. The implied length (if no length modifier is specified) is half the number of hexadecimal digits in the constant.

Truncation or padding occurs if a length modifier has understated or over-stated the constant storage area. Truncation and hexadecimal zero padding start at the *leftmost* end of the constant. (See chart 3-20.)

Chart 3-20. Defining Hexadecimal Constants

NAME	OPERATION	OPERAND	COMMENTS
TAGA	DC	X'40206B'	Generates - 40206B LENGTH IS 3
TAGB	DC	2XL3'A6F4E'	Generates - 0A6F4E0A6F4E PADDING
TAGC	DC	3XL2'A6F4E'	Generates - 6F4E6F4E6F4E TRUNCATION

Binary Constants (B-Type)

◆ Binary constants are written using the binary digits 1 and 0, enclosed in single quotation marks and preceded by a B. The maximum length of a binary constant is 256 bytes. The length modifier range is, as in all the constants previously mentioned, summarized in table 3-4. The implied length is the number of bytes including padding used to store the constant. Padding and truncation begins at the *leftmost* byte. Padding is with zeros. (See chart 3-21.)

Chart 3-21. Defining Binary Constants

NAME	OPERATION	OPERAND	COMMENTS
BCON	DC	B'11011101'	Generates - 11011101 LENGTH IS 1
BTRUNC	DC	BL1'100100011'	Generates - 00100011 TRUNCATION
BPAD	DC	BL1'101'	Generates - 00000101 PADDING
BDUP	DC	2BL1'11111111'	Generates - 1111111111111111

Packed Decimal Constants (P-Type)

◆ A decimal constant is written as a signed or unsigned decimal value. The absence of a sign causes a plus sign to be assumed. The decimal point may be written or omitted from the constant. The placement of the decimal point does not affect the assembly of the constant. Decimal point alignment is not performed by its use within the constant. Proper decimal point alignment is determined by the programmer before defining the data or by selecting instructions that will operate on the data properly. Boundary alignment is not performed. The maximum size of the decimal constant is 31 decimal digits and a sign.

Each pair of decimal digits is translated and stored in one byte. The rightmost byte contains the rightmost digit and sign. The plus sign is translated into the hexadecimal C and the minus sign into the hexadecimal D. (See chart 3-22.) The length attribute of the constant, if length modification is not specified, will be the number of bytes the constant occupies.

Note

◆ If an even number of packed decimal digits is stated, the leftmost byte is left unpaired and the unused bits are set to zero. The rightmost byte combines the last digit with the sign. Truncation or padding occurs when the length modifier and actual constant values disagree. Truncation or zero (00_{16}) padding occurs starting at the *leftmost* byte.

Chart 3-22. Example of Packed Decimal Constants

NAME	OPERATION	OPERAND	COMMENTS
TAX	DC	P'+1.25'	Generated CONSTANT 125C 2 BYTES
	DC	PL4'-0.5'	Generated CONSTANT - 0000005D 4 BYTES

Zoned Decimal Constants (Z-Type)

◆ In zoned decimal format (Z), each decimal digit is translated into one full byte (not paired). The rightmost byte contains the sign and the rightmost digit. The remaining rules for zoned decimal are identical to the packed decimal rules specified above. Padding is done with full bytes of decimal zeros ($F0_{16}$).

Zoned Decimal Constants (Z-Type) *(Cont'd)*

Chart 3-23. Example of Zoned Decimal Constants

NAME	OPERATION	OPERAND	COMMENTS
PRINT01	DC	ZL2'1'	Generated Constant - F0C1
ZEROS	DC	132ZL1'0'	Generates 132 bytes, Length of 1
BLANKS	DC	ZL132' '	Generates 132 bytes, Length of 132

Some coding illustrations of the previous types of constants used as literals in instructions are illustrated in chart 3-24.

Chart 3-24.

NAME	OPERATION	OPERAND
	MVC	FIELDX(5),=5C' '
	AP	FIELDY(3),= PL3'1'
	CLC	FIELDZ(6),=6X'0'
	XC	BINCODE(1),=B'111'
	PACK	MAXIMUM,=5ZL2'99'

Fixed-Point Constants (F-,H-Type)

◆ When the fixed-point arithmetic mode is selected, fixed-point binary data constants are specified by the F-type or the H-type DC.

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

Format

NAME	OPERATION	OPERAND
Symbol or blank	DC	[D] T [S±n E±n] 'constant [E±n]' 'series of constants'

where:

D = the Duplication factor.
T = fullword (F) or halfword (H).
S± = the Scale Modifier.
E± = the Exponent Modifier (preceding) or the Exponent of the constant (following).

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

Halfword or fullword alignment is performed unless an explicit length is specified. A length of two bytes for halfword or four bytes for fullword is implied unless an explicit length is stated. The explicit length may not exceed eight bytes.

Format (Cont'd)

The binary number occupies the rightmost portion of the field in which it is placed. The unoccupied portion (the leftmost bits) is filled with the sign. That is, the setting of the bit designating the sign is the setting for the bits in the unused portion of the field. If the value of the number exceeds the length, the necessary leftmost bits are dropped. A negative number is generated in 2's complement binary form as shown in chart 3-25.

Chart 3-25.

NAME	OPERATION	OPERAND
MINUS1	DC	F'-1' generates $FFFFFFFF_{\wedge 16}$

A mixed number such as 1.5 may be defined using a scale modifier as shown in chart 3-26.

Chart 3-26.

NAME	OPERATION	OPERAND
MIXS4	DC	HS4'1.5' generates $001_{\wedge}8_{16}$

When the scale modifier is omitted a binary integer is generated. (See chart 3-27.)

Chart 3-27.

◆

NAME	OPERATION	OPERAND
	DC	H'100' generates $0064_{\wedge 16}$

Scale Modifier

◆ The scale modifier specifies the power of 2 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. (See chart 3-28.)

Scale Modifier (Cont'd)

Chart 3-28.

NAME	OPERATION	OPERAND
MIXSF1	DC	HS1'1.5' generates $000000000000001_\wedge1_2$
MIXSF4	DC	HS4'1.5' generates $000000000001_\wedge1000_2$
MIXSF8	DC	HS8'1.5' generates $00000001_\wedge10000000_2$

A negative scale shifts the integral portion of the number right, thereby deleting rightmost integral positions. (See chart 3-29.)

Chart 3-29.

NAME	OPERATION	OPERAND
V1	DC	HS0'14' generates $000F_{\wedge 16}$
HALFV1	DC	HS-1'14' generates $0007_{\wedge 16}$
QTRV1	DC	HS-2'14' generates $0004_{\wedge 16}$

Where positions are lost because of scaling, rounding occurs in the leftmost bit of the lost portion. The rounding is reflected in the rightmost position saved.

Note:

If a scale modifier does not accompany a fixed-point constant containing a fractional part, the fractional part is lost and the closest integer is generated. (See chart 3-30.)

Chart 3-30.

NAME	OPERATION	OPERAND
	DC	F'1.5' generates $00000002_{\wedge 16}$
	DC	F'1.1' generates $00000001_{\wedge 16}$

To retain the fractional value a scale factor must be specified in the DC.

The decimal number may be adjusted by a power of ten <u>before</u> it is converted to binary form. This Exponent of the constant is specified by appending E with a positive or negative power of ten. (See chart 3-31.)

Chart 3-31.

NAME	OPERATION	OPERAND
	DC	H'0.4E1' generates $0004_{\wedge 16}$

This allows the fraction to be written as such, but generated as an integer.

Scale Modifier (Cont'd)

The exponent can be in the range -85 to +75. If an unsigned exponent is specified, a plus sign is assumed.

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}

When a series of binary constants are coded the exponent modifier and scaling option, if stated, apply to all the constants. (See chart 3-32.)

Chart 3-32.

NAME	OPERATION	OPERAND
	DC	HS4E1'1.5,2.5,3.5'

would adjust 1.5,2.5 and 3.5 by 10^1 and then the generated values would each be moved left four places to represent 15.0,25.0 and 35.0.

The Exponent modifier precedes the constant(s), but the Exponent of the constant pertains only to the constant it follows.

Chart 3-33.

NAME	OPERATION	OPERAND
	DC	FE2'44E5' means $44\text{x}10^5\text{x}10^2$

Floating-Point Constants (E-,D-Type)

◆ Floating-point constants are specified by the E-type and D-type constants for floating-point arithmetic.

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

Floating Point Constants (E-, D-Type) *(Cont'd)*

A floating-point constant is written as a decimal number, which is followed by a decimal exponent, if desired. The number can be an integer, a fraction, or a mixed number (that is, one with integral and fractional portions). The format of the constant is as follows:

Format

◆ NAME	OPERATION	OPERAND
Symbol or blank.	DC	D T Sn E ± n 'constant E ± n' 'series of constants'

where:

D = the Duplication factor.

T = E (single word) or D (double word).

Sn = the Scale Modifier.

E±n = the Exponent Modifier (preceding) the Exponent of the constant (following).

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

Chart 3-34.

◆ NAME	OPERATION	OPERAND
	DC	E'0.5' generates 40800000_{16}
	DC	E'5.0' generates 41500000_{16}

Scale Modifier

◆ When the scale modifier is omitted a normalized floating-point number is generated; that is, the fraction is not preceded by any hexadecimal zeros. (See chart 3-35.)

Chart 3-35.

NAME	OPERATION	OPERAND
	DC	E'0.1' generates $4019999A_{16}$

Only a positive scale modifier can be used with a floating-point constant. This modifier 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.) The point is assumed to be at the left of the leftmost position in the field. Because the point cannot be moved left, the fraction is shifted right and the exponent is adjusted to retain the correct magnetude. Thus, scaling that is specified for a floating-point constant provides an assembled fraction that is unnormalized; that is, contains hexadecimal zeros in the leftmost positions of the fraction. When hexadecimal positions are lost, rounding occurs in the leftmost hexadecimal position of the lost portion. The rounding is in the rightmost hexadecimal position saved.

Exponent Modifier

◆ This modifier denotes the power of 10 by which the constant is to be multiplied before its conversion to the proper internal format. The modifier is written as En where n is a decimal value. The decimal value may be preceded by a sign; if none is present, a plus sign is assumed. The maximum values for exponent modifiers are summarized in table 3-4.

Chart 3-36.

NAME	OPERATION	OPERAND
	DC	DE2'0.01' generates 4019999999999A

The same value can be obtained by the Exponent Modifier and the Exponent of the constant being specified as in chart 3-37.

Chart 3-37.

NAME	OPERATION	OPERAND
	D	DE1'0.01E1'

The exponent modifier is written immediately before the number as En, where n is an optionally signed decimal value specifying the exponent of the base 10. The exponent can be in the range -85 to +75. If an unsigned exponent is specified, a plus sign is assumed.

This modifier is not to be confused with the exponent of the constant itself. Both are denoted 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 can be specified with an exponent of +2, and an exponent modifier of +5 can precede the constant. In effect, the constant has an exponent of +7.

Note:

There is a maximum value for exponents, both positive and negative, listed in table 3-4. This applies both to exponent modifier and exponents specified as part of the constant, or to their sum if both are specified.

Any duplication factor that is present is applied after the constant is converted to its binary format and assembled into the proper number of bytes.

A field of three full words is generated from the statement in chart 3-38. The location assigned to CONWRD is the address of the leftmost byte of the first word, and the length attribute is four, the implied length for a fullword, fixed-point constant. The expression CONWRD + 4 could be used to address the second constant (second word) in the field.

Chart 3-38. F-Type Constant

NAME	OPERATION	OPERAND	COMMENTS
CONWRD	DC	3F'658474'	

Exponent Modifier (Cont'd)

In chart 3-39, the next constant (3.50) is multiplied by 10 to the -2 before being converted to its binary format. The scale modifier reserves eight bits for the fraction portion. The same constant could be specified as a literal. (See chart 3-40.)

Chart 3-39. H-Type Constant, Scaled for Eight Bits

NAME	OPERATION	OPERAND	COMMENTS
FULLCON	DC	HS8'3.50E-2'	

Chart 3-40. H-Type Constant as a Literal

NAME	OPERATION	OPERAND
	AH	7,=HS8'3.50E2'

Address Constants

◆ An address constant is a storage address that is translated into a constant. Address constants are normally used to initialize base registers (A-type), represent base-displacement addresses within instructions (S-type) or provide a means of transferring control between control sections of a multisection program (V-type). In addition, a Y-type address constant is provided to represent addresses in two bytes, halfword aligned.

Format

◆ The address constant is enclosed in parentheses with A, Y, S, or V preceding the left parentheses. There must be a separate statement line for each address constant. A-type and V-type constants are fullword aligned. Y- and S-type constants are halfword aligned.

A-Type Address Constant

◆ The A-type address constant provides a storage location (word oriented) for the assembly to store the value of a simple expression (symbol) or a calculated complex expression. The maximum value of the expression may not exceed 2^{31}-1 for the 70/35-45-55 Processors.

The implied length of the A-type constant is four bytes and is aligned on a fullword boundary. If length modifier notation is used, it will override normal fullword alignment. Length modifier specification depends on the type of expression generated. If the expression is absolute, a length of one to four bytes may be specified with the value placed in the rightmost portion. (See chart 3-41.)

An A-type constant may contain a reference to the Location Counter, which refers to the leftmost byte of the constant.

When a Location Counter reference occurs in a literal, the value of the Location Counter is the address of the first byte of the instruction. (See chart 3-42.)

Chart 3-41.

NAME	OPERATION	OPERAND
ADCON1	DC	A(STRT)
ADCON2	DC	A(8192)

Chart 3-42.

NAME	OPERATION	OPERAND
LC	DC	A(*)
	LM	4, 4, = A (* +4096)

Y-Type Address Constant

◆ The Y-type constant provides the storage facilities for a 16-bit address. The storage location is aligned on a halfword boundary and has an implied length of two bytes. Length specification may specify one byte or two bytes. The remaining characteristics of the Y-type constant are the same as the A-type constant mentioned above.

Complex Relocatable Expressions

◆ A complex relocatable expression can only be used to specify A-type or Y-type address constants. A complex relocatable expression occurs when two (or three) unpaired relocatable terms are combined. For example, if the relocatable symbol A is defined in CSECT1 and the relocatable symbol B is defined in CSECT2, the reference A + B is a complex relocatable expression.

In contrast to relocatable expressions, complex relocatable expressions may represent a negative value. The symbols A and B as described above could be expressed A - B. If B were larger a negative value would occur.

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.

Absolute or paired relocatable terms may be present in the expression containing unpaired relocatable terms or a negative relocatable term.

S-Type Address Constant

◆ S-Type address constants are used to store an address in base displacement format. S-type constants are assembled as halfword values and stored on halfword boundaries. The leftmost four bits of the constant are the register number and the remaining 12 bits are the displacement value. If length specification is used, only two bytes may be specified. The constant can be specified as an absolute or relocatable expression, or the constant expression is stated as two absolute terms, the first term representing the displacement and the second term representing the base register. (See chart 3-43.)

Chart 3-43. Example of Address Constants, S-Type

NAME	OPERATION	OPERAND	COMMENTS
ADCON1	DC	S(BETA)	GEN CON ADDRESS OF BETA
ADCON	DC	S(400(13))	GEN CON ADDRESS OF 400 AND GR13 IN BASE DISPL'T FORMAT

Note

◆ S-Type address constants may not be specified as literals.

If an S-type constant is specified as an EXTRN, a USING statement must be issued to provide the base register designation. (See EXTRN, page 4-16.)

V-Type Address Constants

♦ This constant reserves storage for the address of an external symbol that is used for branching to other programs (separately assembled control sections).

A V-type constant is aligned to a fullword boundary. The implied length is four bytes. A length modifier of three or four bytes may be specified, but boundary alignment does not occur.

The reserved word is set to zeros until the program containing the named symbol is bound. The symbol is specified as one relocatable symbol. Specifying a symbol as the operand of a V-type constant does not constitute a definition of the symbol for this assembly. Whatever symbol is used is assumed to be an external symbol because it is supplied in a V-type constant.

A V-type constant need not be identified by an EXTRN statement.

Note:

The constant cannot be used for external data references.

V-type constants provide a convenient method for linking to a separately assembled object module or control section. A V-type address constant is specified with the name of the external symbol as the operand. When control is to be transferred to the external object module, the constant value is loaded by the programmer into a general register and a branch to the control section desired is issued by means of the BALR instruction. (See chart 3-44.)

Chart 3-44. V-Type External Address Referencing

♦

```
NAME      OPERATION      OPERAND

MAIN        CSECT

BEGIN       BALR        2,0

            USING       *,2
              .
              .
              .
            L           3,=V(VECTORX1)

            BALR        1,3
              .
              .
              .
            END         BEGIN
```

Table 3-4. Summary of Constants

Type	Implied Length (Bytes)	Alignment	Length Modifier Range (Lm)	Specified by	Number of Constants per Operand	Exponent Modifier Range (Em)	Scale Modifier Range (Em)	Truncation/ Padding Side
C	as needed	byte	1 to 256	characters	one			right
X	as needed	byte	1 to 256	hexadecimal digits	one			left
B	as needed	byte	1 to 256	binary digits	one			left
F	4	word	1 to 8	decimal digits	multiple	-85 to +75	-187 to +346	left
H	2	halfword	1 to 8	decimal digits	multiple	-85 to +75	-187 to +346	left
E	4	word	1 to 8	decimal digits	multiple	-85 to +75	0 to 2L-2 (1)	right
D	8	double word	1 to 8	decimal digits	multiple	-85 to +75	0 to 2L-2 (1)	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	any expression	one			left
V	4	word	3 or 4	relocatable symbol	one			left
S	2	halfword	2 only	one absolute or relocatable expression or two absolute expressions: exp (exp)	one			left
Y	2	halfword	1 to 4	any expression	one			left

(1) L is length of constant. Negative scaling is not permitted.

4. PROGRAM STRUCTURE

CONTROL SECTIONS

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

For instance, a single control section may be defined by a series of statements preceded by a START or CSECT instruction and terminated by an END instruction. The output of the assembly consists of the assembled control section and a Control Dictionary.

To the Linkage Editor, there are no programs, only control sections or object modules that must be fashioned into an object program. The Control Dictionaries contain information needed by the Linkage Editor to complete cross-referencing between control sections so that they may be combined into an object program.

The Linkage Editor can take control sections from various assemblies and combine them properly with the help of the corresponding Control Dictionaries. Successful combination of separately assembled control sections depends on the techniques used by the programmer to provide symbolic linkages between the control sections.

Control Section Definition

◆ 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, which may be divided into sections called control sections. Control sections are written so 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, within the same assembly, without altering or impairing the operating logic of the program. It is normally identified by the CSECT assembly instruction. However, if it is desired to specify a tentative starting location, the START assembly instruction may be used to identify the first control section.

Sectioning a program is optional, and many programs can best be written without sectioning. The Assembly System, however, provides facilities for creating multisectioned programs, which can be assembled separately and linked at a later time into an object program.

Whether the programmer writes an unsectioned program, a multisectioned program, or part of a multisectioned program, eventually these sections will be entered into storage. Because storage has been defined symbolically, the exact location of each section may not be shown. 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. Sectioning is not synonymous with segmentation or overlay methods.

Control Section Definition (Cont'd)

Note:

The combined number of control sections and dummy sections may not exceed 32. The combined number of EXTRN and V-type address constants may not exceed 255.

Two or more control sections assembled together cannot define the same symbol. However, the symbol that appears as the name of a START or CSECT instruction may be used on a subsequent CSECT to designate the continuation of the CSECT. For instance:

```
NAME    OPERATION           OPERAND

PROG    START      FIRST CONTROL SECTION
          .
          .
          .
DATA    CSECT      SECOND CONTROL SECTION
          .
          .
          .
PROG    CSECT      CONTINUATION OF FIRST CSECT
          .
          .
          .
        END
```

Control section contents can be intermixed because the Assembly System provides a Location Counter for each control section. Locations are assigned to control sections so that the sections are placed in storage consecutively in the same order as they first occur in the program. Each control section after the first control section begins at the next available double-word boundary. For example, if Control Section 1 starts at location 1000 and is 98 bytes long, then the Location Counter for Control Section 2 is set to 1104. If Control Section 1 is resumed after Control Section 2, and the resumed part is 102 bytes long, then Control Section 2 will begin at location 1200 instead. Thus, the programmer may code data and program sequences as they are required, but still maintain them in distinctly assembled control sections.

First Control Section

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

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

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

START
Start Assembly

General Description

◆ The START instruction is used to specify a tentative starting location for the program. Only one START instruction is permitted in an assembly. The START instruction may be preceded by any type of assembly statement that does not affect or depend upon the setting of the Location Counter.

Format

◆ The format of the START instruction is as follows:

NAME	OPERATION	OPERAND
A symbol or blank.	START	A self-defining term or blank.

Specification Rules

Name Field

◆ If a symbol is specified in the Name Field it must be a valid relocatable symbol. The symbol represents the address of the first byte of the control section. Its length attribute is one. The control section is considered unnamed if the Name Field is left blank.

Note:

A control section that contains internal or external references must be named.

Operation Field

◆ START.

Operand Field

◆ The assembly uses the self-defining value specified by the operand as the tentative starting location of the program. This value must reference a double-word boundary. The operand field may be blank.

Chart 4-1. Examples of START Instruction

◆

NAME	OPERATION	OPERAND
	START	
	START	4096
	START	X'1000'
PROG2	START	
PROG2	START	8192
PROG2	START	X'2000'

END
End Assembly

General Description

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

Format

◆ The format of the END instruction is as follows:

NAME	OPERATION	OPERAND
Not Used	END	A relocatable expression or blank.

Specification Rules

Name Field

◆ Not used.

Operation Field

◆ END.

Operand Field

◆ Contains any expression whose value specifies the point to which control is transferred after loading the object program. If the operand is left blank, control is transferred to the first byte of the control section.

Chart 4-2. Example of END Instruction

◆

NAME	OPERATION	OPERAND
PRGNAM	START	
IO	DTFSR	...
	⋮	
ENTRY	BALR	4,0
	USING	*,4
NEXT	⋮	
	END	ENTRY

Note

◆ The operand may contain an external symbol, which must be a single-term, relocatable expression.

CSECT Identify Control Section

General Description

◆ The CSECT instruction identifies the beginning or the continuation of a control section. All statements that follow the CSECT instruction are assembled as part of that control section until another statement that identifies a different control section is encountered.

Format

◆ The format of the CSECT instruction is as follows:

NAME	OPERATION	OPERAND
A symbol or blank.	CSECT	Not used. Comments allowed.

Specification Rules

Name Field

◆ The symbol entered in the Name field establishes the name of the control section. If omitted, the section is considered to be unnamed. The symbol in the Name field must be a valid relocatable symbol the value of which represents the address of the first byte of the control section. It has a length attribute of one.

Several CSECT statements with the same name may appear within a program. The first statement 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. The Location Counter for each CSECT instruction is set to the next highest double-word boundary. However, the START card may be used to identify the first CSECT, and the START card may specify an initial value for its Location Counter. CSECT text becomes output in the same physical order as the input source.

Unnamed Control Section - If neither a named CSECT instruction nor START instruction appears at the beginning of the program, the assembly determines that it is to assemble an unnamed control section in a program. If one unnamed control section is initiated and is then followed by a named control section, any subsequent unnamed CSECT statements are considered to resume the unnamed control section.

Operation Field

◆ CSECT.

Example

◆ The example in chart 4-3 shows the use of the CSECT instruction in sectioning a program that consists of two sections. The first section is oriented to start at the setting 1600. Although the CSECT named BGN is split, note that the location assigned to NAME is 2400. The CSECT named SEG2 is assembled at location 4800 and terminates at 6000.

Chart 4-3. Example of CSECT Instruction

◆

NAME	OPERATION	OPERAND
BGN	START	1600
	.	
	.	800 BYTES
	.	
	.	
SEG2	CSECT	
OVER	.	
	.	1200 BYTES
	.	
	.	
BGN	CSECT	
NAME	.	
	.	2400 BYTES
	.	
	.	
	END	BGN

DSECT Identify Dummy Section

General Description

◆ A dummy section is 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.

Format

◆ The format of the DSECT instruction is as follows:

NAME	OPERATION	OPERAND
A symbol.	DSECT	Not used. Comments allowed.

Specification Rules

Name Field

◆ The symbol in the Name field establishes the name of the dummy control section and must be a valid relocatable symbol which represents the first byte of the dummy section. A length attribute of one is assigned.

Operation Field

◆ DSECT.

Additional Information

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

Dummy Section Location Assignment

◆ A Location Counter determines the relative locations of named program elements in a dummy section. The Location Counter is always set to zero at the beginning of the dummy section. The location values assigned to symbols that name statements in the dummy section relate to the initial statement in the section.

Note

◆ An address constant may contain a symbol that names a statement in a dummy section only if the symbol is paired (with the opposite sign) with another symbol from the same dummy section.

Addressing Dummy Systems

◆ The programmer may wish to describe the format of an area whose storage location is not 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 refer to the storage area, he does the following:

1. Provides a USING statement that specifies a general register, which the assembly can assign to the machine instructions as a base register, and that specifies a value from the dummy section, which the assembly assumes is contained in the base register.

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 relate to the initial statement of the section. Thus, all machine instructions that refer to names defined in the dummy section will, at execution time, refer to storage locations that relate to the address loaded into the register.

Example

◆ Assume that two independent assemblies (Assembly 1 and Assembly 2) are loaded and are to be executed as a single overall program. Assembly 1 is an input routine that places a record in a specified area of storage, places the address of the input area containing the record in General Register 3, and branches to Assembly 2. Assembly 2 processes the record. The coding shown in Chart 4-4 is from Assembly 2.

The input area is described in Assembly 2 by the DSECT control section named INAREA. Portions of the input area (that is, record) that the programmer wishes to work with are named in the DSECT control section as shown. The Assembly 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.

Chart 4-4. Example of DSECT Option

```
NAME     OPERATION OPERAND                    COMMENTS
ASMBLY2  CSECT
BEGIN    BALR  2,0
         USING *,2
         •
         •
         USING AREAIN,3              ASSURES CHOICE OF REGISTER 3
         NC    CODEIN,=C'A'                  FOR DSECT NAMES
         BE    ATYPE
         •
         •
ATYPE    MVC   WORKA,NPUTA
         MVC   WORKB,NPUTB
         •
         •
WORKA    DS    CL20
WORKB    DS    CL18
         •
         •
AREAIN   DSECT
CODEIN   DS    CL1
NPUTA    DS    CL20
NPUTB    DS    CL18
         •
         END
```

LTORG
Begin Literal Pool

General Description

◆ The LTORG instruction causes all literals thus far encountered in the source program to be assigned at appropriate boundaries starting at the first double-word boundary following the LTORG statement. Literals that are not collected by a LTORG statement are placed at the end of the first control section.

Format

◆ The format of the LTORG instruction is as follows:

NAME	OPERATION	OPERAND
Symbol or blank.	LTORG	Not used.

Specification Rules

Name Field

◆ Contains any symbol representing the first byte of the relocated literal pool. The symbol used to name the field is assigned a length attribute of one.

Operation Field

◆ LTORG.

Operand Field

◆ Not used.

Chart 4-5. Example of a LTORG Statement

◆
```
NAME      OPERATION      OPERAND
          START
BEGIN     BALR           2,0
          USING          *,2
            :
SECT2     CSECT
          BALR           3,0
          USING          *,3
            :
          L              4,=A(TABLE)
            :
          AP             COUNT,=PL1'1'
            :
          LTORG          ,NOTE THAT THIS LTORG
*STATEMENT ENSURES THAT THE ABOVE LITERALS ARE
*ASSEMBLED WITH THIS CONTROL SECTION
          END            BEGIN
```

Notes

◆ 1. Literals are listed and punched in the object program when the LTORG statement is encountered. Literals not covered by a LTORG statement are listed and punched when the END card is detected. In TOS/TDOS, the STMNT field on the listing shows the statement number which first specified a given literal.

2. Duplicate literals within a pool are punched and listed only once. However, if a literal is an address constant containing a reference to the Location Counter, a duplicate literal is generated.

3. If there are no LTORG statements in a program, 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 use 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. It is recommended that all programs using FCP contain a LTORG statement at the end of the user coding to ensure that all user literals are covered by a base register.

4. A maximum of 32 LTORG instructions may be specified.

COM Define Common Control Section

General Description

◆ The COM Assembler instruction identifies and reserves a common area of storage that may be referred to by independent assemblies that have been linked and loaded for execution as one overall program. The common area may be broken into subfields through the use of the DC and DS Assembler instructions. Names of subfields are defined relative to the beginning of the common section, as in the DSECT control section.

Format

◆ The format of the COM instruction is as follows:

NAME	OPERATION	OPERAND
Symbol or blank.	COM	Not used.

Specification Rules

Name Field

◆ Symbol or blank.

Operation Field

◆ COM.

Operand Field

◆ Not used.

Notes

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

2. When assembled, common location assignment starts on the next double-word boundary after the highest tentative location assigned to the assembly. If more than one common section is defined, the first is assigned as described above; the second common section starts on the next double-word boundary after the highest tentative location assigned to the first common; the third after the second, and so forth. Common control sections may be split.

Chart 4-6. Example of COM Instruction

NAME	OPERATION	OPERAND
MAIN	START	
BEGIN	BALR	12,0
	USING	*,12
	L	13, = A(COMAREA)
	USING	COMAREA,13
	LPOV	SECT1
	L	15, = A(SECT1)
	BALR	14,15
	TYPE	CODE,80
	TERM	
	EXTRN	SECT1
COMAREA	COM	
	DS	CL80
	END	BEGIN
SECT1	START	
	USING	*,15
	L	13, = A(COMAREA)
	USING	COMAREA,13
	MVC	LETTER,CODE
	MVC	CON1,ENTRY1
	MVC	CON2,ENTRY2
	MVI	ENTRY3,C'A'
	MVC	ENTRY3 + 1(28),ENTRY3
	BR	14
LETTER	DC	CL1'C'
CON1	DC	5CL5'12345'
CON2	DC	5CL5'ABCDE'
	DS	CL50
COMAREA	COM	
CODE	DS	CL1
ENTRY1	DS	CL25
ENTRY2	DS	CL25
ENTRY3	DS	CL29
	END	

PROGRAM LINKAGE CONTROLLING CODES

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

Note:

These symbolic linkages are described as linkages between independent assemblies; more specifically, they are linkages between *independently assembled* control sections.

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

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

Another means of obtaining symbolic linkage is by using the V-type address constant. It is created from an externally defined symbol, but that symbol need not be identified by an EXTRN statement.

Note

◆ The V-type address constant may be used for effecting branches to other programs. It may not be used for referring to data in other programs. For instance:

```
L       15,=V(symbol)

BALR    14,15
```

ENTRY Identify Entry Point Symbol

General Description

◆ The ENTRY instruction identifies a linkage symbol that is defined in this program but may be used by some other program.

Format

◆ The format of the ENTRY instruction is as follows:

NAME	OPERATION	OPERAND
Not used.	ENTRY	A relocatable symbol that also appears as a statement name.

Specification Rules

Name Field ◆ Not used.

Operation Field ◆ ENTRY.

Operand Field ◆ Contains any symbolic name to identify an entry point to the assembly.

Example ◆ (See chart 4-7.)

Notes

◆ 1. The symbol used in the Operand field may be used by other programs as operand entries.

2. ENTRY statements may not contain a symbol defined in an unnamed control section or a dummy section.

3. The name of a control section need not be identified by an ENTRY instruction when another program uses it as an entry point. The Assembly System will automatically include the section names in the Control Dictionary.

Chart 4-7. Example of ENTRY Instruction

```
NAME     OPERATION  OPERAND                      COMMENTS
*POSSIBLE ENTRY POINTS
MAIN     START      3ØØØ
         ENTRY      SINE
         ENTRY      COSINE
         OPEN       MASTER,REPORT,PUNCHOT        SEE FCP DESCRIPTIONS
          .
         GET        MASTER                       SEE FCP DESCRIPTIONS
SINE     TM         SENSE,X'F'
          .
          .
          .
          .
COSINE   STM        3,3,SAVE
          .
          .
          .
WRITE    PUT        MASTER,WORKA                 SEE FCP DESCRIPTIONS
         ENTRY      WRITE
MASTER   DS         CL8Ø                         ONE 8Ø-BYTE AREA
WORKA    DS         4ØCL2                        FORTY 2-BYTE FIELDS
SAVE     DC         A(Ø)                         ONE 4-BYTE FIELD ZERO FILLED
          .
          .
         END        MAIN
```

EXTRN Identify External Symbol

General Description

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

Format

◆ The format of the EXTRN instruction is as follows:

NAME	OPERATION	OPERAND
Not used.	EXTRN	A relocatable symbol.

Specification Rules

Name Field

◆ Not used.

Operation Field

◆ EXTRN.

Operand Field

◆ Contains any relocatable symbol defined in some other control section. It may not appear as the name of a statement in the section containing the EXTRN statement.

Examples

◆ In chart 4-8, Program A contains two Branch instructions that refer to a program called Calculation (chart 4-9). Calculation contains two Branch instructions that refer to Program A. The points of entry between between Program A and Calculation are described to the assembly by the EXTRN and ENTRY statements shown in charts 4-8 and 4-9. Program A will branch to Calculation at points named CALC1 and CALC2. The return points to Program A will be at points CONT and CONT2.

One method to reference externally defined areas is by using the EXTRN instruction to identify the external symbol, and by creating an A-type address constant from the symbol. The generated address constant is loaded into a base register and used for base register calculation of addresses.

The example in chart 4-10 shows address calculation for an externally defined area.

Notes

◆ 1. External symbols, when used in an expression, may not be paired. The assembly processes them as though they originated from different control sections.

2. A symbol may be redundantly defined to be external.

3. V-type address constants need not be defined by an EXTRN statement.

Chart 4-8. Program A

NAME	OPERATION	OPERAND
BEG	START	X'0BB8'
	ENTRY	CONT
	EXTRN	CALC1
	EXTRN	CALC2
	USING	CALC1,4
	USING	CALC2,5
	:	
	LM	4,4,EXT
	LM	5,5,EXT1
	:	
	B	CALC1
CONT	:	
	BAL	6,CALC2
CONT2	:	
EXT	DC	A(CALC1)
EXT1	DC	A(CALC2)
	:	
	END	BEG

Chart 4-9. Calculation

NAME	OPERATION	OPERAND
SUBRT	START	
	ENTRY	CALC1
	ENTRY	CALC2
	EXTRN	CONT
CALC1	MVC	
	USING	CONT,5
	LM	5,5,ACONT
	B	CONT
CALC2	AP	
	:	
	BALR	0,6
ACONT	DC	A(CONT)
	END	SUBRT

Chart 4-10. Data Reference from External Control Section

```
NAME        OPERATION   OPERAND

MAINPROG    CSECT
BEGIN       BALR        2,0
            USING       *,2
              :
            EXTRN       RATETBL
              :
            LM          4,4,RATEADDR
            USING       RATETBL,4
            A           3,RATETBL
              :
RATEADDR    DC          A(RATETBL)
            END         BEGIN
```

5. ADDITIONAL ASSEMBLY INSTRUCTIONS

LISTING CONTROLS

TITLE Identify Assembly Output

General Description

◆ The TITLE instruction is used to identify an assembly listing and assembly output cards.

Format

◆ The format for the TITLE instruction is as follows:

NAME	OPERATION	OPERAND
Name or Not used.	TITLE	A sequence of characters, enclosed in single quotation marks.

Specification Rules

Name Field

◆ One to four characters, or not used. Used for punching columns 73-76 of the output cards of the program except cards produced by the REPRO or PUNCH instructions. Only the first TITLE card of a program should have a name entry. Name fields on subsequent TITLE cards must be blank.

Operation Field

◆ TITLE.

Operand Field

◆ Contains the title of the program to be printed on the assembly listings. Maximum entry is 100 characters enclosed in single quotation marks.

Chart 5-1. Example of TITLE Instruction

◆

NAME	OPERATION	OPERAND
PA01	TITLE	'PAYROLL UPDATE RUN'

Notes

◆ 1. A program may contain more than one TITLE statement. Each statement provides the heading for the listing of the statements that follow it until another TITLE card is read.

2. Each TITLE card encountered after the first one causes a page change before the header is printed.

3. The additional title cards must not contain name entries. The first title card name will remain the constant value to be punched into the object cards (columns 73-76), and printed at the top of each assembly page.

4. In chart 5-1, PA01 is punched in columns 73-76 of all output cards (except REPRO or PUNCH) and the heading "PAYROLL UPDATE RUN" appears at the top of each page.

EJECT Start New Page

General Description

◆ 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.

Format

◆ The format for the EJECT instruction is as follows:

NAME	OPERATION	OPERAND
Not used.	EJECT	Not used; should be blank, but will be treated as a comment.

Specification Rules

Name Field

◆ Not used.

Operation Field

◆ EJECT.

Operand Field

◆ Blank.

Note

◆ If the next line of the listing normally appears at the top of a new page, the EJECT statement has no effect.

SPACE Space Listing

General Description

◆ The SPACE instruction is used to insert one or more blank lines in the listing.

Format

◆

NAME	OPERATION	OPERAND
Blank	SPACE	A decimal value or Blank.

Specification Rules

Name Field

◆ Blank.

Operation Field

◆ SPACE.

Operand Field

◆ Contains a decimal value up to 15 that 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 the value exceeds the number of lines remaining on the listing page, the statement will have the same effect as an EJECT statement.

Chart 5-2. Example

◆

NAME	OPERATION	OPERAND
	SPACE	4

PRINT
Print Optional Data

General Description

◆ The PRINT instruction controls printing of the assembly listing.

Format

◆ The format of the PRINT instruction is as follows:

NAME	OPERATION	OPERAND
Not used.	PRINT	One to four operands.

Specification Rules

Name Field

◆ Not used.

Operation Field

◆ PRINT.

Operand Field

◆ One or all of the following terms can be used in the operand field:

<u>SINGLE</u> - Text listing is single spaced.

DOUBLE - Text listing is double spaced.

<u>ON</u> - A listing is printed.

OFF - No listing is printed.

<u>GEN</u> - All statements generated by macro instructions are printed.

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

DATA - Constants are printed in full in the listing.

<u>NODATA</u> - Up to 8 bytes (16 hexadecimal digits) of the first constant, whichever is shorter, of the assembled data is printed on the listing.

<u>DECK</u> - Resume punching of the object program if object program output was specified.

NODECK - Inhibit punching of the object program. (Note: in TOS this will inhibit tape and/or card output.)

<u>NUM</u> - Print the card number of the various object program card types. The card number is printed as a separate line when the card is punched. (TOS/TDOS.)

NONUM - Inhibit printing the card number of the various card types. (TOS/TDOS.)

(Note: NUM and NONUM are accepted by the POS Assembler but do not have any effect on the listing.)

<u>OPEN</u> - Cross reference listing is double spaced (TOS/TDOS).

CLOSED - Cross reference listing is single spaced (TOS/TDOS).

Note:

Underlined options are the preset conditions.

Examples

◆ Until the first PRINT statement is encountered, the statement in chart 5-3 is assumed. For example, if the statement in chart 5-4 appears in a program, 256 bytes of zeros are assembled. If the statement in chart 5-5 is the last PRINT statement to appear before the DC statement, all 256 bytes of zeros are printed in the assembly listing. However, if the statement in chart 5-6 is the last PRINT statement to appear before the DC statement, only eight bytes of zeros are printed in the assembly listing.

Chart 5-3.

◆ NAME	OPERATION	OPERAND
	PRINT	ON,NODATA,GEN,SINGLE,DECK,NUM

Chart 5-4.

◆ NAME	OPERATION	OPERAND
	DC	XL256'00'

Chart 5-5.

◆ NAME	OPERATION	OPERAND
	PRINT	DATA

Chart 5-6.

◆ NAME	OPERATION	OPERAND
	PRINT	NODATA

Note

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

AOPTN Assembler Option*

General Description

◆ The AOPTN instruction is used to control the normal outputs of the Assembler.

Format

◆ The format of the AOPTN instruction is as follows:

NAME	OPERATION	OPERAND
Not used.	AOPTN	One or more of the specified options, separated by commas.

Specification Rules

Name Field

◆ Not used.

Operation Field

◆ AOPTN.

Operand Field

◆ Each of the following options may be specified in separate AOPTN cards or appear as multiple operands (separated by commas) in a single card.

NODECK - The object program (ESD, TEXT, and RLD data) will not be produced on cards or tape. (This does not affect their appearance on the Listing.)

NOESD - External Symbol Dictionary cards will not be produced in the object program or on the Listing.

NORLD - Relocatable control cards will not appear in the object program.

NOLIST - Program listing will not be produced; however, statements containing errors will be listed.

*Valid on POS only. AOPTN functions are performed by Monitor PARAM message on TOS and TDOS.

Operand Field (Cont'd)

NOERR - Error flags will not be printed on the program listing, but a statement indicating the number of errors will be listed.

NOSYM - The symbol table will not be listed.

IPL - The IPL loader will be included in the object program preceding the ESD data (POS only).

LITERAL - This option is ignored, and literals may be used without specifying the option.

ENTRY - An entry card will be produced following the output card that is generated for the End statement. This option is required by the POS Linkage Editor.

Notes

◆ 1. If NOLIST and NOERR are specified there is no need to specify a listing device.

2. Any number of AOPTN cards may be specified; there is no restriction as to their order or placement within the source program.

3. AOPTN cards may be used to specify options separately or in combination.

PROGRAM CONTROLS

ICTL
Input Format Control

General Description

◆ The ICTL instruction allows the programmer to alter the normal format of his source program statements. The ICTL statement may be used as often as desired. The fields must be in the sequence: Name, Operation, Operand. Each must be separated by one or more blanks.

Format

◆ The format of the ICTL instruction is as follows:

NAME	OPERATION	OPERAND
Not used.	ICTL	1-3 decimal values of the form b, e, c.

Specification Rules

Name Field

◆ Not used.

Operation Field

◆ ICTL code.

Operand Field

◆ Contains one to three decimal values in the format b, e, c.

b specifies the begin column of the source statement. This value must always be used. Operand b must be less than c.

e specifies the end column of the source statement. If omitted, column 71 is assumed to be the end of the statement line. Operand e must be less than or equal to 80.

c specifies the continuation column of the source statement. If the continue column is not specified, or if column 80 is specified as the end column, the assembly assumes no continuation cards (all statements must be contained on a single card). Operand c must be less than e.

Example

◆ The example in chart 5-7 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.

Chart 5-7. Example of ICTL Instruction

◆

NAME	OPERATION	OPERAND
	ICTL	25

Notes

◆ If the ICTL statement is omitted in the source program, the assembly assumes a statement line is contained in columns 1-71 and that continuation lines begin in column 16. Any number of ICTL statements may be used in an assembly.

The first ICTL must conform to standard Assembler format as opposed to the format described by the statement. Succeeding ICTL statements must conform to the format of the ICTL currently in effect.

ISEQ Input Sequence Checking

General Description

◆ The ISEQ instruction checks the sequence of source input cards.

Format

◆ The format of the ISEQ is as follows:

NAME	OPERATION	OPERAND
Not used.	ISEQ	Two decimal values of the form L,R; or blank.

Specification Rules

Name Field

◆ Not used.

Operation Field

◆ ISEQ code.

Operand Field

◆ Contains two decimal values in the form L,R.

L specifies the leftmost column of the input card to be checked.

R specifies the rightmost column of the input card to be checked.

Notes

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

2. Any ISEQ with a blank operand terminates the operation. Checking can be resumed with another ISEQ statement.

3. Statements generated by macros are not included in the sequence check. (Source deck macro definitions will be checked.)

4. Operand L must be greater than the end column plus one.

5. Operand R must be equal to or greater than L.

6. The maximum value of R-L is seven; this is a maximum field size of eight bytes.

REPRO Reproduce Following Card

General Description

◆ The REPRO instruction allows the inclusion of Linkage Editor phase definition cards into the object program deck (module) to eliminate the necessity of manually inserting them.

Format

◆ The format of the REPRO instruction is as follows:

NAME	OPERATION	OPERAND
Not used.	REPRO	Blank or any operand for comments.

Specification Rules

Name Field

◆ Not used.

Operation Field

◆ REPRO.

Operand Field

◆ Blank.

Chart 5-8. Example of Stacked Assemblies - Separately Bound (TOS/TDOS)

◆
```
//_STARTM
     Insert ASSIGNS
//_JOB_STACK
     Insert Assembly options (PARAM)
//_ASSMBL
     REPRO
_PROG_ASSY1
     START
       .
       .
       .
     END
     REPRO
_PROG_ASSY2
     START
       .
       .
       .
     END
// LNKEDT
// ENDMON
```

Notes

◆ 1. REPRO causes a duplicate (80-80 card format) of the card immediately following.

2. Reproduced cards resulting from REPRO instructions appear at the same point in the object as they were in the source deck.

3. If a REPRO instruction precedes the START instruction or the implied START instruction, the cards reproduced will precede the ESD cards for the assembly.

4. In TOS, MONITOR control cards cannot be reproduced by the REPRO Statement.

PUNCH
Punch a Card

General Description

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

Format

♦ The format of the PUNCH instruction is as follows:

NAME	OPERATION	OPERAND
Not used.	PUNCH	80-character maximum self-defining term.

Specification Rules

Name Field

♦ Not used.

Operation Field

♦ PUNCH code.

Operand Field

♦ A character self-defining term of 80 characters maximum enclosed in single quotation marks.

Chart 5-9. Example of PUNCH Instruction

♦

NAME	OPERATION	OPERAND
	PUNCH	'ABCDEFG'

Notes

♦ 1. The position immediately to the right of the left quotation mark is regarded as column one of the card to be punched.

2. The assembly does not process the data in the Operand field other than to punch it.

3. The punched cards appear at the same point in the assembled text as they appeared in the source program.

4. The main difference between the PUNCH instruction and the REPRO instruction is the capability of the macro generator to substitute values for symbolic parameters or to set variable symbols in the operand of a punch instruction appearing in a macro definition. (This allows such things as controlled generation of phase names.)

5. If the PUNCH card precedes the START card, the punched cards will precede the ESD cards of the assembly.

XFR Generate a Transfer Card*

General Description

◆ A transfer card is used by the Loader and Linkage Editor routines to define the transfer point or entry point of a phase or overlay. The XFR assembly instruction causes the generation of a transfer card in the assembled text in the same location that the XFR instruction appeared in the source program.

Format

◆ The format of the XFR instruction is as follows:

NAME	OPERATION	OPERAND
Not used.	XFR	A relocatable symbol.

Specification Rules

Name Field

◆ Not used.

Operation Field

◆ XFR code.

Operand Field

◆ Any predefined symbol from within the assembly or defined as an ENTRY or EXTRN point.

Example

◆ See Appendix H Overlay Methods.

*Valid on POS only. This card is flagged, but produced on TOS/TDOS.

MCALL Macro Call

General Description

◆ The optional instruction MCALL permits the specifying of any or all macros required by a program. Inasmuch as macros are normally retrieved from the macro library in the order in which they are called, this feature eliminates access to the Library on an "as needed" basis.

Format

◆ The format of the MCALL is as follows:

NAME	OPERATION	OPERAND
Not used.	MCALL	Symbols separated by commas.

Specification Rules

Name Field

◆ Not used.

Operation Field

◆ MCALL.

Operand Field

◆ Symbols separated by commas specifying the macros to be called from the macro library.

Notes

◆ 1. If the macro has been previously specified in a prior MCALL statement, defined as a source-deck macro, or already called, the symbol is ignored.

2. Any number of MCALL statements may be specified and the statement is allowed in a macro definition (that is, as a model line).

3. After the macro definition is retrieved from the library, it is encoded into a form which requires less memory. The encoded macro is retained in memory or placed on a work tape if sufficient memory is not available.

The MCALL verb gives the programmer the capability to accomplish the following:

a. To specify the macros that should first be placed into HSM if space exists.

b. To specify the order in which the macros should be placed on the work tape.

c. To reduce substantially the search time required to fetch macros from the library tape. Note that each macro is called only once from the library tape.

Notes *(Cont'd)*

4. Macros are retained on the macro library in four priority groups. The macros which are specified in the MCALL operand field are retrieved from the tape in the order in which the macros appear on the tape, not necessarily in the order they were specified.

Example

♦

OPERATION	OPERAND
MCALL	P, B, G, X, H, A

Assume macros X and H are in priority group 1; and that P, B, and A are in priority group 2, and that G is in priority group 3.

The macros are called from the library tape in the following order:

H, X, A, B, P, G. After macro G has been retrieved, tape searching terminates, since no further priority 3 or any priority 4 macros are specified.

MPRTY Macro Priority

General Description

◆ This instruction allows the user to specify which priority groups of macros, when called, are encoded and placed in memory and/or on a work tape when sufficient memory does not exist.

The statement may be issued as often as desired to control this process.

Format

◆ The format of the MPRTY is as follows:

NAME	OPERATION	OPERAND
Not used.	MPRTY	Combination of four 0's and 1's.

Specification Rules

Name Field

◆ Not used.

Operation Field

◆ MPRTY.

Operand Field

◆ Combination of four 0's and 1's, that refer to priority groups 1 to 4, respectively, from left to right.

Note

◆ The Assembler presets the MPRTY indicator to 1100. Macros specified by MCALL are always encoded regardless of the MPRTY indicator setting for the macros' priority group.

Example

◆

OPERATION	OPERAND
MPRTY	1110

If a macro is called which is in priority groups 1, 2, or 3, it is encoded prior to expansion. If macro X in priority group 4 is called, it is not encoded but is expanded (in its definition format) from the library. Subsequent calls for X result in its retrieval and expansion from the library rather than directly from memory or the work tape.

6. INTRODUCTION TO SPECTRA 70 MACRO LANGUAGE

◆ The Spectra 70 macro language is a facility of the Spectra 70 Assembly System by which the programmer can generate standardized coding. Some advantages of the macro language are:

1. Program coding is simplified;
2. Functional coding may be standardized;
3. Coding errors may be reduced;
4. Macro definitions can be easily maintained;
5. Simple or tailored macros can be written.

Macros are defined, called, and generated (also referred to as "expanded"). The macro definition is written only once, and a single macro call statement is written each time the programmer wants to generate the desired sequence of Assembler language statements.

Note:

Macro call statements also are referred to as "macro call(s)" or "macro call line(s)" in this manual.

MACRO DEFINITION

◆ The macro is defined by a series of statements which include:

1. The macro header statement (MACRO) - start of macro definition;
2. The macro prototype statement - gives the mnemonic operation code (that is, TERMS in chart 6-1) and format in which the macro call statement will appear (see chart 6-2):
3. The macro model statements - stating the sequence of statements to be generated when the macro mnemonic (that is, TERMS) is called;
4. The macro trailer statement (MEND) - end of macro definition.

Macro definitions can be incorporated in a program at assembly time in two ways:

1. Source deck - Macros are available only in the source program in which the definition appears;
2. Macro Library - Tape or random access facility of entering macro definitions (RCA and/or user), which may be used in any source program (see Utility Manuals).

Note:

A macro definition must be available or defined before any call is made for the macro. (See Section 7 - Writing Macro Definitions.)

Macro Definition Structure

◆ Every macro definition must contain a minimum of four statements. They are:

1. A header statement (MACRO);
2. The prototype statement;
3. One or more model statements; and
4. A trailer statement (MEND).

The following command statements are optional for macro generation:

1. Set and Conditional Commands (Section 9);
2. MEXIT and MNOTE (Section 10);
3. MTRAC and NTRAC (Section 10).

TYPES OF MACROS

◆ Spectra 70 macro language permits macros to be written in either positional or keyword format. Both the macro prototype and its associated call statements must be of the same format. The only difference between the keyword and the positional macro is in the format of the prototype (and associated macro call) statements.

Positional Macros

◆ A positional macro requires that the prototype and call statements be written in a fixed format.

Parameters in the prototype statement and values in the call line are said to be "positionally significant" and are separated by a comma (,).

Notes:

Omission of a positional value must be indicated by an extra comma (,). For example:

NAME	OPERATION	OPERAND
	DTYPE	DEVICE , , STORE (Macro Call)

The second value has been omitted (signified by , ,). See Sections 7 - Writing Macro Definitions, and 8 - Macro Call Statements.

Chart 6-1. Example of a Positional Macro

NAME	OPERATION	OPERAND	
	MACRO		(Macro Header)
&NAME	TERMS	&PROG	(Prototype Statement)
&NAME	SVC	28	(Model Statements)
	SVC	10	
	DC	CL6'&PROG	
	MEND		(Trailer Statement)

Notes

◆ 1. "MACRO" signifies the start of any macro definition; "MEND", the end.

2. "&NAME" is a symbolic parameter; that is, a variable symbol.

Keyword Macros

◆ Keyword macros allow the keyword values to be written in a random order or omitted, because each value is associated with a keyword. Standard values, unless overridden by the programmer, may be inserted automatically. (See Section 11.)

Note

◆ The DTFSR macro is a keyword macro. The lack of a READ= value will cause FORWARD to be inserted from the prototype. (See Section 11 - Keyword Macros.)

MACRO CALL STATEMENT

◆ A macro call is a statement which causes the assembly macro generator to insert the macro's model statements at the point of the macro call.

The macro call may exist as a user's source statement or it may be one of a macro's model statements. The latter is an inner macro call and is generated when it appears.

Chart 6-2. Example of a Positional Macro Call Statement

◆

NAME	OPERATION	OPERAND	
ENTRY	TERMS	PROGB	(Macro Call)

Note

◆ See Section 8 - Macro Call Statements.

Chart 6-3. Generated Statements

◆ Assembler results of charts 6-1 and 6-2:

... LOCTN	OBJECT CODE	M	SOURCE	STATEMENT	
			ENTRY	TERMS	PROGB
00000	0A 1C	M1	ENTRY	SVC	28
00002	0A 0A	M1		SVC	10
00004	D7D9D6C7C240	M1		DC	CL6'PROGB'

VARIABLE SYMBOLS

◆ A variable symbol is an assembly symbol representing varying values which may be assigned, changed, or tested at any time during macro generation, by the programmer and/or the Assembler. Current values are examined to determine what model statements are to be generated.

Types of Variables

◆ Variable symbols can be:

1. Symbolic parameters;
2. System variable symbols (Section 10); or
3. Set variable symbols (Section 9).

Valid Symbols

◆ A variable symbol is written as a ampersand (&) followed by one to seven alphabetic and/or numeric characters, the first of which must be alphabetic. The dollar sign ($), the commercial at sign (@), and the number sign (#) are considered to be valid alphagetic characters.

Chart 6-4. Examples of Variable Symbols

◆

VARIABLE SYMBOL	TYPE OF SYMBOL
&NAME	Symbolic Parameter
&FROM2	Symbolic Parameter
&SYSNDX	System Variable
&SYSECT	System Variable
&BG2	SETB Symbol
&CG3	SETC Symbol
&AL1	SETA Symbol

Note

◆ The types of variable symbols illustrated in chart 6-4 are explained under the appropriate topic.

SYMBOLIC PARAMETERS

◆ A symbolic parameter is a type of variable symbol that is assigned values by the programmer when he writes a macro call statement. (See Section 8 - Macro Call Statements.)

The programmer may vary statements that are generated for each occurance of a macro call by varying the values assigned to symbolic parameters.

Examples

◆

NAME	OPERATION	OPERAND	
&NAME	MOVE	&FROM,&TO	PROTOTYPE
FIRST	MOVE	FIELD,WORK	CALL #1
SECOND	MOVE	RECORD,STORE	CALL #2

In Call #1 above, the symbolic parameters &NAME, &FROM, and &TO have been given the following values: FIRST, FIELD, and WORK respectively as a result of the positional call line. In Call #2, &NAME is SECOND, &FROM is RECORD; and &TO is STORE.

Restrictions for Symbolic Parameters

◆ The programmer cannot use any symbolic parameters that have &SYS as the first four characters. Further, symbolic parameters in the form &ALn, &AGn, &BLn, &BGn, and &CGn, where n is from one to five numeric characters, cannot be used. Symbols of these types are reserved for internal use. (See Section 9.)

Examples of Valid Symbolic Parameters

◆ The following are valid symbolic parameters:

&READER	&LOOP2	&TAG
&A23456	&N	&BLC
&X4#F2	&S4	&FROM

VARYING THE GENERATION

◆ The same sequence of generated statements is used from the macro definition in the absence of any Conditional macro generator commands. Thus, Conditional commands are used, usually in conjunction with Set commands, to vary the number and structure of the generated statements.

Note:

See Section 9 - Set and Conditional Macro Commands.

SECTIONING OF MACRO LANGUAGE INFORMATION

◆ The Spectra 70 macro language portion of this manual is further divided into the following sections:

7. WRITING MACRO DEFINITIONS

♦ To call a macro by means of a macro call statement, the macro must be previously defined. The programmer defines a macro by writing the instruction statements in a special macro definition language. This section discusses this definition language for positional macros. Keyword macros will be discussed in Section 11.

The programmer makes a macro definition available to many programs by placing the definition in the macro library. Macro definitions in the macro library can be inserted, deleted or replaced according to the needs of the programmer (see Utility Manuals).

MACRO DEFINITION CONTENTS

♦ A macro definition consists of the following types of statements (see chart 6-1, page 6-2).

HEADER STATEMENT (MACRO) - This statement indicates the beginning of a macro definition.

PROTOTYPE STATEMENT - This statement defines the format and mnemonic operation code of the macro call statement. Because the parameters defined in prototype statements must be general, the entries are referred to as symbolic parameters (see Section 6). The format of the prototype statement is the only difference between a positional macro definition and a keyword macro definition (see Section 11).

MODEL STATEMENTS - The model statements are comprised of machine instructions and/or assembly commands. The Operand fields of the model statements can contain symbols defined in source programs or symbolic parameters incorporated in the macro definition. The symbolic entries are, in turn, replaced by the values they represent. The symbolic entries can be symbolic parameters (see Section 6) or other variable symbols that are described in Sections 9 and 10.

TRAILER STATEMENT (MEND) -This statement indicates the end of a macro definition.

Notes

♦ 1. In writing all macro definitions, the begin column is column 1, the end column is column 71, the continue indicator column is column 72, and the continuation column is column 16.

2. The number of macro definitions transcribed to memory and/or the work tape during assembly by MCALL statements, source deck definitions, or by calling and the proper MPRTY switch = 1 is limited to 50 in POS. The TOS and TDOS limit is 75.

3. If sequence checking of the source deck is specified, the macro definition is not included. When the macro definition is terminated, checking will be resumed if it was in effect before encountering the macro definition.

MACRO Header Statement

General Description

◆ The macro definition header statement indicates to the assembler that a macro definition follows. It must be the first statement in every positional or keyword macro definition.

Format

◆ The format of the MACRO header statement is as follows:

NAME	OPERATION	OPERAND
Not used.	MACRO	See Operand Field (below).

Specifications Rules

Name Field

◆ Not used.

Operation Field

◆ MACRO.

Operand Field

◆ Although not scanned by the Assembler, certain Macro Library Update utility programs require the following information to appear in the Operand field:

VERnnn mm/dd/yy

where:

nnn = version number,

mm = month of version,

dd = day of version,

yy = year of version.

Note

◆ See appropriate Utility Routine reference manual for the Operating System being used.

MEND Trailer Statement

General Description

♦ This statement signifies to the Assembler that the macro definition is complete. It must appear as the last coding line of a macro definition.

Format

♦ The format for the trailer statement is as follows:

NAME	OPERATION	OPERAND
A sequence symbol or blank.	MEND	Not used.

Specification Rules

Name Field

♦ A sequence symbol consists of a period followed by a maximum of seven alphabetic and/or numeric characters, the first of which must be alphabetic. Sequence symbols are discussed in detail in Section 9.

Operation Field

♦ MEND.

Operand Field

♦ Not used.

Positional Prototype Statement

General Description

◆ The positional macro prototype statement must be the second statement of a macro definition. It specifies the mnemonic operation code and format of the positional macro operand. The values contained in the macro call statement will be substituted, on a positional basis, for the symbolic parameters specified in the prototype statement. The prototype statement is written in a format similar to other Assembly Language statements. The Name field, if used, must start in the begin column and must appear on the same card as the Operation field, and is followed by at least one blank.

Format

◆ The format is as follows:

NAME	OPERATION	OPERAND
A symbolic parameter or blank.	A Symbol.	Comma (,) or a maximum of 49 symbolic parameters, separated by commas.

Specifications Rules

Name Field

◆ A symbolic parameter or blank. The symbolic parameter is normally used to produce a label in the generated coding. See Section 6 for discussion of symbolic parameters.

Note:

The parameter associated with the Name field is numbered zero (0).

Operation Field

◆ The symbol in the Operation Field must appear in every macro call statement referred to this macro definition. The mnemonic operation code is a maximum of five alphabetic and/or numeric characters, the first of which must be alphabetic. The symbol must not be the same as the mnemonic operation code of a machine instruction, Assembler command, or macro generator command.

Notes:

1. Source deck definitions override identically named macro library definitions, which, once discarded, cannot be recalled during this program, but must be redefined if the discarded definition is needed.
2. The last source deck definition has precedence in case of conflict.

Operand Field

◆ The Operand field may contain a maximum of 49 symbolic parameters that positionally correspond to values submitted by the programmer by means of the macro call statement. To allow for a maximum of 49 symbolic parameters, as many continuation cards as required may be used. However, a line cannot be continued on the next card unless the Operand field of the line to be continued extends through column 71, with no embedded spaces, and column 72 does not contain a space.

Notes:

1. The absence of any parameters in the Operand field is indicated by an initial comma (,) followed by at least one blank. Comments may then follow. If there are neither symbolic parameters nor comments, no entry is required.

2. Symbolic parameters in the Operand field are numbered 1-49. Parameter 0 (Name field) plus 49 parameters (Operand field) gives a maximum total of 50 parameters for any prototype statement.

Examples

◆ Chart 7-1 is an example of a macro prototype that contains three symbolic parameters: one in the Name field and two in the Operand field. The mnemonic operation code is MOVE.

Chart 7-1. Positional Prototype

NAME	OPERATION	OPERAND
&NAME	MOVE	&TO,&FROM

Chart 7-2 shows the portion of the MOVE macro definition thus far discussed.

Chart 7-2. Macro Header, Prototype, and Trailer

NAME	OPERATION	OPERAND	
	MACRO		HEADER
&NAME	MOVE	&TO,&FROM	PROTOTYPE
	.		
	.		
	.		
	MEND		TRAILER

Model Statements

General Description

◆ Model statements are representations of the statements that replace the macro call in the object program. A model statement that contains no symbolic parameters or variable symbols appears in the source program in the same format as it appears in the macro definition. If the model statement contains symbolic parameters or variable symbols, they are replaced by their values when the model statement is expanded and inserted into the object program.

Any symbolic parameter appearing in a model statement must be defined in the prototype of the macro definition.

Specification Rules

◆ One or more model statements must follow the macro prototype statement. A model statement consists of from two to four fields (from left-to-right): Name field, Operation field, Operand field, and Comments field. These fields are written in standard Spectra 70 Assembly Language format as defined in Section 2.

Name Field

◆ Contains a symbol, symbolic parameter, sequence symbol, or blank.

Operation Field

◆ 1. Contains machine or Assembler mnemonic operation code, except START, END, ISEQ, and ICTL.

2. Contains symbolic parameter (see note 1).

Operand Field

◆ Symbols, symbolic parameters, other variable symbols, and other combination of characters (see note 2).

Comments

◆ Any combination of characters preceded by at least one blank (see note 3).

Notes

◆ 1. Variable symbols cannot be used to generate:

a. macro generator commands;

b. mnemonics which do not begin with a letter;

c. mnemonics larger than five characters;

d. START, END, ISEQ, or ICTL op codes.

2. The Operand field of all model lines (except an inner macro) must be completed through the "end" column before a continuation line is specified. A model statement can be continued on as many cards as necessary. The maximum number of characters permitted in the Operand field of a generated model statement is 112. However, if the model line is an inner macro instruction, the expanded Operand field may be as large as necessary.

3. Variable symbols appearing in the Comments field, are not replaced with their corresponding macro call values.

Notes *(Cont'd)*

4. The card following a REPRO model statement is not scanned by the macro generator, but merely reproduced.

5. Symbolic parameters used in a model statement must be defined in the prototype statement.

6. Symbols used in a model statement must be defined in the macro definition or within the source program.

7. Two ampersand signs (&&) or quotes ('') must be used to represent a single ampersand (&) or quote (') in a character value or self-defining value. (See chart 7-1, page 7-5, and chart 7-7, page 7-8.)

Examples

◆ The following set of charts illustrate the macro definition of MOVE, the calling of MOVE, and resultant generated assembly statements. The macro MOVE allows the programmer to move two separate fields, with associated lengths to one combined area.

In chart 7-3, five symbolic parameters are defined in the macro prototype statement. The symbol, PRINT, is defined outside the macro definition. Note that each of the symbolic parameters used in the model statements appears in the macro instruction prototype statement.

Chart 7-3. Model Statements within a Definition

◆

NAME	OPERATION	OPERAND	
	MACRO		HEADER
&NAME	MOVE	&FRA,&LNA,&FRB,&LNB	PROTOTYPE
&NAME	MVC	PRINT(&LNA),&FRA	MODEL
	MVC	PRINT+&LNA(&LNB),&FRB	MODEL
	MEND		TRAILER

The values of the call for the positional macro MOVE in chart 7-4 correspond to the symbolic parameters of the positional macro prototype statement in chart 7-3. Namely, FIRST, NAME, 20, ADDR, and 15 (chart 7-4) correspond to &TAG, &FRA, &LNA, &FRB, and &LNB in chart 7-3. Any occurrance of the symbolic parameter in the Name, Operation, or Operand field of a model statement will be replaced by the corresponding characters; that is, &TAG is replaced with FIRST: &FRA with NAME, etc.

Chart 7-4. A Macro Call Statement for MOVE

◆

NAME	OPERATION	OPERAND	
FIRST	MOVE	NAME,20,ADDR,15	MACRO CALL

If the macro call statement in chart 7-4 were used in the source program, the Assembly Language statements shown in chart 7-5 would be generated.

Chart 7-5. Generated Assembly Statements

◆ STMNT	M	SOURCE	STATEMENT	
00010	M1	FIRST	MVC	PRINT(20),NAME
00011	M1		MVC	PRINT+20(15),ADDR

Note

◆ The MOVE macro removed from the programmer the clerical burden of putting the left-hand-end of ADDR in the 21st position (+20) of the PRINT area. Clerical errors such as transposition are also minimized. Given chart 7-3 notice the generation when the macro call values change (chart 7-6).

Each time the programmer uses the macro call statement MOVE in the same program, the Assembler uses the same macro definition for interpretation and generation unless superseded by a subsequent definition. (See chart 7-6.)

Chart 7-6. Macro Calls Followed by Their Generations

◆ STMNT	M	SOURCE	STATEMENT		
00050		SECOND	MOVE	A,50,B,30	MACRO CALL
00051	M1	SECOND	MVC	PRINT(50),A	GENERATED
00052	M1		MVC	PRINT+50(30),B	STATEMENTS
			. . .		
00100		THIRD	MOVE	C,5,D,2	MACRO CALL
00101	M1	THIRD	MVC	PRINT(5),C	GENERATED
00102	M1		MVC	PRINT+5(2),D	STATEMENTS
			. . .		
00125		FOURTH	MOVE	E,40,F,80	MACRO CALL
00126	M1	FOURTH	MVC	PRINT(40),E	GENERATED
00127	M1		MVC	PRINT+40(80),F	STATEMENTS

Chart 7-7. Use of Ampersands in Character and Self-Defining Values

◆ STMNT	M	SOURCE	STATEMENT	
01000	&TAG	DC C'&&TAGΔISΔNAME'		MODEL STATEMENT
01025	NAME	DC C'&&TAGΔISΔNAME'		GENERATED

The constant can be seen graphically as follows in the object code:

&TAGΔISΔNAME

Combining Symbolic Parameters (Concatenation)

♦ The characters represented by symbolic parameters in model statements can be combined with symbols, self-defining values, character values, and other symbolic parameters to produce symbols, self-defining values, and character values.

In combining symbolic parameters the following points must be considered:

1. When a symbolic parameter is followed by a left parenthesis, a period, an alphabetic character, or a numeric character, a period must separate the symbolic parameter from the character that follows:
2. When a symbolic parameter is followed by a single period, the period does not appear in the generated output.

Examples

♦ In the following examples, assume that &PARAM = A.

EXPRESSION	GENERATION
&PARAM.(BC)	A(BC)
&PARAM..BC	A.BC
&PARAM.BC	ABC
&PARAM.2BC	A2BC
BC,&PARAM	BC,A
B2&PARAM	B2A
&PARAM&PARAM	AA
&PARAM..&PARAM	A.A

Note

♦ The generated value of any expression cannot begin with a single & (symbolic parameter or other variable symbol).

For example: Assume &TO = &AR and &FROM = EA. Then &TO&FROM would produce &AREA, which would be flagged. However if &TO = AR and &FROM = EA, then &TO&FROM would generate AREA properly.

Chart 7-8. Combining Symbolic Parameters

STMT	M	SOURCE	STATEMENT		
00101			MACRO		DEFINITION
00102		&NM	ARITH	&OP,&TOT,&TAG	
00103		&NM&OP	&OP.P	&TOT.A,&TAG.A	
00104			&OP.P	&TOT.B,&TAG.B	
00105			&OP.P	&TOT.C,&TAG.C	
00106			MEND		
00200		TEST	ARITH	S,TOTAL,FIELD	CALL
00201	M1	TESTS	SP	TOTALA,FIELDA	GENERATION
00202	M1		SP	TOTALB,FIELDB	
00203	M1		SP	TOTALC,FIELDC	

Comments Statements

◆ Comments statements can be interspersed in the model statements of a macro definition. Two types of comments statements are permitted. The first type has an asterisk (*) in column 1, followed by the comment. This type is generated when the macro definition is assembled. The generated statement is identical to the statement coded by the programmer. The second type of comments statement has a period-asterisk (.*) in columns 1 and 2, followed by the comment. This type documents the macro definition and is not generated when the macro is assembled. See chart 7-9.

Chart 7-9. Comments

STMT	M	SOURCE STATEMENT	
00301		MACRO	(Definition)
00302		COMNT	
00303		*THIS COMMENT WILL NOT GENERATE	
00304		*THIS COMMENT WILL GENERATE	
00305		MEND	
00401		COMNT	(Call)
00402	M1	*THIS COMMENT WILL GENERATE	(Generated)

8. MACRO CALL STATEMENTS

GENERAL DESCRIPTION

◆ The macro call is a statement written in an Assembly language source program that calls the series of statements that make up the macro definition. This single statement is, in turn, replaced in the program by the variable number of generated statements from the macro definition. The statements that replace the macro call are called generated statements. A different call is required for each generation of a macro. This section discusses the positional macro call statement. The keyword macro call is explained in Section 11.

Example

◆ Chart 8-1 contains a part of a sample program utilizing macro calls. This example shows the macro definitions, macro call statements, and the generated statements.

The following reference table for chart 8-1 gives the statement numbers for each macro:

Statement Numbers (STMNT)

Macro Name	Macro Definition	Macro Call	Generated Statements
GETOD	00002 to 00006	00426	00427 and 00428
TERMS	00007 to 00012	00429	00430 to 00432

Chart 8-1. Macro Definitions, Calls, and Generation

LOCTN	OBJ. CODE...	STMNT	M	SOURCE	STATEMENT	
00000		00001		PROG	START	
		00002			MACRO	
		00003		&TAGA	GETOD	&TIME
		00004		&TAGA	SVC	23
		00005			DC	AL4(&TIME)
		00006			MEND	
		00007			MACRO	
		00008		&TAGB	TERMS	&NAME
		00009		&TAGB	SVC	28
		00010			SVC	10
		00011			DC	CL6'&NAME'
		00012			MEND	
00000		00013		BEGIN	BALR	3,0
00002		00014			USING	*,3
.		.			.	
.		.			.	
.		.			.	
.		.			.	
		00426		CALL01	GETOD	TIME
01C8A	0A 17	00427	M1	CALL01	SVC	23
01C8C	00001AB4	00428	M1		DC	AL4(TIME)
		00429		CALL02	TERMS	PROGB
01C90	0A 1C	00430	M1	CALL02	SVC	28
01C92	0A 0A	00431	M1		SVC	10
01C94	D7D9D6C7C240	00432	M1		DC	CL6'PROGB'
.		.			.	
.		.			.	
.		.			.	
.		.			.	
00000		00810			END	BEGIN

Positional Macro Call Statement

General Description

♦ The placement and order of the operand values in a positional macro call statement are determined by the placement and order of the symbolic parameters defined in the operand field of the macro prototype statement. (See Writing the Macro Definition, Section 7.) During generation, each symbolic parameter in the Name field, Operation field, or Operand field of a model statement is replaced by the operand values of the macro calls that positionally correspond to the symbolic parameters in the macro prototype statement.

Format

♦ The format for the positional macro call is as follows:

NAME	OPERATION	OPERAND
A symbol or blank.	Mnemonic operation code.	Comma (,) or a maximum of 49 operand values, separated by commas, in the form described below.

Specification Rules

Name Field

♦ The Name field of a macro call statement may contain a symbol. This symbol will only be defined if 1) a symbolic parameter appears in the Name field of the macro prototype statement and; 2) the same parameter appears in the Name field of a generated model statement.

If the Name field is blank, the symbolic parameter in the macro definition is considered to be a null parameter. (See NULL Parameter, page 8-6.) The value associated with the Name field is numbered zero (0).

Note:

In chart 8-1 the symbol CALL01 in the call statement will be defined because the symbolic parameter &TAGA appears in both the prototype statement and a model statement. CALL02 is similarly defined.

Operation Field

♦ The mnemonic operation code is the code assigned to the macro definition. This entry must contain the same operation code that appears in the Operation field of the prototype statement.

Note:

In chart 8-1 the operation code "GETOD" is used in STMNT 00003 (prototype) and STMNT 00426 (macro call).

Operand Field

◆ The Operand field may contain a maximum of 49 operand values, also called operand(s) or value(s), which must be separated by commas. The placement and order of the values in the macro call is determined by the placement and order of the symbolic parameters in the operand field of the macro prototype statement.

Note:

Operand values in the Operand field of the call statement are numbered 1-49. Value 0 (Name field) and 49 values in the Operand field give a maximum total of 50 operand values for any call statement.

Operand Rules

◆ The following are rules for the Operand field of the macro call:

1. The number of operand values must not exceed 49.

2. A comma must follow each value except the last.

   MOVE &FRA,&LNA,&FRB,&LNB PROTOTYPE

   MOVE A,5,B,10 CALL

3. A single comma (,) followed by at least one blank indicates that no operand exists.

   MOVE ,ΔΔ CALL WITH NO VALUES

4. The end of the Operand Field is indicated by at least one blank.

   MOVE C,2ΔΔCALL ENDS WITH 2 VALUES

5. Omitted operands must be indicated by an extra comma (,).

   MOVE D,,EΔΔCALL BOTH LENGTHS OMITTED

Note:

The operand field of any macro call statement is not scanned if the Operand field of the associated prototype statement contains no symbolic parameters.

Comments

◆ Comments may: 1) appear after the blank that indicates the end of all operands, 2) extend through the end column, and 3) be continued on one additional card.

Continuation Rules

◆ The following rules apply to continuing a positional call operand:

1. A line may be continued if the Operand Field to be continued extends through the end column.
2. To allow for a maximum of 49 values, as many continuation cards as required may be used.
3. Any operand value may be split between cards.

Chart 8-2. Example of Continuation for Positional Call Operands and Comments

NAME	OPERATION	OPERAND	
FIRST	MOVE	OPERA,20,OPERB,30,OPERC,5,OP	X
		ERD,15,OPERE,4,OPERF,2,OPERG	X
		,3,OPERH,1,OPERI,20,OPERK,10	X
		,OPERL,10ΔΔCONTINUEDΔOPERAND	X
		S,ΔCOMMENTS,Δ&ΔSPLITΔVALUESΔ	Δ

Quoted Strings

◆ A quoted string is any series of characters enclosed in quotation marks.

A quoted string starts with the first quotation mark in the operand value and ends with the first even numbered quotation mark that is not followed immediately by a quotation mark.

Subsequent quoted strings start with the first quotation mark after the quotation mark that ends the previous quoted string.

X'A'X'B'

Thus, 'A' and 'B' are quoted strings.

Call Values (Eight Characters)

◆ Any combination of up to eight characters can be used as an operand value of a macro call if the following rules are observed:

1. Quotation marks must always occur in pairs. (See Quoted Strings.)

   X'FF'

2. Two quotation marks must be used to represent a quotation mark enclosed in paired quotation marks.

   'CAN''T'

3. If a quotation mark is immediately preceded by the letter L and immediately followed by a letter, the quotation mark is not considered in determining paired quotation marks.

   L'MASTER

4. Parentheses must always occur in pairs, left parenthesis then right parenthesis.

   20(15,0)

5. Paired parentheses can be enclosed in paired parentheses.

   (A(2),B)

6. A parenthesis that occurs between paired quotation marks is not considered in determining paired parentheses.

   ')'

Call Values (Eight Characters) *(Cont'd)*

7. An equal sign can occur only as the first character in an operand or within paired quotation marks or paired parentheses.

   =X'FF' 'TO=A' E(F=G)

8. A comma indicates the end of an operand unless placed between paired parentheses or paired quotation marks.

   1,2,3 Three operand values

   (1,2),3 Two operand values

9. A blank indicates the end of the operand field unless placed between paired quotation marks.

   '3ΔORΔ4'

10. Ampersand signs must occur in pairs.

    '3Δ&&Δ4'

Note

◆ The total length of any call operand value must not exceed eight characters, including enclosed spaces.

Example

◆ Chart 8-3 shows a sample prototype statement and associated call statement. Each operand value contains the maximum of eight characters.

Chart 8-3. Maximum Length of Call Operands

NAME	OPERATION	OPERAND
&NAME	EXMPA	&A,&B,&C,&D (Prototype)
	EXMPA	PROGRAMA,'1Δ&&Δ2',(15,100),X
		L'MASTERΔCALLΔWITHΔ4ΔVALUESΔ

Null Parameters

◆ A null parameter is a parameter whose value is not included in the macro call, but is included in the prototype statement.

If an operand value is omitted from the Operand field of the macro call, then the comma that would have separated it from the next value must be present. If the last value(s), is omitted from a macro call, then the comma(s) separating the last value(s) from the previous value may be omitted.

Example

◆ The example in chart 8-4 shows a macro prototype followed by several macro calls with null parameters.

Chart 8-4. Examples of Null Parameters

◆

```
NAME      OPERATION                    OPERAND

&NAME     MOVE       &FRA,&LNA,&FRB,&LNB            (Prototype)

*PARAMETER OR OPERAND VALUE ARE AS FOLLOWS

*&NAME=PARAMETER 0, &FRA=1,&LNA=2,&FRB=3,&FRB=4

          MOVE       ,ΔALL PARAMS HAVE NULL VALUES

SECOND    MOVE       NAME,,ADDRΔΔPARAMS 2 & 4 ARE NULL

          MOVE       ,10,,5ΔΔPARAMS 0,1 & 3 ARE NULL

FOURTH    MOVE       ,,,20ΔΔPARAMS 1, 2, & 3 ARE NULL
```

Example

◆ If the symbolic parameter that corresponds to a null parameter is used in a model statement, a null character value replaces the symbolic parameter in the generated statement. The result will be the same as though the symbolic parameter did not appear in the statement.

For example, the first statement that follows is a model statement that contains the symbolic parameter &A. If the operand value that corresponds to &A were omitted from the macro, the second statement would be generated from the model statement. (See chart 8-5.)

Chart 8-5. A Null Parameter in a Model Statement

◆

```
NAME       OPERATION     OPERAND

NAME&A     MVC           WORK&A,FIELD&A          (Model)

NAME       MVC           WORK,FIELD              (Generated)
```

Inner Macro Calls

General Description

◆ A macro call may be used as a model statement in a macro definition. Macro calls used as model statements are called inner macro calls. (See chart 8-8 and chart 8-9.)

A macro call that is not used as a model statement is referred to as an outer macro call. (See chart 8-9.)

The rules for writing inner calls and outer calls are the same.

Any symbolic parameters used in an inner macro call are replaced by the corresponding values of the outer macro call before the inner call is scanned or generated.

Example

◆ In chart 8-9, the symbolic parameter &FILEA is replaced by READER in STMNTS 00426 and 00430. This value was given in the outer call for OPEN (STMNT 00425).

Nested Macros

◆ When a macro definition contains a macro call, the macros are said to be nested. The maximum depth of nesting is three. The following rules apply to nesting macros:

1. The outer macro is referred to as a first-level macro. Generation of the first-level macro is identified by M1 in the "M FIELD" on the assembled listing. (See Appendix A.)
2. The outer macro can generate as many second-level inner macro calls as are required. Generation of second-level macros is identified by M2.
3. Each second-level macro can generate as many third-level inner macro calls as are required. Third-level macro generation is identified by M3.
4. A third-level macro cannot generate a macro call.

Note

◆ The outer macro and the inner macros can be of the same or different types, either positional or keyword.

Chart 8-6. ASSGN Macro on Library

◆

NAME	OPERATION	OPERAND	
	MACRO		
	ASSGN	&CCB	
	CNOP	2,4	ASSGN GENERATION
	SVC	29	
	DC	A(&CCB)	
	MEND		

Chart 8-7. DTYPE Macro on Library

NAME	OPERATION	OPERAND	
	MACRO		
	DTYPE	&DEVICE, &R, &AREA	
	SVC	6	DTYPE GENERATION
	DC	CL6'&DEVICE'	
	DC	AL4(&AREA)	
	MEND		

Chart 8-8. OPEN Macro in Source Deck

STMNT	M		SOURCE	STATEMENT	
00004			MACRO		
00005		&NAME	OPEN	&FILEA, &FILEB, &FILEC	
00006			ASSGN	&FILEA	(Inner Call)
00007			DTYPE	&FILEA, , STORE	(Inner Call)
00008			B	&NAME	
00009		STORE	DS	CL1	
00010		&NAME	CLI	STORE, X'06'	IS CARD?
00011		*END OF PARTIAL GENERATION			
00070				MEND	

Chart 8-9. Macro Containing Two Inner Calls (Second-Level)

STMNT	M		SOURCE	STATEMENT	
00425		BEGIN	OPEN	READER, TAPEA	(Outer Call)
00426	M1		ASSGN	READER	(Inner Call)
00427	M2		CNOP	2,4	ASSGN GENERATION
00428	M2		SVC	29	
00429	M2		DC	A(READER)	
00430	M1		DTYPE	READER, , STORE	(Inner Call)
00431	M2		SVC	6	DTYPE GENERATION
00432	M2		DC	CL6'READER'	
00433	M2		DC	AL4(STORE)	
00434	M1		B	BEGIN	
00435	M1	STORE	DS	CL1	
00436	M1	BEGIN	CLI	STORE, X'06'	IS CARD?
00437	M1	*END OF PARTIAL GENERATION			

9. SET AND CONDITIONAL COMMANDS

INTRODUCTION

◆ The facilities described in Sections 6, 7, and 8 are sufficient to define and call a relatively simple macro.

For each of the macro definitions given in the preceding pages, a fixed series of statements are generated during assembly each time a macro call is encountered. The only difference in the generated statements of two or more macro calls for the same macro definition is the specific values and labels in each statement.

The Set and Conditional commands facilitate the writing of a more complex macro definition that will produce a tailored set of generated statements based on the values given in the macro call statement.

The sequence, number, and type of generated model statements can be based on the presence, absence, or values of: 1) operands in a particular macro call or, 2) set variable symbols (see below). Thus, the statements generated for two macro calls for the same macro definition might differ while the functions performed by the statements are basically the same.

SET VARIABLE SYMBOLS

♦ Set symbols and symbolic parameters are two types of variable symbols discussed in Section 6. Set symbols differ from symbolic parameters in two ways:

1. How they are assigned values;

2. Whether or not values assigned to them can be changed.

Symbolic parameters are assigned values when the programmer writes a macro call statement, whereas Set symbols are assigned values when the programmer uses the SETA, SETB, and SETC macro generator commands (see Defining Values). 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 can change during the use of a macro definition.

Defining Values

♦ The Set Commands (SETA, SETC, and SETB) assign arithmetic, character, and logical values, respectively, to Set variable symbols. If a value is not assigned by the programmer, values are assumed to be as follows:

1. SETA variable symbols (arithmetic values) have an assumed value of zero;

2. SETC variable symbols (character values) have a null character value, zero bytes in length;

3. SETB variable symbols (logical values) have an assumed value of false (0).

During the generation of a macro, the results of the operations performed by the Set Commands are contained in a series of specially provided areas in core storage referred to by Set variable symbols.

Global Values

♦ All Set variable symbols can be defined to be global in nature. This means that after a value has been defined for a particular Set variable symbol, the value remains in effect for all references to the variable symbol within the assembly until changed by another Set command.

For example, if a source program contains three macro calls and a SETA variable symbol is defined to have the value 6 in the macro definition called by the first macro call, the value 6 is used for the occurrence of the same SETA variable symbol within the macro definitions called by the other two macro calls unless changed. The programmer can, however, redefine the SETA variable symbol to have a value that differs from 6.

Local Values

◆ Two groups of Set variable symbols, SETA and SETB, can be defined to be *local* in nature. This means that after a value has been defined for a particular SETA or SETB variable symbol, the value remains in effect for all references to the variable symbol within the macro in which it was defined. After the macro is generated, the value of the SETA or SETB variable symbol is reset to zero or false.

For example, if a source program consists of two macro calls, and a SETB variable symbol is assigned a value of true in the macro definition called by the first macro call, the SETB variable symbol is reset to a value of false after the called macro is generated.

Notes

◆ 1. SETC variable symbols (character) must be defined as global.

2. When many calls are made for the same macro definition, it is sometimes helpful to use a binary global switch (see SETB) to generate a subroutine only once. The binary global is false initially. The macro definition sets the global switch to true after generation. Since a test of the switch will signal a true condition, the next call will generate only linkage to the already generated subroutine.

3. When macros are nested (see page 8-8), local SETA and SETB variable symbols defined in the outer (containing) macro are reset to zero and false, respectively, immediately before the inner (contained) macro is generated. After the inner macro has been generated the local variable symbols are reset to their previous values.

Uses for Set Symbols

◆ The Set commands allow arithmetic calculation, character manipulation, and the setting and testing of binary switches on the basis of logical and relational expressions.

The Conditional commands enable the programmer to tailor the statements generated by defining, conditionally or unconditionally, the next statement in the macro definition to be executed or generated. They also provide the means to generate error messages if a required condition is not met.

Where Set Symbols are Used

◆ Set variable symbols can be used in model statements, Set commands and Conditional commands.

Set variable symbols can be used in the Name, Operation, and Operand fields of macro definition statements with the following restrictions:

1. They cannot be used to generate a sequence symbol, (see page 9-22) a Set variable symbol, or a symbolic parameter;
2. They cannot appear in a macro prototype statement;
3. The SETC variable symbol can be used in the Operand field of a SETA statement only if the character string is composed of positive decimal digits. (See page 9-4.)

Note

◆ The functions of the Set and Conditional commands are interrelated, because the generated output is usually tailored by the use of Conditional commands based on the results obtained from the values generated by the Set commands. Their practical use is more clearly shown in the examples in the Conditional commands section.

SET COMMANDS

SETA
Set Arithmetic

General Description

◆ The SETA command assigns an arithmetic value to a SETA variable symbol. The programmer can change the value assigned to a SETA variable symbol by using another SETA command with the same variable symbol in the Name field.

The arithmetic value defined by a SETA instruction is represented in a model statement by the SETA variable symbol assigned. When a SETA variable symbol is detected during macro generation, it is replaced by the value of the symbol converted to a positive decimal, self-defining value with leading zeros dropped. (If the arithmetic value is zero, it will be converted to a single zero.)

Format

◆ The format of the SETA instruction is as follows:

NAME	OPERATION	OPERAND
A SETA variable symbol.	SETA	An arithmetic expression.

Specification Rules

Name Field

◆ 1. The SETA variable symbol defined in this field can be either local or global.

2. A global SETA variable symbol has the format &AGn, where n = 0 to 15.

3. A local SETA variable symbol has the format &ALn, where n = 0 to 15.

4. Therefore, up to 16 global and 16 local variable symbols can be defined. Each arithmetic value is 24 bits in length and is initially zero.

Operation Field

◆ SETA.

Operand Field

◆ 1. The expression in the Operand field can consist of a combination of terms in accordance with the rules given for expressions in Section 2.

2. The terms can be positive decimal self-defining values, symbolic parameters, or Set variable symbols that represent <u>positive</u> decimal self-defining values.

3. The arithmetic operators that can be used to combine terms are + (addition), - (subtraction), * (multiplication), and / (division).

4. An expression cannot contain two terms in succession or two operators in succession. An expression cannot begin with an operator.

5. Substrings are permitted. (See Substring Notation, page 9-10.)

Invalid Value

◆ If the operand of a SETA command is invalid or the result is invalid, a value of zero is assigned to the SETA variable symbol in the Name field.

Range of Values

◆ 1. The final value that can be assigned to a SETA Variable symbol must be positive. It can range from 0 to 16,777,215 ($2^{24}-1$).

2. Intermediate calculation values ran range from -2,147,483,648 (-2^{31}) to 2,147,483,647 ($2^{31}-1$).

Changing Values

◆ 1. If the programmer has assigned an arithmetic value to a SETA variable symbol, he can change the value assigned by using the SETA variable symbol in the Name field of another SETA statement.

2. If a SETA variable symbol has been used in the Name field of more than one SETA statement, the value substituted for the SETA variable symbol is the last value assigned to it. (See chart 9-1.)

Division

◆ 1. Division by zero results in a value of zero.

2. In division, only the integer portion of the quotient is retained. For example, 9 divided by 2 gives the result of 4. The fractional portion of 1/2 is dropped.

Examples

◆ The following are examples of expressions that can be used in the Operand field of a SETA command.

```
150          &AL3*2
&AL1+5       &AG4/4
&AG2-10      &LENGTH
```

In chart 9-1, the MOVE macro has been enlarged to illustrate SETA commands, changing the same Set variable symbol and ability to Move three fields.

It is assumed that there will be 10 spaces preceding the first field and 5 spaces after each field. Therefore, &AL1 will contain the number of spaces; &AL2 will contain the length of the last field moved; and AL3 will contain the position of the next field to be moved.

In chart 9-2 the call statement and generated statements are given for the MOVE macro. Prior to each generated statement, the value of each arithmetic local is shown.

Chart 9-1. SETA Commands with Changing Values

```
◆ NAME    OPERATION          OPERAND

          MACRO
          MOVE      &FRA,&LNA,&FRB,&LNB,&FRC,&LNC
  &AL1    SETA      10                        INITIAL SPACING
  &AL2    SETA      0                         LAST  LENGTH
  &AL3    SETA      &AL3+&AL1+&AL2            NEXT POSITION
          MVC       PRINT+&AL3(&LNA),&FRA
  &AL1    SETA      5
  &AL2    SETA      &LNA
  &AL3    SETA      &AL3+&AL2+&AL1
          MVC       PRINT+&AL3(&LNB),&FRB
  &AL2    SETA      &LNB
  &AL3    SETA      &AL3+&AL2+&AL1
          MVC       PRINT+&AL3(&LNC),&FRC
            .
            .
            .
          MEND
```

Chart 9-2. SETA Generation with Present Values

```
◆ STMNT      M      SOURCE  STATEMENT
  00100             MOVE      NAME,20,ADDR,15,CITY,25

  &AL1=10; &AL2=0; &AL3=0+10+0 or 10.

  00101      M1     MVC       PRINT+10(20),NAME

  &AL1=5; &AL2=20; &AL3=10+20+5 or 35.

  00102      M1     MVC       PRINT+35(15),ADDR

  &AL1=5; &AL2=15; &AL3=35+15+5 or 55.

  00103      M1     MVC       PRINT+55(25),CITY
```

SETC Set Character

General Description

◆ The SETC command assigns a character value to a SETC variable symbol. The programmer can change the character value assigned to a SETC variable by using another SETC command with the same variable symbol in the Name field. The characters specified in the Operand field are assigned to the SETC variable symbol designated in the Name field.

Format

◆ The format for the SETC instruction is as follows:

NAME	OPERATION	OPERAND
A SETC variable symbol.	SETC	Up to eight characters enclosed by a pair of single quote marks.

Specification Rules

Name Field

◆ 1. The SETC variable symbol in the Name field is global in nature. It has the form &CGn, where n = 0-15.

2. The SETC command can define up to 16 different global character values. Each global character value can vary from zero-to-eight bytes in size. Each character value is initially a null character value of zero bytes.

3. If a SETC variable symbol has been used in the Name field of more than one SETC statement, the value substituted for the SETC variable symbol is the last value assigned to it. (See chart 9-4.)

4. A SETC variable symbol used in the name field of a SETC statement can be used in the operand field of SETA, SETB, SETC, AIF, and AIFB statements.

Operation Field

◆ SETC.

Operand Field

◆ 1. The characters in the Operand field are assigned to the SETC variable symbol in the Name field and are substituted for the SETC variable symbol when it is used. (See chart 9-3.) The operand can consist of a string of characters, a previously defined Set variable symbol, symbolic parameters or any combination thereof and must be enclosed in a pair of single quotation marks.

2. Set variable symbols can be combined with other characters in the Operand field of a SETC instruction according to the general rules for combining symbolic parameters with other characters.

3. More than one character value can be combined into a single character value by placing a period between the termination quotation mark of one character value and the opening quotation mark of the next character value. (See chart 9-4.)

Operand Field (Cont'd)

4. Two single quotation marks must be used to represent a quotation mark that is part of a character expression enclosed in quotation marks. (See chart 9-5.)

5. 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 by the SETC symbol. (See chart 9-6.)

6. A SETA variable symbol that has been assigned an arithmetic value by a SETA statement can be used in the Operand field of a SETC statement. It will be replaced by the value of the SETA variable symbol converted to a decimal self-defining value with any leading zeros dropped. (See chart 9-7.)

Examples

◆ Charts 9-3 through 9-7 illustrate the preceding Operand field rules.

Chart 9-3. SETC Command, Last Value Substituted

NAME	OPERATION	OPERAND	
&CG1	SETC	'NAME'	GENERATES NAME
&CG1	SETC	'ADDR'	GENERATES ADDR

Chart 9-4. SETC Command, Combination

NAME	OPERATION	OPERAND	
&CG2	SETC	'&CG1'.'ONE'	GEN. = ADDRONE

Chart 9-5. SETC Command, Two Quotes

NAME	OPERATION	OPERAND	
&CG3	SETC	'L''NAME'	GEN. = L'NAME

Chart 9-6. SETC Command, Two Ampersands

NAME	OPERATION	OPERAND	
&CG4	SETC	'TWO&&'	GEN. = TWO&&

Chart 9-7. SETC Command, Using SETA Symbol

NAME	OPERATION	OPERAND	
&AL4	SETA	12	
&CG5	SETC	'AREA'.'&AL4'	GEN. = AREA12

Chart 9-8. MOVE Macro Using SETC

NAME	OPERATION	OPERAND
	MACRO	
	MOVE	&FRA,&LNA &FRB,&LNB
&CG1	SETC	'PRINT'
&CG2	SETC	'NAME'
	MVC	&CG1(&LNA),&CG2
&CG2	SETC	'ADDR'
	MVC	&CG1+&LNA(&LNB),&CG2
	MEND	

Chart 9-9. MOVE Macro With SETC Generation

STMNT	M	SOURCE	STATEMENT
00200		MOVE	,20,,15
00201	M1	MVC	PRINT(20),NAME
00202	M1	MVC	PRINT+20(15),ADDR

Substring Notation

♦ The Operand field of a SETC or SETA variable symbol command can be composed of a substring. Substrings permit the programmer to assign to a SETC or SETA variable symbol a portion of the value assigned to another character string.

Format

♦ The format for the SETC and SETA substring is as follows:

NAME	OPERATION	OPERAND
A set variable symbol.	SETC or SETA	'CC...C'(X, Y)

Specification Rules

Name Field

♦ See SETC (page 9-7) or SETA (page 9-4).

Operation Field

♦ SETC or SETA.

Operand Field

♦ 1. The Operand field consists of a character string 'CC...C' followed by two arithmetic expressions (X,Y) enclosed by parentheses and must be separated by a comma (see 4 below).

2. 'CC...C' may be: (a) other Set variable symbols; (b) symbolic parameters; (c) self-defining values; or (d) any valid combination thereof.

3. The calculated character string 'CC...C' to be extracted from an intermediate string must not exceed eight characters. An intermediate string must not exceed sixteen characters in length at any one time.

4. X and Y may be any valid arithmetic expressions which are allowed in the Operand field of a SETA command, where:

   X = the position of the first character (LHE) in the character string to be assigned to the SETC or SETA symbol in the Name field.

   Y = the number of consecutive characters to be assigned to the SETC or SETA symbol in the Name field. The characters must be numeric if SETA.

Notes

♦ 1. If 'CC...C' is a SETA variable symbol the leading zeros are ignored in determining X.

2. The maximum value for X is 16.

3. The maximum value for Y is 8.

Chart 9-10. SETC and SETA Substrings

NAME	OPERATION	OPERAND	GENERATES
&CG1	SETC	'ABCDEFGH'	ABCDEFGH
&AL1	SETA	4	4
&AL2	SETA	34567	34567
&CG2	SETC	'&CG1'(1,3)	ABC
&CG3	SETC	'&CG1'(2,&AL1)	BCDE
&AL3	SETA	'&AL2'(2,4)	4567

Note

◆ The values of &CG2,&CG3, and &AL3 are generated by valid substring notations.

Combining Substrings SETC

◆ Substrings can be obtained in the Operand field with other character values in a SETC command. (Also see Combining Substrings - SETA, page 9-00.)

1. If a substring follows a character value that is not a substring, the two can be combined by placing a period between the first character value and the substring.

Chart 9-11. SETC Substring Follows Value

NAME	OPERATION	OPERAND	GENERATES
&CG1	SETC	'ABCDEFGH'	ABCDEFGH
&CG2	SETC	'XYZ'.'&CG1'(2,4)	XYZBCDE
&CG3	SETC	'XYZ&CG1'(2,4)	YZAB

Notes

◆ 1. The value of &CG2 illustrates that only &CG1 is substringed, whereas the value of &CG3 includes the constant XYZ before substringing.

2. If the substring precedes another character value, the two can be combined by placing the terminating parenthesis of the substring and the opening quote of the next character value adjacent to one another.

Chart 9-12. SETC Substring Precedes Value

NAME	OPERATION	OPERAND	GENERATES
&CG1	SETC	'ABCDEFGH'	ABCDEFGH
&CG2	SETC	'&CG1'(2,4)'XYZ'	BCDEXYZ
&CG3	SETC	'&CG1'(2,4)'&CG1'(3,4)	BCDECDEF

Combining Substrings SETA

◆ Combining substrings in the Operand field of a SETA command requires that there cannot be two terms in succession. Thus, only one term may be present, or each term present must be separated by an operator (+, -, *, or /).

Chart 9-13. SETA Substrings

NAME	OPERATION	OPERAND	GENERATES
&AL1	SETA	2345678	2345678
&AL2	SETA	4	4
&AL3	SETA	2	2
&AL4	SETA	'&AL1'(2,4)	3456
&AL5	SETA	'&AL1'(&AL2,&AL3)	56
&AL6	SETA	'&AL2'+'&AL1'(4,2)+5	65
&AL7	SETA	'2345678'(&AL2,&AL2)	5678
&AL8	SETA	'2345678'(2,5)	34567

Note

◆ The values of &AL4 through &AL8 are generated by valid substrings for SETA commands.

Use of Substrings

◆ Substrings are useful in assigning a portion of an existing variable symbol, symbolic parameter, or value to another variable symbol.

Example

◆ In chart 9-14 the MOVE macro contains two operand symbolic parameters representing four values. The substring technique is used to separate the values. &FROM represents two 4-character NAMES; &LENGTH represents two 3-character length values.

Chart 9-14. MOVE Macro Utilizing Substrings

NAME	OPERATION	OPERAND	
	MACRO		
&NAME	MOVE	&FROM,&LENGTH	
&CG1	SETC	'&FROM'(1,4)	FIRST NAME
&AL1	SETA	'&LENGTH'(1,3)	FIRST LENGTH
&CG2	SETC	'&FROM'(5,4)	SECOND NAME
&AL2	SETA	'&LENGTH'(4,3)	SECOND LENGTH
&CG3	SETC	'PRINT'	CONSTANT PRINT
&NAME	MVC	&CG3(&AL1),&CG1	
	MVC	&CG3+&AL1(&AL2),&CG2	
	MEND		

In chart 9-15 the call and generation for the macro definition in chart 9-14 is shown.

Chart 9-15. Substring Macro Generation

STMNT	M	SOURCE STATEMENT		
00009		FIRST	MOVE	NAMEADDR,020015
00010	M1	FIRST	MVC	PRINT(20),NAME
00011	M1		MVC	PRINT+20(15),ADDR

Note

◆ Chart 9-15 gives the same generation as did chart 7-5 (page 7-8).

SETB Set Binary

General Description

◆ The SETB command assigns the value true or false to a SETB variable symbol. The programmer can change the value assigned to a SETB variable symbol by using another SETB command.

The logical expression or relational expression in the Operand field is evaluated to determine if it is true or false, and the value 1 or 0, respectively, is assigned to the SETB variable symbol appearing in the name field.

Format

◆ The format for the SETB instruction is as follows:

NAME	OPERATION	OPERAND
A SETB variable symbol.	SETB	A 0 or a 1, or a logical or relational expression enclosed within parentheses.

Specification Rules

Name Field

◆ 1. The SETB variable symbol in this field can be either local or global.

2. A global SETB variable symbol has the format &BGn, where n = 0-127.

3. A local SETB variable symbol has the format &BLn, where n = 0-127.

4. There are a maximum of 128 global and 128 local variable symbols which are initially set to zero (false).

Operation Field

◆ SETB.

Operand Field

◆ The Operand field may consist of either a logical expression or a relational expression enclosed by parentheses. Single-termed logical expressions 0 or 1 may have the parentheses omitted.

Invalid Value

◆ If the operand of a SETB is invalid or the result is invalid, a value of zero (false) is assigned to the SETB variable symbol in the Name field.

Note

◆ The logical value that has been assigned to a SETB variable symbol is substituted for the SETB variable symbol when it is used in the Operand field of a SETB, AIF, or AIFB (see pages 9-23 and 9-25) conditional assembly instruction. If the variable symbol is used in any other assembly language statement, the logical value is converted to an integer. The logical value True is converted to the integer 1, and the logical value False is converted to the integer 0.

Logical Expressions

◆ A logical expression can consist of one of the following:

1. Single term.
2. Two or more terms separated by one of the logical operators NOT, AND, or OR.
3. One or more sequences of logical expressions enclosed in parentheses.

The following procedure is used to evaluate a logical expression in the operand field of a SETB instruction:

1. Each term (that is, 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. The priority of performing operators is: NOT, AND, and then OR.
3. The computed result is the value assigned to the SETB symbol in the Name field (see Logical Operator Evaluation, page 9-21).
4. 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.

Single Term Logical Expressions

◆ If a logical expression consists of a single term, the term must be one of the following:

1. The value of 0 (false);
2. The value 1 (true);
3. SETB variable symbol;
4. The operator NOT followed by one SETB symbol;

Chart 9-16. Examples of Single Term Logical SETB

◆

NAME	OPERATION	OPERAND	GENERATES
&BG1	SETB	1	1 = True
&BL2	SETB	0	0 = False
&BG3	SETB	(&BG1)	1 = True
&BL4	SETN	(BL2&)	0 = False
&BG5	SETB	(NOT &BL2)	1 = True
&BL6	SETB	(NOT &BG1)	0 = False

Notes

◆ &BG3 and &BL4 take on the same value as the SETB symbol in the Operand field. &BG5 and &BL6 take on the opposite value because of the NOT. A symbolic parameter may not be used.

Two-Term Logical Expression

◆ In a logical expression consisting of two terms, the terms must be SETB variable symbols, separated by at least one operator and enclosed in parentheses.

Note

◆ The following are rules for two-term expressions:

Assume &BL1 = 1 (True), and &BG2 = 0 (False).

1. The two terms must be separated by an operator.

   &BL3 SETB (&BL1ΔORΔ&BG2)

   Generates 1 or True

2. Two operators may appear in succession only if the pair of operators are ANDΔNOT or ORΔNOT.

   &BG4 SETB (&BG1ΔANDΔNOTΔ&BL2)

   Generates 1 or true.

3. NOT may begin an expression, whereas AND and OR cannot.

   &BL5 SETB (NOTΔ&BL1ΔORΔ&BG2)

   Generates 0 or False.

4. The logical operators must be separated from the terms they relate by at least one blank.

   &BG6 SETB (&BL1ΔANDΔ&BG2)

   Generates 0 or False.

5. The entire logical expression must be enclosed with parentheses.

   &BL7 SETB (&BL1ΔORΔNOTΔ&BG2)

   Generates 1 or True.

Multiterm Logical Expressions

◆ When a logical expression consists of more than two terms, three levels of parentheses and only one continuation card are permitted. The expression is examined from the innermost parentheses outward.

Within each pair of parentheses the logical operators are performed in the following order: NOT, AND, OR. Each set of operators are performed from left to right. (See chart 9-17.)

Note

◆ The rules for two-term expression apply to multiterm expressions.

Example

◆ In chart 9-17, two logical expressions, &BG5 and &BG6 have two different values by adding an inner set of parentheses (nested).

Chart 9-17. Nested Multiterm Logical Expressions

◆ NAME	OPERATION	OPERAND	GENERATES
&BG1	SETB	1	1 = True
&BG2	SETB	0	0 = False
&BG3	SETB	1	1 = True
&BG4	SETB	0	0 = False
&BG5	SETB	(&BG1ΔORΔ&BG2ΔANDΔ&BG4)	1 = True
&BG6	SETB	((&BG1ΔORΔ&BG2)ΔANDΔ&BG4)	0 = False

Relational Expressions

◆ A relational expression can be an arithmetic relation or a character relation.

A relational expression cannot contain two values in succession. A relational expression cannot contain two operators in succession. The relational operators must be separated from the values they relate by at least one blank.

The relational operators are EQ (equal), NE (not equal), LT (less than), GT (greater than), LE (less than or equal to), and GE (greater than or equal to).

Example

◆ Chart 9-18 illustrates several examples of valid arithmetic and character relations.

Chart 9-18. Relational SETB Expressions

◆ NAME	OPERATION	OPERAND	TYPE OF COMPARISON
&BL1	SETB	(&AL4ΔEQΔ12)	Arithmetic
&BL2	SETB	(&LNAΔLTΔ256)	Arithmetic
&BL3	SETB	('&CG1'ΔNEΔ'PRINT')	Character
&BL4	SETB	('&FRA'ΔEQΔ'NAME')	Character

Note

◆ The type of expressions in the relation determines the nature of the comparison that is involved. A logical compare results when all the relational expressions are considered as character; that is, <u>all the expressions are enclosed in single quotes.</u> All other cases result in an arithmetic (algebraic) comparison.

Arithmetic Relational Expressions

◆ An arithmetic relation consists of two arithmetic expressions connected by a relational operator and must be enclosed within parentheses. The terms are not enclosed by single quotes.

An arithmetic expression can be a SETA variable symbol, a SETC variable symbol, or any valid operand of a SETA statement. If a SETC variable symbol is used in an arithmetic relation, the SETC variable symbol must represent a positive decimal arithmetic value.

An arithmetic or algebraic comparison is made between two arithmetic expressions by performing a Compare Word (RR format) instruction on the values involved.

Note

◆ If any of the terms of a relational expression are not enclosed by single quotes, the entire expression is considered to be arithmetic.

Examples

◆ Chart 9-19 illustrates valid arithmetic relational expressions. Assume the following values: &AL1 = 23; &CG1 = 123; &LNA = 10.

Chart 9-19. SETB Arithmetic Relational Expressions

NAME	OPERATION	OPERAND	GENERATES
&BL5	SETB	(&AL1ΔGTΔ5)	True
&BL6	SETB	(&AL1ΔEQΔ'&CG1'(2,2))	True
&BL7	SETB	(&LNA+5ΔGEΔ20)	False
&BL8	SETB	AL1+&LNA*2ΔGTΔ3*&CG1+4)	False

Character Relational Expressions

◆ A character relation consists of two character values connected by a relational operator. Each character value must be enclosed by single quotation marks. A character value can be a SETA variable symbol, a SETC variable symbol, or any valid operand of a SETC statement, including substrings. If a SETA variable symbol is used in a character relation, the SETA variable symbol is treated as a character value. The maximum length of any character value used in a character relation is eight characters. If two character values in a character relation are of unequal length, the longer value is always considered greater, regardless of the content of the two values.

A logical compare is made by first determining if the expressions are of equal length; if not, the longer is considered greater and no further testing is performed. If the expressions are equal in length, the two character strings are compared and their relationship determined.

Note

◆ Unless all of the terms within an expression are enclosed by single quotes, an arithmetic relation is assumed.

Examples

◆ Chart 9-20 illustrates valid character relational expressions. Assume the following values: &CG1 = DOG, &CG2 = CAT, &CG3 = CAGE.

Chart 9-20. SETB Character Relational Expressions

NAME	OPERATION	OPERAND	GENERATES
&BL9	SETB	('&CG1'ΔGTΔ'&CG2')	True
&BL10	SETB	('&CG1'ΔGTΔ'&CG3')	False
&BL11	SETB	('&CG1'ΔGTΔ'&CG3'(1,3))	True

Complex Relational Expressions

◆ When the Operand field of a SETB command contains a combination of logical and relational expressions, the relational expressions are evaluated first according to the rules for relational expressions. Similarly, the logical expressions are then evaluated.

Examples

◆ Chart 9-21 shows two valid complex expressions. Assume the same values as chart 9-19 and chart 9-20.

Chart 9-21. Complex SETB Relational Expressions

NAME	OPERATION	OPERAND	GENERATES
&BL12	SETB	(NOTΔ(&BL8ΔANDΔ &LNA+5ΔGEΔ20))	True
&BL13	SETB	(&BL8ΔANDΔ('&CG2'Δ EQΔ'CAT'ΔORΔ&BL6))	False

Testing for Null Parameters

◆ The SETB, AIF, and AIFB commands can be used to test for the presence of a null parameter. (See pages 9-23 and 9-25.) This is done by placing the symbolic parameter to be tested in the Operand field of a AIF, AIFB, or SETB command and equating (EQ) it to a null character string. A null character string is represented by two single quote marks. If the parameter value is present in the macro call, the result is false or 0. If the parameter value is not present in the macro call, the result is true or 1. (If NE is used the results are reversed.)

Chart 9-22. Testing for Null Parameters

NAME	OPERATION	OPERAND	COMMENTS
	MACRO		
&NAME	ADD	&FROM1, &FROM2, &SUM	PROTOTYPE
&BG1	SETB	('&FROM1'ΔEQΔ'')	IS &FROM1 = NULL
&BG2	SETB	('&FROM2'ΔEQΔ'')	IS &FROM2 = NULL
&BG3	SETB	('&FROM1'ΔNEΔ'')	IS &FROM1 ≠ NULL
&BG4	SETB	('&FROM2'ΔNEΔ'')	IS &FROM2 ≠ NULL
	MEND		
FIRST	ADD	FIELD1, , FIELD3	

*&BG1 = ZERO(0) i.e. FALSE - FROM1 ISN'T NULL

*&BG2 = ONE(1) i.e. TRUE - FROM2 IS NULL

*&BG3 = ONE(1) i.e. TRUE - FROM1 IS NOT NULL

*&BG4 = ZERO(0) i.e. FALSE - FROM2 ISN'T NOT NULL

Logical Operator Evaluation

◆ A term(s) in conjunction with a logical operator is (are) evaluated according to chart 9-23 of Boolean logic.

Chart 9-23. Boolean Logic for Logical Operators

◆ Operator(s)	First Term	Second Term	Value of Expression
NOT	FALSE	———	TRUE
	TRUE	———	FALSE
AND	TRUE	TRUE	TRUE
	FALSE	FALSE	FALSE
	TRUE	FALSE	FALSE
	FALSE	TRUE	FALSE
OR	TRUE	TRUE	TRUE
	TRUE	FALSE	TRUE
	FALSE	TRUE	TRUE
	FALSE	FALSE	FALSE
AND NOT	TRUE	FALSE	TRUE
	TRUE	TRUE	FALSE
	FALSE	TRUE	FALSE
	FALSE	FALSE	FALSE
OR NOT	TRUE	TRUE	TRUE
	TRUE	FALSE	TRUE
	FALSE	FALSE	TRUE
	FALSE	TRUE	FALSE

CONDITIONAL COMMANDS

♦ The conditional commands enable the programmer to alter the sequence in which the statements of a macro definition will be generated and thus executed.

The AGO or AGOB command is similar to an unconditional branch instruction. It indicates, by means of a sequence symbol, the next statement to be processed by the macro generator.

The AIF or AIFB command is similar to a conditional branch instruction. It indicates, by means of the logical value obtained from the operand of a SETB statement and a sequence symbol, the next statement to be processed by the macro generator if the condition is true.

To assist the programmer in validating complex macro logic, a trace mode is available to indicate on the assembly listing nongenerative conditional transfers. (See MTRAC, Section 10.)

The ANOP command is essentially a no-op instruction that is used with the AGO, AGOB, AIF, and AIFB conditional commands.

Sequence Symbols

♦ The Name field of a statement may contain a sequence symbol. The sequence symbol can be used in the Operand field of an AIF, AIFB, AGO, or AGOB statement to refer to the statement named by the sequence symbol.

A sequence symbol consists of a period followed by a maximum of seven alphabetic and/or numeric characters, the first of which must be alphabetic. All sequence symbols used in a macro definition must be different. A sequence symbol that appears in the Name field can be referred to only by AIF, AIFB, AGO, and AGOB commands in the same macro definition.

The following are valid sequence symbols:

.READER	.A23456	.AG4
.LOOP2	.X4F2	.SYSTEM
.N	.S4	.BL16

A sequence symbol can be used in the Name field of any model statement within a macro definition that does not require a symbol or Set variable symbol, except a macro definition header statement (MACRO) or a macro prototype statement. Sequence symbols can then be used in the Operand field of an AIF, AIFB, AGO or AGOB command to refer to the statement named by the sequence symbol. A sequence symbol appearing in the Name field of a model statement does not appear in the generated statement.

If a sequence symbol appears in the Name field of an inner macro call in a macro definition, and the corresponding macro prototype contains a symbolic parameter in the Name field; the sequence symbol does not replace the symbolic parameter in the model statement.

Note

♦ A sequence symbol that is used in the Name field can be referred to only by AIF, AIFB, AGO, and AGOB in the same definition.

AIF Conditional Branch

General Description

◆ The AIF command alters conditionally the sequence in which macro definition statements are executed or generated in the object program. The sequence symbol in the Operand field must be in the Name field of any macro definition statement following the AIF command.

Format

◆ The format of the AIF command is as follows:

NAME	OPERATION	OPERAND
A sequence symbol or blank.	AIF	A logical or relational expression enclosed in parentheses followed by a sequence symbol.

Specification Rules

Name Field

◆ A sequence symbol or blank.

Operation Field

◆ AIF.

Operand Field

◆ Any logical or relational expression that can be used in the Operand field of a SETB command can be used as the expression in the Operand field of an AIF command including testing for null parameter values. The logical or relational expression must be enclosed in parentheses. The sequence symbol in the Operand field must immediately follow the closing parentheses of the logical or relational expression. The sequence symbol in the Operand field must be in the Name field of any macro definition statement following the AIF command.

The logical or relational expression in the Operand field is evaluated to determine if it is true (1) or false (0). If the expression is true, the macro definition statement named by the sequence symbol in the Operand field is the next statement processed by the macro generator. If the expression is false, the next sequential statement is processed by the macro generator.

The following are examples of valid Operand fields of the AIF command:

(&BG12ΔANDΔ&BL10).LOOP

(&AL10ΔEQΔ&AG6).LAST

Note

◆ The statement following the REPRO statement is not scanned during macro generation.

Example

◆ The example in chart 9-24 illustrates the use of the AIF Conditional command. It also illustrates the use of global Set variable symbols to carry values between macro calls in the same assembly. The first time the macro call appears in an assembly, record area is defined. The generated instructions of all additional calls of this macro definition in an assembly use the record area specified in the first appearance of the macro call.

Note that the B and DS statements are not generated for the second macro call, because when the first macro was generated, &BG100 was set to 1.

Note:

Although the prototype allows for two fields, &FRB is tested for null. Thus, the second macro call generates only one MVC statement.

Chart 9-24. Use of AIF Command

◆

NAME	OPERATION	OPERAND
	MACRO	
	MOVE	&FRA,&LNA,&FRB,&LNB
	AIF	(&BG100).GO
&BG100	SETB	1
&CG15	SETC	'RECORD'
	B	&CG15+150
&CG15	DS	CL150
.GO	MVC	&CG15(&LNA),&FRA
	AIF	('&FRB'ΔEQΔ'').END
	MVC	&CG15+&LNA(&LNB),&FRB
.END	MEND	

Chart 9-25 shows the macro calls and generation for the definition in chart 9-24.

Chart 9-25. AIF Changes Generations

◆

STMNT	M	SOURCE		STATEMENT	
00020			MOVE	NAME,20,ADDR,15	CALL
00021	M1		B	RECORD+150	
00022	M1	RECORD	DS	CL150	
00023	M1		MVC	RECORD(20),NAME	
00024	M1		MVC	RECORD+20(15),ADDR	
00025			MOVE	A,50	CALL
00026	M1		MVC	RECORD(50),A	

AIFB Conditional Branch Backward

General Description

◆ The AIFB command alters conditionally the sequence in which macro definition statements are executed or generated in the object program. The AIFB command is identical to the AIF command, except that the sequence symbol in the Operand field must be in the Name field of any macro definition statement preceding the AIFB command.

Format

◆ The format for the AIFB command is as follows:

NAME	OPERATION	OPERAND
A sequence symbol or blank.	AIFB	A logical or relational expression enclosed in parentheses followed by a sequence symbol.

Specification Rules

◆ The rules for the Name field and Operand field are identical to those given under AIF on page 9-23, except as noted in the Operand fields.

Name Field

◆ Sequence symbol or blank.

Operation Field

◆ AIFB.

Operand Field

◆ A logical or relational expression followed by a sequence symbol, which appears in the Name field of any macro definition statement preceding the AIFB command.

Example

◆ The example in chart 9-26 illustrates the use of the AIFB command. The function of the macro definition is to move a specified number of bytes of information from one location in core storage to another. The macro mnemonic is MOVER. The first parameter represents the number of bytes to be moved. The second parameter specifies the first position of the field to be filled. The third parameter specifies the location of the first byte to be moved. Note that the value of the local variable symbol &AL1 is initially zero.

Chart 9-26. AIFB Command

NAME	OPERATION	OPERAND
	MACRO	
	MOVER	&NOCHAR,&TO,&FROM
&AL2	SETA	&NOCHAR
	AIF	(&AL2 LE 256).LSTMOV
.LOOP	MVC	&TO+&AL1.(256),&FROM+&AL1
&AL1	SETA	&AL1+256
&AL2	SETA	&NOCHAR - &AL1
	AIFB	(&AL2 GT 256).LOOP
.LSTMOV	MVC	&TO+&AL1.(&AL2),&FROM+&AL1
	MEND	

In chart 9-27, the macro calls and generation using the macro definition in chart 9-26 are shown.

Chart 9-27. AIFB Generations

STMNT	M	SOURCE	STATEMENT	
00100		MOVER	540,OUT,INPUT	CALL 1
00101	M1	MVC	OUT+0(256),INPUT+0	
00102	M1	MVC	OUT+256(256),INPUT+256	
00103	M1	MVC	OUT+512(28),INPUT+512	
00104		MOVER	97,OUT+540,RESULT	CALL 2
00105	M1	MVC	OUT+540+0(97),RESULT+0	

AGO Unconditional Branch

General Description

◆ The AGO command alters the sequence in which macro definition statements are executed or generated in the object program. The sequence symbol in the Operand field must be in the Name field of any macro definition statement following the AGO command.

Format

◆ The format of the AGO instruction is as follows:

NAME	OPERATION	OPERAND
A sequence symbol or blank.	AGO	A sequence symbol.

Specification Rules

Name Field

◆ A sequence symbol or blank.

Operation Field

◆ AGO.

Operand Field

◆ The sequence symbol in the Operand field must be in the Name field of any macro definition statement following the AGO command. The statement named by the sequence symbol in the Operand field is the next statement processed by the macro generator.

Example

◆ The example in chart 9-28 illustrates the use of the AGO conditional command. The macro definition in this example is functionally the same as the macro definition in chart 9-26.

Chart 9-28. AGO Command

◆

NAME	OPERATION	OPERAND
	MACRO	
	MOVER	&NOCHAR,&TO,&FROM
&AL2	SETA	&NOCHAR
	AIF	(&AL2 GT 256).LOOP
	AGO	.LSTMOV
.LOOP	MVC	&TO+&AL1.(256),&FROM+&AL1
&AL1	SETA	&AL1+256
&AL2	SETA	&NOCHAR - &AL1
	AIFB	(&AL2 GT 256).LOOP
.LSTMOV	MVC	&TO+&AL1.(&AL2),&FROM+&AL1
	MEND	

Note

◆ In chart 9-27 the macro calls and generation using the macro definition in chart 9-28 are shown.

AGOB Unconditional Branch Backward

General Description

◆ The AGOB alters the sequence in which macro definition statements are executed or generated in the object program. The AGOB command is identical to the AGO command, except that the sequence symbol in the Operand field must be in the Name field of any macro definition statement preceding the AGOB command.

Format

◆ The format of the AGOB command is as follows:

NAME	OPERATION	OPERAND
A sequence symbol or blank.	AGOB	A sequence symbol.

Specification Rules

◆ The rules for the Name field and the Operand field are identical to those given under AGO except as noted in the Operand field.

Name Field

◆ A sequence symbol or blank.

Operation Field

◆ AGOB.

Operand Field

◆ Identical to AGO command except the sequence symbol must be in the Name field of any macro definition statement preceding the AGOB command.

Example

◆ The example in chart 9-29 illustrates the use of the AGOB conditional command. The macro definition in this example is functionally the same as the macro definition in chart 9-26.

Chart 9-29. AGOB Instruction

◆

NAME	OPERATION	OPERAND
	MACRO	
	MOVER	&NOCHAR,&TO,&FROM
&AL2	SETA	&NOCHAR
.LOOP	AIF	(&AL2 LE 256),LSTMOV
	MVC	&TO+&AL1.(256),&FROM+&AL1
&AL1	SETA	&AL1+256
&AL2	SETA	&NOCHAR - &AL1
	AGOB	.LOOP
.LSTMOV	MVC	&TO+&AL1.(&AL2),&FROM+&AL1
	MEND	

Note

◆ In chart 9-27 the macro calls and generation using the macro definition in chart 9-29 are shown.

ANOP No Operation

General Description

♦ The ANOP command facilitates conditional or unconditional branching to statements that are named by symbols or Set variable symbols in the Name field. The ANOP statement should be placed before the statement it is desired to branch to and a branch performed to the ANOP statement.

Format

♦ The format for the ANOP command is as follows:

NAME	OPERATION	OPERAND
A sequence symbol.	ANOP	Not used.

Specification Rules

Name Field

♦ A sequence symbol or blank.

Operation Field

♦ ANOP.

Operand Field

♦ Not used.

Example

♦ The example in chart 9-30 illustrates the use of the ANOP command. This example allows a field of any length to be moved. The source and destination fields need not be on a fullword boundary. The Name field contains the symbolic name of the first instruction of the generated macro. Note that the value of the local variable symbol &AL1 is initially zero.

Chart 9-30. ANOP Command

♦

NAME	OPERATION	OPERAND
	MACRO	
&NAME	MOVER	&NOCHAR,&TO,&FROM
&AL2	SETA	&NOCHAR
&CG1	SETC	'&NAME'
.LOOP	AIF	(&AL2 LE 256).LSTMOV
&CG1	MVC	&TO+&AL1.(256),&FROM+&AL1
&AL1	SETA	&AL1+256
&AL2	SETA	&NOCHAR - &AL1
&CG1	SETC	''
	AGOB	.LOOP
.LSTMOV	ANOP	
&CG1	MVC	&TO+&AL1.(&AL2),&FROM+&AL1
	MEND	

In chart 9-31 the macro call and generation using the macro definition in chart 9-30 are shown.

Chart 9-31. ANOP Generations

◆ STMNT	M	SOURCE	STATEMENT		
00200		FIRST	MOVER	540,OUT,INPUT	CALL
00201	M1	FIRST	MVC	OUT+0(256),INPUT+0	
00202	M1		MVC	OUT+256(256),INPUT+256	
00203	M1		MVC	OUT+512(28),INPUT+512	
00204		SEC	MOVER	60,OUTPUT,IN	CALL
00205	M1		MVC	OUTPUT+0(60),IN+0	

10. SPECIAL PURPOSE FEATURES

INTRODUCTION

◆ In Sections 7, 8, and 9, the facilities for writing and calling a basic to medium complex macro have been described.

This section describes the specialized features of the macro language. The extended features of the macro language allow the programmer to:

1. Terminate processing of a macro definition (see MEXIT);

2. Generate macro error messages (see MNOTE);

3. Use the system variable symbols (see &SYSNDX, &SYSECT, and &SYSLIST);

4. Assist in validating complex macro logic by utilizing the macro trace mode to indicate on the assembly listing the values of Set variable values and branches taken or not taken (see MTRAC and NTRAC).

ADDITIONAL GENERATOR COMMANDS

MEXIT Macro Definition Exit

General Description

◆ The MEXIT command indicates to the macro generator that processing of a macro definition is to be terminated. The MEXIT command is used in a macro definition when the programmer wishes to execute and generate only a portion of the definition.

Format

◆ The format of the MEXIT command is as follows:

NAME	OPERATION	OPERAND
A sequence symbol or blank.	MEXIT	Not used.

Specification Rules

Name Field

◆ A sequence symbol or blank.

Operation Field

◆ MEXIT.

Operand Field

◆ Not used.

Note

◆ If the macro generator processes a MEXIT statement, the next statement assembled is the statement following the call to the macro being generated.

Difference Between MEXIT and MEND

◆ The MEXIT command should not be confused with the MEND command. The MEND command indicates the end of a macro definition to the macro editor, as well as signifying the end of generation. Every macro definition must contain a MEND command even if the definition contains one or more MEXIT commands.

Example

◆ Chart 10-1 illustrates the use of the MEXIT command. The macro definition in this example is functionally the same as the macro definition in chart 9-26 (page 9-26). MEXIT has been used to show the flexibility available to the programmer.

Chart 10-1. MEXIT Command

NAME	OPERATION	OPERAND
	MACRO	
	MOVER	&NOCHAR,&TO,&FROM
&AL2	SETA	&NOCHAR
.RETURN	AIF	(&AL2 GT 256).LOOP
	MVC	&TO+&AL1.(&AL2),&FROM+&AL1
	MEXIT	
.LOOP	MVC	&TO+&AL1.(256),&FROM+&AL1
&AL1	SETA	&AL1+256
&AL2	SETA	&NOCHAR - &AL1
	AGOB	.RETURN
	MEND	

In chart 10-2 the macro calls and generation using the macro definition in chart 10-1 are shown.

Chart 10-2. MOVER Generation With MEXIT

STMNT	M	SOURCE	STATEMENT	
00300		MOVER	540,OUT,INPUT	CALL 1
00301	M1	MVC	OUT+0(256),INPUT+0	
00302	M1	MVC	OUT+256(256),INPUT+256	
00303	M1	MVC	OUT+512(28),INPUT+512	
00304		MOVER	97,OUT+540,RESULT	CALL 2
00305	M1	MVC	OUT+540+0(97),RESULT+0	

MNOTE Error Message Request

General Description

◆ The MNOTE command produces an error message in the program listing during generation. If any symbolic parameters or variable symbols are used in the Operand field, they are replaced by the values they represent. The MNOTE line will appear as a comment in the program listing without the quotation marks.

Format

◆ The format of the MNOTE command is as follows:

NAME	OPERATION	OPERAND
A sequence symbol or blank.	MNOTE	An error code, followed by a comma followed by the desired error message enclosed within quotation marks.

Specification Rules

Name Field

◆ A sequence symbol or blank.

Operation Field

◆ MNOTE.

Operand Field

◆ The error code is a decimal digit from 0 to 9. If the error code is omitted, 0 is assumed. As the error code value increases, the error becomes more serious.

Example

◆ The example in chart 10-3 illustrates the use of the MNOTE statement. This macro definition tests for the presence of the three parameters in the macro call. If any parameter is missing, an appropriate message is printed and generation of the macro is terminated.

Chart 10-3. MNOTE Command

```
◆ NAME      OPERATION   OPERAND

            MACRO
            MOVE        &NOCHAR,&TO,&FROM
            AIF         ('&NOCHAR' NE '').CHKTO
            MNOTE       'FIRST PARAMETER OMITTED'
  &BL1      SETB        1
  .CHKTO    AIF         ('&TO' NE '').CHKFR
            MNOTE       'SECOND PARAMETER OMITTED'
  &BL1      SETB        1
  .CHKFR    AIF         ('&FROM' NE '').TESTSW
            MNOTE       'THIRD PARAMETER OMITTED'
  .TERM     MNOTE       3,'GENERATION TERMINATED'
            MEXIT
  .TESTSW   AIFB        (&BL1).TERM
  &AL2      SETA        &NOCHAR
  .LOOP     AIF         (&AL2 LE 256).LSTMOV
            MVC         &TO+&AL1.(256),&FROM+&AL1
  &AL1      SETA        &AL1+256
  &AL2      SETA        &NOCHAR - &AL1
            AGOB        .LOOP
  .LSTMOV   MVC         &TO+&AL1.(&AL2),&FROM+&AL1
            MEND
```

SYSTEM VARIABLE SYMBOLS

♦ System variable symbols are local variable symbols that are assigned values during generation by the macro generator. There are three system variable symbols: &SYSNDX, &SYSECT, and &SYSLIST. They can be used in the Name field or Operand field of model statements except in the Name field of Set and Conditional commands. The value substituted for the variable symbol is the last value that the macro generator has assigned to the variable symbol. The &SYSLIST system variable symbol cannot be used with a keyword macro definition.

&SYSNDX Macro Call Index

♦ The system variable symbol &SYSNDX can be combined with other characters to create unique symbols for generated statements. If &SYSNDX is used in the Name field or Operand field of a statement that is part of a macro definition, the value substituted for &SYSNDX is the value assigned to it for the macro call being interpreted.

&SYSNDX is assigned a different value for each outer and inner macro call that is interpreted by the macro generator. &SYSNDX is assigned the value 0001 for the first macro call that is interpreted by the macro generator.

The value assigned to &SYSNDX for any other macro call is one plus the value assigned to &SYSNDX for the previous macro call. Throughout one use of a macro definition, the value of &SYSNDX can be considered a four-digit constant that is independent of any macro call in that definition. High-order zeros are not suppressed.

Note

♦ &SYSNDX can be combined with one to four other characters. The resulting Name must conform to other Names permitted in the Assembler (that is, it must begin with an alphabetic character).

Example

♦ One use of the &SYSNDX system variable symbol is shown in the macro definition in chart 10-4. In this example, A&SYSNDX provides a unique symbol in the Name field for branching to a particular instruction generated by the macro definition. In the example, the content of a field will not be moved if the first byte of the field is blank.

Chart 10-4. &SYSNDX Variable Symbol

NAME	OPERATION	OPERAND
	MACRO	
	MOVER	&NOCHAR,&TO,&FROM
	CLI	&FROM,X'40'
	BE	A&SYSNDX
	MVC	&TO(&NOCHAR),&FROM
A&SYSNDX	EQU	*

If the macro calls in chart 10-5 and chart 10-6 were the 106th and the 107th macro calls interpreted by the macro generator, the statements presented in chart 10-5 and 10-6 would be generated.

Chart 10-5. Generation with &SYSNDX Counter=0106

STMNT	M	SOURCE	STATEMENT	
00500			MOVER	20,PRINT,NAME
00501	M1		CLI	NAME,X'40'
00502	M1		BE	A0106
00503	M1		MVC	PRINT(20),NAME
00504	M1	A0106	EQU	*

Chart 10-6. Generation with &SYSNDX Counter=0107

STMNT	M		SOURCE STATEMENT	
00520			MOVER	15,PRINT,ADDR
00521	M1		CLI	ADDR,X'40'
00522	M1		BE	A0107
00523	M1		MVC	PRINT(15),ADDR
00524	M1	A0107	EQU	*

&SYSECT Current Control Section Name

◆ The system variable symbol &SYSECT gives the programmer the ability to generate a separate control section or dummy section during macro generation.

At the time of each macro call, &SYSECT is assigned a value that is the name of the CSECT or DSECT which contains the macro call.

It is possible for an inner macro to have a different value for &SYSECT from that assigned in the outer macro. This would occur where an outer macro contained a CSECT or DSECT statement before the inner call.

Example

◆ Chart 10-7 and chart 10-8 illustrate outer and inner macro calls taking on different &SYSECT values. Notice that when the macro call OUTER is given, the value of &SYSECT is PROGB, whereas when the macro call INNER is given, the value of &SYSECT is SUBRA. This is because SUBRAΔΔΔCSECT was given prior to INNER.

Chart 10-7. &SYSECT Variable Symbol

◆

NAME	OPERATION	OPERAND
	MACRO	
	OUTER	
SUBRA	CSECT	
	DS	100C
	DC	A(&SYSECT)
	INNER	SUBRB
&SYSECT	CSECT	
	MEND	
	.	
	.	
	.	
	.	
	MACRO	
	INNER	&ID
&ID	CSECT	
	DS	50C
	DC	A(&SYSECT)
	MEND	

Chart 10-8. &SYSECT Generation with Inner Call

◆

STMNT	M		SOURCE	STATEMENT	
01000		PROGA	CSECT		
01001			DS	200C	
⋮			⋮		
01500			OUTER		OUTER CALL
01501	M1	SUBRA	CSECT		
01502	M1		DS	100C	
01503	M1		DC	A(PROGA)	
01504	M1		INNER	SUBRB	INNER CALL
01505	M2	SUBRB	CSECT		
01506	M2		DS	50C	
01507	M2		DC	A(SUBRA)	
01508	M1	PROGA	CSECT		

&SYSECT for Minimum Generation

◆ The value of &SYSECT can be very useful to generate a subroutine in a new CSECT and on subsequent macro calls, only the linkage to the already generated subroutine is generated.

Example

◆ Based on the macro definition in chart 10-9, the first call to MOVEY generates the entire CSECT and the linkage, whereas the second call and subsequent calls will generate only the linkage. (Chart 10-10.)

Chart 10-9. &SYSECT for Minimum Generation

◆

NAME	OPERATION	OPERAND
	MACRO	
	MOVEY	
	AIF	(&BG101).LINK
&BG101	SETB	1
MOVEMAC	CSECT	
	NOP	0
	NOP	0
	NOP	0
	NOP	0
	NOP	0
	B	0(10)
&SYSECT	CSECT	
	B	AMOVEMAC+4
AMOVEMAC	DC	A(MOVEMAC)
.LINK	L	9,AMOVEMAC
	BALR	9,10
	MEND	

Chart 10-10. Generation for Subroutine and Linkage

```
◆ STMNT   M1   SOURCE  STATEMENT

  01000        PROGA     CSECT

  01001                  DS       200C
    .                     .
    .                     .
    .                     .
  02250                  MOVEY    , FIRST CALL

  02251   M1   MOVEMAC   CSECT

  02252   M1             NOP      0

  02253   M1             NOP      0

  02254   M1             NOP      0

  02255   M1             NOP      0

  02256   M1             NOP      0

  02257   M1             B        0(10)

  02258   M1   PROGA     CSECT

  02259   M1             B        AMOVEMAC+4

  02260   M1   AMOVEMAC  DC       A(MOVEMAC)

  02261   M1             L        9,AMOVEMAC

  02262   M1             BALR     9,10
    .                     .
    .                     .
    .                     .
  02300                  MOVEY    , SECOND CALL

  02301   M1             L        9,AMOVEMAC

  02302   M1             BALR     9.10
```

&SYSLIST Macro Operand Field

◆ The system variable symbol &SYSLIST provides the programmer with an alternate way to refer to macro call operand values. &SYSLIST and symbolic parameters can be used in the same macro definition.

&SYSLIST(n) refers to the nth value of a positional macro call. Symbol (n) can be an arithmetic expression. The &SYSLIST variable symbol cannot be used in a keyword macro definition.

Note

◆ The operand values in a positional macro call are referenced in the following manner:

(0)	=	Name field operand value.
(1) through (49)	=	Operand field values.

Examples

◆ The macro definition in chart 10-11 illustrates the &SYSLIST system variable symbol. Depending on the number of parameters included in the macro call, two, three, or four fields will be added. The result will be stored in the last field that is specified in the macro call operand.

Note that if &AL1 = 2, then &SYSLIST (&AL1) would be &SYSLIST(2) or refer to the 2nd operand value of FICA in chart 10-12. When the value of &SYSLIST (&AL2) is null, the last value (&SYSLIST(&AL1)) is stored.

Chart 10-11. &SYSLIST Variable Symbol

NAME	OPERATION	OPERAND	
	MACRO		
&NAME	ADD	&F1,&F2,&F3,&F4,&F5	
&NAME	ST	2,WORK	
	L	2,&F1	LOAD 1st VALUE
&AL1	SETA	2	
.ADD	A	2,&SYSLIST (&AL1)	ADD(N) Value
&AL1	SETA	&AL1+1	FIRST TIME = 3
&AL2	SETA	&AL1+1	FIRST TIME = 4
	AIFB	('&SYSLIST(&AL2)' ΔNEΔ''). ADD	
	ST	2,&SYSLIST(&AL1)	
	L	2,WORK	
	MEND		

Chart 10-12. Values Substituted in SYSLIST Macro

◆ STMNT	M	SOURCE STATEMENT	
02400		ADD	FTAX,FICA,STAX,BONDS,DEDUCT
02401	M1	ST	2,WORK
02402	M1	L	2,FTAX
02403	M1	A	2,FICA
02404	M1	A	2,STAX
02405	M1	A	2,BONDS
02406	M1	ST	2,DEDUCT
02407	M1	L	2,WORK
02500		ADD	REGHRS,OTHRS,TOTHRS
02501	M1	ST	2,WORK
02502	M1	L	2,REGHRS
02503	M1	A	2,OTHRS
02504	M1	ST	2,TOTHRS
02505	M1	L	2,WORK

TRACE COMMANDS

MTRAC Macro Trace

General Description

◆ The MTRAC command is available to assist the programmer in determining the effective conditional transfers within the macro logic.

Each conditional command (AGO,AGOB, AIF, AIFB, and ANOP) that is executed is printed on the assembly listing; a "Y" or "N" printed in column 80 indicates whether or not the branch was performed. A minus "- " in column 80 indicates an ANOP command or invalid statement.

Each Set command (SETA, SETB, SETC) which is executed is also printed on the assembly listing and its current value printed in columns 73-80.

a. SETA Variables: displayed as eight decimal digits in columns 73-80 and zero filled; negative values are displayed as a character value (X'D0'=0, does not print; X'D1'=1, prints as 'J';..., X'D9'=9, prints as 'R').

b. SETB Variables: displayed as a single character in column 80, 'T'=true or 1 value; 'F'=false or 0 value.

c. SETC Variables: displayed as one to eight characters, beginning in column 73, and space filled (Null values print as: --NULL--).

Format

◆ The format of the MTRAC command is as follows:

NAME	OPERATION	OPERAND
Not used.	MTRAC	Not used.

Specification Rules

Name Field

◆ Not used.

Operation Field

◆ MTRAC.

Operand Field

◆ Not used.

Notes

◆ 1. The command assumes that the NOGEN option is not in force.

2. This command can be used both inside and outside macros.

3. This command affects only macro generations following the MTRAC statement.

Example

◆ Chart 10-13 shows the macro definition, macro call, and generation with the MTRAC command in effect. Note that the MTRAC values are shown to the right. The actual listing was compressed for printing. Also, see charts 9-30 and 9-31.

Chart 10-13. Example of MTRAC Output

```
OBJECT CODE         M    SOURCE  STATEMENT

                                    MTRAC
                                    MACRO
                         &NAME      MOVER    &NOCHAR,&TO,&FROM
                         * EXAMPLE OF  MTRAC  COMMAND
                         &AL2       SETA     &NOCHAR
                         &CG1       SETC     '&NAME'
                         .LOOP      AIF      (&AL2 LE 256),LSTMOV
                         &CG1       MVC      &TO+&AL1,(256),&FROM+&AL1
                         &AL1       SETA     &AL1+256
                         &AL2       SETA     &NOCHAR-&AL1
                         &CG1       SETC     ''
                                    AGOB     .LOOP
                         .LSTMOV    ANOP
                         &CG1       MVC      &TO+&AL1,(&AL2),&FROM+&AL1
                                    MEND

                         ONE        MOVER    540,OUT,INPUT
                    M1   * EXAMPLE OF  MTRAC  COMMAND
                    M1   &AL2       SETA     &NOCHAR                      00000540
                    M1   &CG1       SETC     '&NAME'                      ONE
                    M1   .LOOP      AIF      (&AL2 LE 256),LSTMOV                N
D2 FF 30EB 3AAC     M1   ONE        MVC      OUT+0(256),INPUT+0
                    M1   &AL1       SETA     &AL1+256                     00000256
                    M1   &AL2       SETA     &NOCHAR-&AL1                 00000284
                    M1   &CG1       SETC     ''                           --NULL--
                    M1              AGOB     .LOOP                               Y
                    M1   .LOOP      AIF      (&AL2 LE 256),LSTMOV                N
D2 FF 31E8 3BAC     M1              MVC      OUT+256(256),INPUT+256
                    M1   &AL1       SETA     &AL1+256                     00000512
                    M1   &AL2       SETA     &NOCHAR-&AL1                 00000028
                    M1   &CG1       SETC     ''                           --NULL--
                    M1              AGOB     .LOOP                               Y
                    M1   .LOOP      AIF      (&AL2 LE 256),LSTMOV                Y
                    M1   .LSTMOV    ANOP                                         -
D2 1B 32E8 3CAC     M1              MVC      OUT+512(28),INPUT+512
```

NTRAC
No Trace

General Description

◆ The NTRAC command cancels the MTRAC function described on page 10-00.

Format

◆ The format of the NTRAC is as follows:

NAME	OPERATION	OPERAND
Not used.	NTRAC	Not used.

Specification Rules

Name Field

◆ Not used.

Operation Field

◆ NTRAC.

Operand Field

◆ Not used.

Notes

◆ 1. The command cancels the MTRAC function described on page 10-13.

2. The command can be used both inside and outside macros.

3. This command affects only macro generations following the NTRAC statement.

11. KEYWORD MACROS

INTRODUCTION

◆ 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 operand values in each macro call that corresponds to the definition, and to write the operand values in any order.

The positional macro call, as described in Section 8, required the operand values to be written in the same order as the corresponding symbolic parameters in the Operand field of the prototype statement (Section 7).

In the keyword macro definition, the programmer can assign standard values to any symbolic parameters that appear in the Operand field of the prototype statement. The standard value assigned is substituted for the symbolic parameter, if the programmer does not write anything in the Operand field of the macro call to correspond to the symbolic parameter. The maximum length of the standard value is eight characters.

When a keyword macro call is written, the programmer need only write one operand for each symbolic parameter value he wants to change.

Keyword macro definitions are prepared the same way as positional macro definitions (Section 7), except that the prototype statement is written differently, and &SYSLSIT may not be used in the definition.

KEYWORD MACRO PROTOTYPE STATEMENT

General Description

◆ The keyword macro prototype statement indicates to the assembly the format and mnemonic operation code of the keyword macro the assembly is to interpret. It must be the second statement of every macro definition. This type of prototype statement differs from the positional macro prototype only in regard to the equal sign (=) requirement and the standard value option. Otherwise, the specification rules given for the positional prototype apply also to the keyword prototype.

Format

◆ The format is as follows:

NAME	OPERATION	OPERAND
A symbolic parameter or blank.	A symbol.	Comma(,) or a maximum of 49 operands, separated by commas, of the form described below.

Specification Rules

Name Field

◆ See positional macro prototype statement (page 7-4).

Operation Field

◆ See positional macro prototype statement (page 7-4).

Operand Field

◆ The Operand field may contain a maximum of 49 operands separated by commas as follows:

1. Each operand must consist of a symbolic parameter, immediately followed by an equal sign (=) and optionally followed by a standard value.

   &PARAMTR= [STDVALUE]   MAXIMUM LENGTH

2. A standard value that is a part of an operand must immediately follow the equal sign.

3. All operands, except the last, must be immediately followed by a comma.

4. Anything that can be used in the Operand field of a macro call (except variable symbols), may be used as a standard value. For a further discussion of valid operand values see Section 8.

5. The last operand must be followed by a space instead of a comma.

6. The same symbolic parameter cannot be used more than once as part of an operand.

Examples

◆ The following are valid keyword macro prototype operands:

&TO=

&FROM=NAME

&SPACE=10

&V3T=X'FF'

Note

◆ The rules for continuation and absence of parameters are discussed in Section 7.

Example

◆ The sample keyword macro prototype in chart 11-1 contains a symbolic parameter in the Name field and nine operands in the Operand field. The mnemonic operation code is KEYMV. &INITIAL, &SPACE, and &AREA are assigned standard values whereas, the remaining six operands are not.

Chart 11-1. Keyword Macro Prototype

```
◆ M   SOURCE   STATEMENT

               MACRO

      &NAME    KEYMV    &INITIAL=10,&SPACE=5,&AREA=X
                        PRINT,&FRA=,&LNA=,&FRB=,&LNX
                        B=,&FRC=,&LNC=
```

KEYWORD MACRO CALL

General Description

◆ This is the second type of macro call format and it allows the values specified by each parameter to be used with a predefined keyword. The presence of the keyword allows the parameters to be specified in any order in the macro call.

Format

◆ The format for the keyword macro is as follows:

NAME	OPERATION	OPERAND
A symbol or blank.	Mnemonic operation code.	Comma(,) or a maximum of 49 operands, separated by commas, in the form described below.

Specification Rules

Name Field

◆ See positional macro call statement (page 8-3).

Operation Field

◆ The Operation field of a keyword macro call contains the same operation code that appears in the Operation field of the macro prototype.

Operand Field

◆ Each operand must consist of a keyword immediately followed by an equal sign and an optional value. Anything that can be used as an operand value in a positional macro call statement can be used as a value in a keyword macro call statement. The rules for forming valid positional operand values are detailed in Section 8.

A keyword consists of a maximum of seven letters and digits, the first of which must be a letter.

The keyword part of each keyword macro call must correspond to one of the symbolic parameters that appears in the Operand field of the keyword prototype statement; that is, the keyword portion must be identical to the characters of the symbolic parameter that follows the ampersand (&).

NAME	OPERATION	OPERAND	
&NAME	KEYMV	&INITIAL=10,......	PROTOTYPE OPERAND
	KEYMV	INITIAL=30,	CALL OPERAND

Examples

◆ The following are valid keyword macro call operands:

TO=WORK

FROM=

SPACE=15

V3T=X'40'

Operand Order

◆ The operands in a keyword macro call can be written in any order. If an operand appeared in a keyword macro prototype statement, a corresponding operand does not have to appear in the keyword macro call statement. Because the operands can be written in any order if an operand is omitted, the comma that would have separated it from the next operand need not be written.

Operands can appear on separate cards. A comma must follow every operand except the last, and the continuation column must contain a non-blank character. Comments can be contained on the separate cards that contain individual operands.

Replacement Rules

◆ Rules used to replace the symbolic parameters in the model statements of a keyword macro definition are as follows:

1. If the symbolic parameter appeared in the Name field of the macro prototype, and the corresponding characters of the macro call are a symbol, the symbolic parameter in the Name field is replaced by the symbol. Otherwise, the symbolic parameter in the Name field is considered to be a null parameter.

2. The value associated with each keyword in the macro prototype becomes the value of the symbolic parameter unless a value is associated with each keyword specified in an operand of the macro call. In this case, the value in the macro call replaces the value obtained from the prototype for the symbolic parameter.

Example

◆ In Chart 11-2 a keyword macro definition is illustrated. This definition will be used in succeeding charts.

Chart 11-2. Keyword Macro Definition

```
M  SOURCE STATEMENT

           MACRO
   &NAME   KEYMV  &INITIAL=10,&SPACE=5,&AREA=PRINT,&FRA=,&LNA=,&FRB=,&LNB=C
                  ,&FRC=,&LNC=
   * EXAMPLE OF A KEYWORD MACRO
   &AL1    SETA   &INITIAL                 SET INITIAL
   &AL2    SETA   0                        LAST LENGTH
   &AL3    SETA   &SPACE                   SPACES BETWEEN
   &AL4    SETA   &AL1                     NEXT POSITION
           AIF    ('&FRA'ΔEQΔ'').TRYB
           MVC    &AREA+&AL4(&LNA),&FRA
   &AL2    SETA   &LNA
   &AL4    SETA   &AL4+&AL2+&AL3
   .TRYB   AIF    ('&FRB'ΔEQΔ'').TRYC
           MVC    &AREA+&AL4(&LNB),&FRB
   &AL2    SETA   &LNB
   &AL4    SETA   &AL4+&AL2+&AL3
   .TRYC   AIF    ('&FRC'ΔEQΔ'').END
           MVC    &AREA+&AL4(&LNC),&FRC
   .END    MEND
```

Example

◆ Chart 11-3 illustrates a call and generation for the definition in chart 11-2. Notice two standard values (INITIAL and AREA) are used and FRA and LNA are given values.

Chart 11-3. Keyword Call Using Standard Values

STMNT	M	SOURCE	STATEMENT
01000		KEYMV	FRA=NAME,LNA=20
01001	M1	MVC	PRINT+10(20),NAME

Example

◆ Chart 11-4 illustrates SPACE used as standard, INITIAL and AREA changed.

Chart 11-4. Keyword Call Replacing Standard Values

STMNT	M	SOURCE	STATEMENT	
01002		KEYMV	INITIAL=30,AREA=WORK,	C
01003			FRB=ADDR,FRC=CITY,	C
01004			LNB=15,LNC=25	
01005	M1	MVC	WORK+30(15),ADDR	
01006	M1	MVC	WORK+50(25),CITY	

Null Parameters

◆ Null parameters in a keyword macro definition are processed in the same way as in the positional macro definition. Null parameters are formed under any of the following conditions:

1. If a symbolic parameter appears in the Name field of a macro prototype statement and the Name field of a macro call is blank, a null parameter is formed.

2. If a keyword is specified in the Operand field of a macro call and no value is associated with the keyword, a null parameter is formed, regardless of the presence of a standard value in the prototype statement.

3. If no standard value is associated with a keyword in the Operand field of a keyword prototype statement, and the keyword and its associated value are omitted from the Operand field of a macro call, a null parameter is formed.

Example

◆ Using the macro definition in chart 11-2, chart 11-5 illustrates the creation of null parameters. A null parameter is created for INITIAL, although it contained a standard value. FRA, LNA, FRB, and LNB have been created as null parameters by the omission of these keywords.

Chart 11-5. Keyword Null Parameters

STMNT	M	SOURCE STATEMENT		
01020		KEYMV	INITIAL=,	C
01021			FRC=CITY,	C
01022			LNC=25	
01023	M1	MVC	PRINT+0(25),CITY	

Note

◆ In chart 11-5, AREA retains its standard value and SPACE (still 5) is not used with this macro call.

APPENDIX A

SUMMARY OF ASSEMBLY INPUT/OUTPUT

INPUT

◆ Input to the Assembly System consists of symbolic source language statements punched as described on page 2-1. These source statements are normally contained on cards but may be on magnetic tape in card image format or in blocked format (except 70/25). In addition, source corrections can be applied against a source library tape with the TOS/TDOS Assembler.

Macro definition statements may be included within the source deck and macro expansion accomplished without referencing the macro library.

See appropriate Operators' Guide for detailed information on control cards, device assignments, and deck composition.

OUTPUT

◆ The normal Assembly output consists of two major "files"; namely, the Object Program and the Program Listing. A summary of each output type is described below.

Object Program Output

◆ Five different types of cards may be produced by the Assembly. A brief description of each is shown below. For complete format information, refer to the Spectra 70 Systems Standards Manual.

ESD Card (External Symbol Dictionary)

◆ This card specifies the EXTRNSs, ENTRYs, V-CONs, and COMS defined for a program. ESD cards supply all the necessary information to link together program sections to form an operating program. For instance, the ESD card contains all symbols defined in this assembly which are referred to in another assembly, (ENTRYs) and all symbols referred to by this section which are defined in some other assembly (EXTRNs).

TXT Card (Generated Program Text)

◆ The generated machine instructions to be loaded into storage are contained on TXT cards. The address of the instructions or data and the number of bytes are contained within card. The TXT cards will be modified as required by RLD information (see below).

RLD Card (Relocation Dictionary)

◆ The RLD card identifies portions of the TXT card which must be modified due to relocation (that is, floated). The RLD cards provide the information necessary to perform the relocation and are intermingled with the TXT cards. However the TXT card to which the RLD card refers is always produced first.

XFR Card (Transfer)

◆ The XFR card is only produced by the Assembler at the point in the text where specified by the XFR Assembler instruction. This card is used by the Program Loader and Linkage Editor routines to define the transfer point or entry point of a phase, overlay. (Not produced in TOS/TDOS assembly.)

END Card

◆ The END card is always generated by the assembly and indicates the end of a program section or object module.

Program Listing Formats

◆ The Assembly System produces three basic listings. These listings may be eliminated by use of the AOPTN instruction (70/25 and POS) or by specifying the ASMLST option of the TOS and TDOS monitors. Each listing type is described below.

ESD Listing

◆ The ESD listing lists each Control section (CSECT) and Dummy section (DSECT) that is defined in the program. The ESID number, assembled origin, and size of the section are also provided. A list of the EXTRNs and V-CONs is provided with their ESID number. The format of the ESD listing is shown below.

```
SAMPLE ASSEMBLY PROGRAM POS

      SYMBOL    TYPE ID   ADDR   LENGTH                              EXTERNAL SYMBOL DICTIONARY

      BEGIN      SD  01   02710  01550
      SEQ2       SD  02   03C60  000A8
      SEQ3       SD  03   03D08  00030
      SEQ4       SD  04   03D38  003A8
```

SYMBOL - contains the name of the control section, EXTRN, V-CON, or ENTRY assembled. DSECTS are preceded with the word "dummy" enclosed in parentheses.

Note:

ENTRY symbols are followed by an asterisk when the symbol specified is undefined, defined in a dummy section, or defined in an unnamed control section.

TYPE - Contains a two-character code identifying the element as:

1. TYPE SD (Section Definition)

   The name, assembled origin, length (adjusted to a double-word boundary), and ESID number of each control section (CSECT) or dummy control section (DSECT) are listed.

2. TYPE ER (External)

   The name and ESID number of each symbol specified as an EXTRN.

3. TYPE VC (V-CON)

   The name and ESID number of each symbol specified as a V-CON are listed. This is not produced on the 70/25.

4. TYPE CM (Common)

   The name and ESID number of each symbol specified as COM (valid on TOS/TDOS only).

ESD Listing *(Cont'd)*

5. TYPE LD (Entry)

   The name, address, and ESID number of each symbol specified as an ENTRY are listed.

ID - contains a two character ESID number that is assigned to the element.

ADDR - contains the assembled origin address of the element.

LENGTH - contains the length (hexadecimal) of the section assembled.

Symbol Table Listing

◆ The symbol table listing contains all the symbols used (including FCP) listed alphabetically, four to a line. The ESID number of the control section in which the symbol is defined, the length, and the address of the symbol are provided. A sample symbol table listing is reproduced below.

SAMPLE ASSEMBLY PROGRAM POS

SYMBOL	CSECT	VALUE	L	SYMBOL	CSECT	VALUE	L	SYMBOL	CSECT	VALUE	L	SYMBOL	CSECT	VALUE	L
A	02	03C62	04	ADA1	02	03CC8	02	ADB1	02	03CC4	02	ADC1	02	03CC6	02
ADDITION	02	03CCA	06	B	03	03D0A	04	BASE	01	03BC8	04	BASE1	UNDEFINED		
BASE2	UNDEFINED			BASE3	04	03D44	04	BEGIN	01	02710	01	C	04	03D3A	04
CARDIN	01	02864	01	CARD1	04	03EF8	50	CARD2	04	03F48	50	CCBREAD	01	03C20	05
CCBTYPER	01	03BFC	05	CCWREAD	01	03C28	02	CCWTYPER	01	03C04	02	CDIN	04	03F98	50
CHECK1	02	03C80	06	END1	04	04094	04	END3	04	040B4	04	EXIT	04	03DAE	04
FINAL	04	040C8	04	HALT5	04	03DB2	04	HSKP	01	03BDC	04	IFABL1	01	02A84	02
IFACBB	01	02A8C	04	IFACCB	00	00006	01	IFACKP	01	02A2A	04	IFADEV	00	0000D	01
IFAFIN	00	00000	01	IFAFNA	00	00000	01	IFAIO1	01	02A82	02	IFALAB	00	00003	01
IFALB	01	02A24	04	IFALBZ	00	00000	01	IFALXX	01	029AC	04	IFAMKS	01	02A80	01
IFAMK1	00	00020	01	IFAMK2	00	00004	01	IFAMK3	01	02B30	01	IFAMK4	01	02A90	01
IFAMSS	01	02A88	01	IFAMVD	01	02A36	06	IFANWS	01	02A7C	04	IFAREA	00	00002	01
IFARGS	01	02B2C	04	IFASER	01	02974	04	IFASTA	00	00004	01	IFASTB	01	029DC	06
IFASTD	01	02A0E	06	IFASTO	01	02A46	06	IFASTT	01	02988	04	IFASTU	01	029C8	04
IFASTW	01	029C2	06	IFASTY	01	029D6	06	IFATAB	01	02B32	01	IFATAP	00	00018	01
IFATWO	01	02A8A	01	IFAVE	01	0296C	04	IFAV14	01	0361C	04	IFAW2	01	02B31	01
IFAZRO	00	000FF	01	IFAZZ	00	0000F	01	IFA04	00	00004	01	IFBADR	01	0290C	06
IFBARW	01	02B3E	02	IFBLFD	01	02EA2	06	IFBLKS	00	00008	01	IFBNSW	01	02B27	01
IFBSTT	01	029DC	06	IFBTDV	01	02A1C	04	IFBVLB	01	031D8	04	IFB01	01	02A87	01
IFB04	01	02A86	01	IFCBSF	01	03360	06	IFCBSR	01	03356	06	IFCBYT	01	034AB	01
IFCCN1	01	02B0E	04	IFCCN2	01	02B12	04	IFCCWA	00	00006	01	IFCCWB	01	031DC	06
IFCCWC	01	034A3	02	IFCCWD	01	0348F	02	IFCCW4	01	03212	06	IFCCW6	01	03224	04
IFCCW9	01	03234	04	IFCDP9	00	00004	01	IFCD10	00	00005	01	IFCD23	00	0000D	01
IFCERG	01	0337E	06	IFCERP	01	034AC	01	IFCFSF	01	03374	06	IFCFSR	01	0336A	06
IFCISS	01	033D8	06	IFCKID	01	02B28	0C	IFCKPA	00	00022	01	IFCLHE	00	0000A	01
IFCLSE	01	02A96	04	IFCMK2	00	00040	01	IFCNT	01	03474	0D	IFCNTT	01	03495	02
IFCNTX	01	03497	02	IFCOM	01	02A5E	04	IFCON1	01	02B51	02	IFCON2	01	02B53	02
IFCP	00	0004C	01	IFCPA	01	02B6C	04	IFCPIN	01	028F2	04	IFCPOP	00	00000	01
IFCPOU	01	02926	04	IFCRBA	01	03487	05	IFCREW	01	03398	06	IFCRGC	00	00018	01
IFCRUN	01	0334C	06	IFCRWA	01	03481	02	IFCSKB	01	034A2	01	IFCSKC	00	00008	01
IFCSKD	01	0349B	01	IFCSKE	01	0349A	01	IFCSKM	01	033CC	06	IFCSKN	01	033BC	06
IFCSK0	00	00020	01	IFCSK1	00	000FF	01	IFCSK2	01	0349C	01	IFCSK3	01	0349D	01
IFCSK4	01	0349E	01	IFCSK5	01	0349F	01	IFCSK6	01	034A0	01	IFCSK8	01	034A1	01
IFCSOP	01	02AF0	06	IFCSPM	01	033B2	06	IFCSPN	01	033A2	06	IFCTDV	01	02AC4	06
IFCTHR	01	034AE	01	IFCTLA	00	0000A	01	IFCTLB	00	0000E	01	IFCTL2	01	03336	06
IFCTRL	01	03326	04	IFCTWL	01	034AD	01	IFCWTM	01	0338E	06	IFCXIT	01	03404	06
IFCZER	01	02B08	01	IFC12	00	0000C	01	IFC13	00	0000D	01	IFC14	00	0000E	01
IFDEC1	01	02B22	01	IFDEC2	01	02B23	01	IFDEVA	00	00001	01	IFDEVR	00	0000E	01
IFDMVM	01	03460	04	IFDPBT	01	03438	06	IFDP10	00	00005	01	IFDP11	00	0001A	01
IFDP12	00	0000C	01	IFDP14	00	00010	01	IFDP16	00	00014	01	IFDP19	00	0000E	01
IFDP2	00	00002	01	IFDP21	00	0001C	01	IFDP23	00	0000C	01	IFDP24	00	0000D	01
IFDP25	00	0000E	01	IFDP27	00	00012	01	IFDP30	00	00010	01	IFDP32	00	00020	01
IFDP35	00	00020	01	IFDP39	00	00019	01	IFDP41	00	00022	01	IFDP8	00	00008	01
IFDR24	00	0000D	01	IFDSP9	00	00004	01	IFDSTM	01	02A16	06	IFDSTT	01	029E2	06
IFDS1	00	00001	01	IFDS10	00	0000A	01	IFDS11	00	0000B	01	IFDS18	00	00012	01
IFDS2	00	00002	01	IFDS20	00	0000C	01	IFDS24	00	00018	01	IFDS26	00	00014	01
IFDS9	00	00004	01	IFDTST	01	03442	06	IFDVST	01	02B1E	01	IFDVSW	01	0341A	06
IFD7LV	01	02B10	01	IFECCB	01	02B00	05	IFECCW	01	02B08	02	IFEGTN	01	02B4D	01
IFEMSS	01	02B0E	0F	IFEM18	01	02B1C	0F	IFEOJ	01	02E5E	04	IFERRC	00	0001A	01
IFERR1	01	02A50	04	IFERR6	01	02AFA	04	IFETCH	01	02B56	02	IFETWL	01	02A89	01
IFFEOA	01	0331A	04	IFFEOV	01	032F4	04	IFFFFF	01	02B44	02	IFGP	01	02B46	04
IFGPEF	01	02BF4	04	IFGPGO	01	02B64	04	IFGPS1	01	02C06	04	IFGPWK	01	02FEC	04
IFGP1	01	02B78	04	IFGP1B	01	02BA6	04	IFGP1C	01	02B98	04	IFGP3	01	02B72	06
IFGP4	01	02BB6	04	IFGP5	01	02BFE	04	IFGP6	01	02C18	04	IFGP7	01	02C20	06
IFGP8	01	02B5C	04	IFGWRC	01	02EE6	02	IFIA	00	00002	01	IFIB	00	00003	01
IFIBLB	00	00005	01	IFIBLK	00	00006	01	IFICKL	00	00006	01	IFICOM	00	00004	01

Symbol Table Listing *(Cont'd)*

CSECT - contains the section number in which the symbol was defined.

VALUE - contains the address of the symbol.

L - contains the length attribute assigned to the symbol.

Note:

Symbols that are not defined are listed with the word "undefined" following them.

TEXT Listing

◆ The TEXT listing contains the generated machine instructions associated with each source statement. A sample TEXT listing is shown on the following page. Each field of a TEXT listing is described below.

FLAGS: Six positions are provided for error flags. The interpretation of these flags is discussed on page A-9.

LOCTN: The location counter or appropriate address.

OBJECT CODE: The assembled object code is listed.

ADDR1,ADDR2: These fields show the resolved address of the operand, where applicable. This facilitates the use of this listing. For example, if the object code specifies that operand one uses register 4 as the base register with a displacement of $(100)_{16}$, and the USING statement directed the Assembler to assume register 4 contained $(1000)_{16}$, then ADDR1 is listed as $(01100)_{16}$.

STMNT: (TOS/TDOS) This field contains a sequential statement number assigned to each statement. This number is used as the statement reference for the optional Cross Reference Listing.

M Field:

- 70/25: If this field contains an M, it specifies that the current line was generated (expanded) by a macro.
- POS/TOS/TDOS: In addition to M, a second character is appended to indicate the depth (that is, nesting) of the macro which generated the line; for example, M1, M2, or M3.

SOURCE STATEMENT: This field contains the user's source statement (or the source statement generated by a macro expansion).

```
FLAGS  LOCTN OBJECT CODE       ADDR1 ADDR2 STMNT M  SOURCE STATEMENT

       TXT CARD # IS 0022.
       05552 47 80 6532         05612       00501              BE    MERGE4                                                C 00034700
       05556 41 DD 0006         00006       00502              LA    13,6(13)               POINT TO NEXT SLOT             P 00034800
       0555A 41 CC 0001         00001       00503              LA    12,1(12)                                                00034900
       0555E 55 D0 600C         050EC       00504              CL    13,GLSCNT              END OF NAME TABLE?            DY 00035000
       05562 47 70 6464         05544       00505              BNE   MERGE2                                                C 00035100
       05566 58 D0 6D74         05E54       00506              L     13,GADRREM             POINT TO FIRST SLOT IN NEW     P 00035200
       0556A 58 C0 6014         050F4       00507              L     12,G#OFSYM             AREA                           C 00035300
       0556E 55 D0 6D78         05E58       00508    MERGE6    CL    13,GSLOT               ARE WE BACK TO ORIGINAL SLOT  DN 00035400
       05572 47 80 64B4         05594       00509              BE    MERGE5                                                C 00035500
       05576 95 00 D000                     00510              CLI   0(13),0                EMPTY SLOT?                   DN 00035600
       0557A 47 80 64B8         05598       00511              BE    MERGE3                                                C 00035700
       0557E D5 05 A000 D000                00512              CLC   0(6,10),0(13)          TAG=CONTENTS OF SLOT?         DN 00035800
       05584 47 80 6532         05612       00513              BE    MERGE4                                                C 00035900
       TXT CARD # IS 0023.
       05588 41 CC 0001         00001       00514              LA    12,1(12)                                                00036000
       0558C 41 DD 0006         00006       00515              LA    13,6(13)               POINT TO NEXT SLOT             P 00036100
       05590 47 F0 648E         0556E       00516              B     MERGE6                                                B 00036200
       05594 41 C0 0001         00001       00517    MERGE5    LA    12,1                   SET NEW SLOT=1                 P 00036300
       05598 50 90 6D98         05E78       00518    MERGE3    ST    9,GSAVE9              SAVE 9(=SLOT IN PARTIAL TABLE)  P 00036400
       0559C 58 F0 6D8C         05E6C       00519              L     15,GWORD              LOAD R15 WITH 4 BYTES OF TAG    P  00036500
       055A0 17 EE                          00520              XR    14,14                 CLEAR 14 FOR MULTIPLY              00036600
       055A2 48 20 6014         050F4       00521              LH    2,G#OFSYM            R2=#TAGS IN BASIC TABLE          P  00036700
       055A6 1D E2                          00522              DR    14,2                  RANDOMIZE TAG TO SLOT IN        P  00036800
       055A8 18 FE                          00523              LR    15,14                PUT REM(=SLOT) INTO R1K          P  00036900
       055AA 18 9E                          00524              LR    9,14                 R14 AND R9 BOTH = SLOT NO        P  00037000
       055AC 41 20 0006         00006       00525              LA    2,6                   RESTORE R2 TO 6                    00037100
       055B0 1C E2                          00526              MR    14,2                                                     00037200
       055B2 5A F0 6D60         05E40       00527              A     15,GADRNT              COMPUTE ADDRESS OF SLOT        P 00037300
       055B6 18 E9                          00528              LR    14,9                                                    00037400
       055B8 95 00 F000                     00529    MERGE7    CLI   0(15),0               EMPTY SLOT                      D  00037500
       055BC 47 80 6522         05602       00530              BE    MERGE9               (NO,YES-MERGE9)                  C  00037600
       TXT CARD # IS 0024.
       055C0 D5 05 A000 F000                00531              CLC   0(6,10),0(15)        TAG =CONTENTS OF SLOT           DN 00037700
       055C6 47 80 6520         05600       00532              BE    MERGE8                                                C 00037800
       055CA 41 EE 0001         00001       00533              LA    14,1(14)               POINT TO NEXT SLOT             P 00037900
       055CE 41 FF 0006         00006       00534              LA    15,6(15)                                                00038000
       055D2 55 F0 6D74         05E54       00535              CL    15,GADRREM             END OF INITIAL TABLE          DY 00038100
       055D6 47 70 64D8         055B8       00536              BNE   MERGE7                                                C 00038200
       055DA 41 E0 0000         00000       00537              LA    14,0                   POINT TO START OF INITIAL      P 00038300
       055DE 58 F0 6D60         05E40       00538              L     15,GADRNT              TABLE                          C 00038400
       055E2 95 00 F000                     00539    MERGE10   CLI   0(15),0               EMPTY SLOT                      D  00038500
       055E6 47 80 6522         05602       00540              BE    MERGE9               (NO,YES-MERGE9)                  C  00038600
       055EA D5 05 A000 F000                00541              CLC   0(6,10),0(15)          TAG=CONTENTS OF SLOT?         DN 00038700
       055F0 47 80 6520         05600       00542              BE    MERGE8                                                C 00038800
       055F4 41 EE 0001         00001       00543              LA    14,1(14)               POINT TO NEXT SLOT             P 00038900
       TXT CARD # IS 0025.
       055F8 41 FF 0006         00006       00544              LA    15,6(15)                                                00039000
       055FC 47 F0 6502         055E2       00545              B     MERGE10                                               B 00039100
       05600 18 CE                          00546    MERGE8    LR    12,14                  SET NEW SLOT#=TO OLD SLOT#       00039200
       05602 48 50 6014         050F4       00547    MERGE9    LH    5,G#OFSYM          IS SLOT IN NEW AREA                DY 00039300
       05606 15 C5                          00548              CLR   12,5                                                    00039400
```

TEXT Listing *(Cont'd)*

CARD NUMBER: (70/25 and POS) — The rightmost columns of this listing specify the card number of the object card which contains the TEXT information. The listing does not give a card number for cards generated by PUNCH, REPRO, XFR, or END. The output cards contain a card number in columns 77-80, unless these cards were produced by a PUNCH or REPRO statement. Cards produced prior to TEXT information (for example, ESD cards) are not numbered.

CARD NUMBER (TOS/TDOS) — The card number of the TXT or END card containing the generated coding is specified in the OBJECT CODE column. Printing of the card number is controlled by the NONUM operand of the PRINT statement.

PROGRAM CONTROL INFORMATION (TOS/TDOS)

◆ The Monitor PARAM message may optionally be used to designate (or omit) specific input/output files. In order to change the configuration assumed by the Monitor, the following operand entries are required:

Param Operand	Meaning
TAPE=NO	Indicates that tape output is not to be generated.
CARD=YES	Indicates that card image output is to be written to SYSOPT.
INPUT=symbolic	Source input device, if other than SYSIPT. (See Source Language Correction.)
OUTPUT=symbolic	Indicates symbolic device, if other than SYSUT1, that is to receive the generated Object Module File(s).
WORK=YES	Indicates assignment of the additional work tape SYSUT4.
LIBRY=NO	Indicates absence of Macro Library.
SOURCE=symbolic	Updated source symbolic device, if other than SYSUT5. (See Source Language Correction.)
ERRLST=NO	Indicates that a listing of error flags is not to be printed.
ASMLST=NO	Indicates that the program listing is not to be produced. Statements containing errors, however, will be printed.
XREF=YES	Indicates that a Cross-Reference listing is to be produced.
MAP=NO	Indicates that the Symbol Table listing is not to be produced.

CROSS REFERENCE LISTING OPTION

General Description

♦ The cross reference option in the TOS/TDOS assembly provides a list of symbols, defined or referenced in the source listing, and the statement numbers in which reference or definition took place. The symbols are listed in the same order as they appear in the symbol table listing. This option is generated by the XREF=YES entry in the PARAM card. (See page A-6.)

Notes

♦ 1. Each symbol is shown in the left column of the listing.

2. The statement numbers referencing or defining the symbol are shown to its right.

3. The statement number which defines the symbol is flagged with an asterisk.

   a. If a symbol is multiply defined, each statement defining the symbol will be flagged with an asterisk.

   b. If a symbol is undefined, none of the statement numbers referencing the symbol will be flagged with an asterisk.

4. Double or single spacing when a new symbol is printed is controlled by the PRINT instruction. (See page 5-4.)

   a. OPEN (preset) - Double spacing.

   b. CLOSED - Single spacing.

5. Continuation lines for a given symbol are single spaced, regardless of PRINT option.

6. If the references to a symbol cause a new page to be printed the symbol is again printed with the first line of references to it on the new page.

   For example, in the sample Cross Reference Listing shown on page A- 8, the symbol GRWD is defined in statement number 715 and referenced in statement numbers 657 and 663. If GRWD had been undefined in this assembly, no asterisk would appear. If GRWD had been defined more than once, an asterisk would appear adjacent to each defining statement number.

CROSS REFERENCE LISTING

SYMBOL	REFERENCES						
GREADPNT	00418	00419	00448	00457	00472	00692*	
GRHPNT	00424	00431	00479	00563	00702*		
GRWD	00657	00663	00715*				
GSAVE11	00561	00564	00744*				
GSAVE9	00518	00554	00747*				
GSIZE	00585	00586	00729*				
GSLOT	00496	00508	00738*				
GTM	00435	00450	00570	00713*			
GUNWTM	00391	00721*					
GWAIT	00661	00665	00667	00714*			
GWORD	00485	00486	00519	00743*			
GWRENC	00394	00439	00641	00653	00655	00676*	
GWRHPNT	00423	00441	00443	00444	00446	00697*	
GWRNEWNT	00413	00415	00471	00659	00687*		
GWRTM	00454	00707*					
GZERO	00596	00739*					
G4096	00402	00404	00734*				
G6	00411	00746*					
G#NEWSYM	00477	00488	00669	00740*			
G#OFSYM	00012*	00427	00497	00507	00521	00547	00670	00671
INITIAL1	00228	00390*					
INITIAL2	00393	00396*					
INITIAL3	00403	00405*					
INITIAL4	00466*	00469					
INITIAL5	00460*	00464					
JRECEND	00578	00632	00748*				

APPENDIX B

ERROR FLAGS

Table B-1. Error Flags

Flag	Condition
A	Invalid Address: ● An address expression specifies multiplication or division of two relocatable operands. ● The final value of an address exceeds 2^{19}-1. ● The intermediate value of an address exceeds 2^{31}-1. ● The displacement of an explicit address (base, register, displacement), exceeds 2^{12}-1. ● An address expression is complex relocatable, but is not in an A or Y type constant.
B	Incorrect Control Statement: ● Incorrect ICTL statement. ● Incorrect ISEQ statement. ● START card incorrectly placed in the source deck.
D	Incorrect Specification: ● Operand in START card not set to double-word boundary. ● Ampersand in character string is specified as & rather than &&. ● Incorrect type code in a DC or DS statement. ● Invalid register number used in USING statement. ● Invalid operand in MCALL statement. ● Invalid scaling defined in DC. Invalid Address: ● L Field not correct in DC or DS statement. ● Location counter set to odd location when CNOP instruction executed. Warning only. ● S type constant specified in a literal. ● Constant string not terminated by a quote in DC or DS statement. ● DC statement does not contain data in constant field or illegal character present in constant field. ● Length specification is incorrect in machine instruction. ● Address is not aligned to appropriate boundary. ● Source cards not in sequence. (Produced only if ISEQ specified.)

ERROR FLAGS
(Cont'd)

Table B-1. Error Flags *(Cont'd)*

Flag	Condition
E	Syntax Error: ● Illegal character in source statement. ● Symbol exceeds eight characters. ● Symbol does not begin with an alphabetic character. ● A required character is not present. ● Consecutive arithmetic operators. ● PRINT statement error. ● Expression in machine instruction is too complicated: (that is, nest of parentheses exceeds three). ● Two literals in one statement. ● Error in AOPTN card.
H	● Location counter exceeds $2^{19}-1$.
I	● Incorrect immediate data or self-defining term.
L	● The number of CSECT and DSECT statements exceeds 32. ● The number of literal pools exceeds 33. ● The number of CSECT, DSECT, EXTRN, and V-CON statements exceeds 255. ● The number of ENTRY statements exceeds 100. ● Incorrect specification in CSECT or DSECT. ● Unpaired DSECT symbol in an A or Y address constant.
M	● Symbol is multiply defined. ● Symbol defined in a statement which caused the location counter to exceed $2^{19}-1$. ● Symbol defined as ENTRY in unnamed CSECT or DSECT. ● Symbol equated to an incorrect symbol.
O	● Invalid character in operation code. An HB instruction is generated which branches to the next instruction (that is, HB *+4). (See note 2 on page C-1.) ● Illegal operation code or macro not found in library.
P	● Privileged instruction used. (Not set by 70/25 Assembler.)

ERROR FLAGS (Cont'd)

Table B-1. Error Flags *(Cont'd)*

Flag	Condition
Q	• An error was detected in an ORG or EQU statement. • Symbol equated to a relocatable symbol in a different control section (see page 1-2).
S	• Illegal symbol in the Name field.
T	<u>Incorrect Macro Translation</u> All macro errors are noted by a special MNOTE message. However, the following conditions, which still allow macro expansion to continue, result in the T flag: 1. Operation code is not legal for generation in a macro. 2. The generated statement is too large. 3. Incorrect format in MNOTE message. 4. A nongenerative statement contains an error or potential error. Macro processing continues with the statement treated as an ANOP and the first line of the statement is listed. <u>The primary error conditions are:</u> • Final character value longer than eight characters. • Intermediate character string longer than 16 characters. • *Illegal operand in arithmetic operation. • *Overflow in arithmetic operation. • Incorrect type operands in boolean expression. • Syntax error in the statement. • An illegal or undefined variable symbol contained within the statement. (See Note on page C-1.)
U	Undefined symbol (in evaluating expressions, the defined symbol is assumed to be absolute with a value of 0 and a length attribute of 1).
Y	A base register cannot be found to resolve the specified implied address.
Z	The symbol table is full. See Appendix D for specified symbol limits.

APPENDIX C

MACRO ERROR FLAGS

◆ The flag field, for errors detected in macro-expansion, other than those noted by T flags, contains "MAC_ER". An MNOTE message, displayed in the source statement field, describes the error.

Any type of error encountered in macro processing effects one assembler generated MNOTE message to be produced for each outer macro instruction call. The form of the message in the source statement field is:

	OPERATION	OPERAND
*	M N O T E	* , C P X G

If any of the letters are not present, then the appropriate field is left blank.

The letter codes in the Operand field designate the following types of errors:

C = An error condition, exclusive of a bad prototype statement, prohibits the called macro from being processed and macro expansion terminates. The error conditions are:

1. calling statement incorrect. (Examples: an operand contains more than eight characters, keyword misspelled, more than 49 parameter values in call line, etc.)
2. unidentified operation code.
3. nesting greater than 3.
4. more than 50 unique source deck macros have been called (70/25 and POS). The limit in TOS and TDOS is 75.
5. keyword parameter specified more than once in macro parameter. Expansion terminated.

P = The prototype statement of the called macro is in error. Macro expansion terminates.

X = An invalid sequence symbol or a sequence symbol which does not exist was specified in an AIF, AIFB, AGO, or AGOB statement. The macro expansion is terminated.

G = Generated statement is <u>bad</u>, invalid op code, or miscellaneous arithmetic errors.

Notes:

1. MNOTE, G indicates the macro involved, <u>not</u> the statement, which is indicated by a "T" flag.
2. If a model line generates an unidentifiable or syntactically incorrect operation code, macro expansion is not terminated in the POS, TOS, or TDOS Assemblies. Instead, three NOPR instructions are generated. This feature allows a macro to call on other macros not yet available without aborting its own expansion.

APPENDIX D

SOURCE PROGRAM SYMBOL LIMITS

◆ The maximum size of a symbol appearing in the Name field of an assembly statement is eight characters. The maximum number of symbols which can be processed is a function of the total amount of member available to the assembly. Since a fixed amount of memory is required for the operating system components, the macro dictionary, encoded macro definitions, and certain miscellaneous tables, the memory available for symbol table usage varies widely.

If the symbol table capacity is exceeded, a Z flag is generated on the listing opposite the symbol that caused the overflow. All subsequent symbols from that point will be undefined. In this case, two alternatives exist:

1. Rewrite the program to reduce the number of symbols.

2. Independently assemble various control sections of the program and then combine into a single program by use of the Linkage Editor.

Presented below are the respective symbol limits for each operating system under which the assembly runs.

70/25 SYMBOL LIMITS

◆ The maximum number of symbols permitted in the 70/25 Processor is as follows:

70/25C (16K) - 1,024 Symbols

70/25D (32K) - 2,048 Symbols

70/25E (65K) - 4,096 Symbols (upper limit)

POS, TOS, AND TDOS SYMBOL LIMITS

◆ The number of symbols, N, permitted in the POS, TOS, and TDOS Assemblers is determined by the following formula:

$$N = \frac{X - S}{8} \text{ or } 4{,}080 \text{ whichever is smaller.}$$

where: X is number of bytes available to the Assembly System.

S = 9,000 for POS and 8,000 for TOS/TDOS.

Examples

POS

◆ For a 32K processor, assume the Supervisor requires 5,768 bytes. The calculation is as follows:

$$N = \frac{(32{,}768 - 5{,}768) - 9{,}000}{8}$$

$$N = \frac{18{,}000}{8} = 2{,}250 \text{ symbols}$$

TOS and TDOS

◆ The assembly is assigned a minimum of 32,000 bytes when running under MONITOR. Thus, the maximum number of symbols that can be processed with only 32K available is as follows:

$$N = \frac{32,000 - 8,000}{8} = 3,000 \text{ symbols.}$$

Note, that because of the multiprogramming capability, other concurrently-operating programs may occupy the remainder of available memory. Thus, in order to process the upper limit of 4,080 symbols, the assembly would require availability of 40,768 bytes. The calculation is as follows:

$$4,096 = \frac{X - 8,000}{8}$$

$$32,768 = X - 8,000$$

$$X = 32,768 + 8,000 = 40,768 \text{ bytes.}$$

Since 4,096 exceeds the limit of 4,080 the lower number is used as the limit.

SYMBOL OVERFLOW (EXCEPT 70/25)

◆ Because memory is the primary means of storage for the symbol table, encoded macro definitions and the macro dictionary, the first pass of the Assembly System may not be able to process the maximum number of symbols described above. A certain amount of memory must be reserved; for example, to store the macro dictionary. In POS it is 1,000 bytes and in TOS/TDOS it is 2,000 bytes. Thus, if a program has n source statements and if, after processing X statements, the Pass I symbol table limit has been exceeded, then Pass IA will be invoked to process the remaining N-X source statements.

The number of symbols (M) allowed in the first pass prior to overflow is 7/8 M where M is computed as follows:

<u>POS</u>

$$M = \frac{X - 24,000}{6} \text{ or } 2,048\text{, whichever is smaller.}$$

where: X = amount of memory available to the assembly. (that is, processor size less supervisor memory).

<u>TOS</u>

$$M = \frac{X - S}{6} \text{ or } 2,048\text{, whichever is smaller.}$$

where: X = amount of memory available to the assembly.

S = 26,000 if source language correction option is not used or 29,000 if it is used.

SYMBOL OVERFLOW (EXCEPT 70/25) (Cont'd)

As the above formulas imply, more than 2,048 symbols could be processed in the first pass on larger processors. In both the POS and TOS assembly systems, the first pass symbol limitation is controlled by the value assigned to symbolic location SYMBOL. This field is defined in the first pass of POS and the root segment of TOS and is preset to 2,048. If it is determined that the average number of symbols per program of a given installation will not approach this first pass limit, then the location SYMBOL could be changed to more accurately reflect actual requirements. An additional advantage to be gained is that more memory is then available for macro encoding and storage. Minimal built-in macro storage is 1,000 bytes in POS and 2,000 bytes in TOS. This minimum area tends to insure that the entire macro dictionary can be contained in memory.

Example

◆ Assume the TOS Assembly system (without source language correction facility) has 44K of memory available.

$$M = \frac{44{,}000 - 26{,}000}{6} = \frac{18{,}000}{6} = 3{,}000 \text{ symbols, before overflow.}$$

However, only 2,048 symbols will be processed before Pass IA is initiated. As noted above, this limitation on M allows additional memory to be used for macro storage. Thus, memory allocation is as follows:

Assembly	26,000	
Symbol table	12,288	(2,048 X 6)
Macro storage	5,712	
	44,000	

Note:

The minimum macro storage area of 2,000 bytes is included in the assembly allocation above, thus the actual macro storage is 7,712 bytes.

If it is determined that a lower Pass I is adequate for an installation's programs, the symbolic SYMBOL can be changed to reflect this lower limit. This has the effect of allowing more memory for macro encoding and storage. Assume 1,000 symbols is new limit (that is, SYMBOL changed to 1,000). Memory allocation then becomes:

Assembly	26,000
Symbol table	6,000
Macro storage	12,000
	44,000

APPENDIX E

70/35-45-55 MACHINE INSTRUCTIONS

LEGEND: (TABLES E-1 AND E-2)

L = Field length in bytes (1-256)

L1 = Length of first Operand field (1-16)

L2 = Length of second Operand field (1-16)

D1 = Displacement value first Operand (0-4095)

D2 = Displacement value second Operand (0-4095)

B1 = Base (general) register number first Operand (0-15)

B2 = Base (general) register number second Operand (0-15)

R1 = General register or floating-point register number

R2 = General register or floating-point register number

R3 = General register or floating-point register number

General Registers (0-15)
Floating-point registers (0,2,4,6)

I2 = Immediate date value (0-255)

*X2 = Index register number (0-15)

S1 = Absolute or relocatable expression

S2 = Absolute or relocatable expression

---

*If B2 is coded explicitly in an RX instruction, X2 must be specified. If indexing is not desired, X2 is written as a zero (0).

Table E-1. 70/35-45-55 Instruction Formats

<table>
<tr><th>Applicable Instruction</th><th>Machine Format</th></tr>
<tr><td>AP, CP, DP, MP, MVO, PACK, SP, UNPK, ZAP.</td><td>
<table>
<tr><td>8</td><td>4</td><td>4</td><td>4</td><td>12</td><td>4</td><td>12</td></tr>
<tr><td>OP</td><td>L1</td><td>L2</td><td>B1</td><td>D1</td><td>B2</td><td>D2</td></tr>
</table>
SS FORMAT (1)</td></tr>
<tr><td>CLC, ED, EDMK, LSP, MVC, MVN, MVZ, NC, OC, SSP, TR, TRT, XC.</td><td>
<table>
<tr><td>8</td><td>8</td><td>4</td><td>12</td><td>4</td><td>12</td></tr>
<tr><td>OP</td><td>L</td><td>B1</td><td>D1</td><td>B2</td><td>D2</td></tr>
</table>
SS FORMAT (2)</td></tr>
<tr><td>CKC, CLI, DIG, HDV, IDL, MVI, NI, OI, PC, RDD, SDV, TDV, TM, WRD, XI.</td><td>
<table>
<tr><td>8</td><td>8</td><td>4</td><td>12</td></tr>
<tr><td>OP</td><td>I2</td><td>B1</td><td>D1</td></tr>
</table>
SI FORMAT</td></tr>
<tr><td>LM, SLA, SLDA, SLDL, SLL, SRA, SRDA, SRDL, SRL, STM, BXH, BXLE.</td><td>
<table>
<tr><td>8</td><td>4</td><td>4</td><td>4</td><td>12</td></tr>
<tr><td>OP</td><td>R1</td><td>R3</td><td>B2</td><td>D2</td></tr>
</table>
RS FORMAT</td></tr>
<tr><td>A, AD, AE, AH, AL, AU, AW, BAL, BC, BCT, C, CD, CE, CH, CL, CVB, CVD, D, DD, DE, EX, IC, L, LA, LD, LE, LH, M, MD, ME, MH, N, O, S, SD, SE, SH, SL, ST, STC, STD, STE, STH, SU, SW, X.</td><td>
<table>
<tr><td>8</td><td>4</td><td>4</td><td>4</td><td>12</td></tr>
<tr><td>OP</td><td>R1</td><td>X2</td><td>B2</td><td>D2</td></tr>
</table>
RX FORMAT</td></tr>
<tr><td>ADR, AER, ALR, AR, AUR, AWR, BALR, BCR, BCTR, CDR, CER, CLR, CR, DDR, DER, DR, HDR, HER, ISK, LCDR, LCER, LCR, LDR, LER, LNDR, LNER, LPDR, LPER, LPR, LR, LTDR, LTER, LTR, MDR, MER, MR, NR, OR, SDR, SER, SLR, SPM, SR, SSK, SUR, SVC, SWR, XR, LNR.</td><td>
<table>
<tr><td>8</td><td>4</td><td>4</td></tr>
<tr><td>OP</td><td>R1</td><td>R2</td></tr>
</table>
RR FORMAT</td></tr>
</table>

Note:

Variations in the above instruction types are reflected in the assembly operand format (see table E-2). The fields not written in the symbolic operand will be assembled as binary zeros.

Table E-2. 70/35-45-55 Instructions

MNEMONIC	INSTRUCTION NAME	PROCESSOR MANUAL PAGE	MACH CODE	FORMAT TYPE
A	ADD WORD	118	5A	RX
AD	ADD NORMALIZED LONG	193	6A	RX
ADR	ADD NORMALIZED LONG	193	2A	RR
AE	ADD NORMALIZED SHORT	193	7A	RX
AER	ADD NORMALIZED SHORT	193	3A	RR
AH	ADD HALFWORD	119	4A	RX
AL	ADD LOGICAL	120	5E	RX
ALR	ADD LOGICAL	120	1E	RR
AP	ADD DECIMAL	142	FA	SS1
AR	ADD WORD	118	1A	RR
AU	ADD UNNORMALIZED SHT	195	7E	RX
AUR	ADD UNNORMALIZED SHT	195	3E	RR
AW	ADD UNNORMALIZED LNG	195	6E	RX
AWR	ADD UNNORMALIZED LNG	195	2E	RR
B	BRANCH UNCONDITIONAL			EX1
BAL	BRANCH AND LINK	179	45	RX
BALR	BRANCH AND LINK	179	05	RR
BC	BRANCH ON CONDITION	178	47	RX
BCR	BRANCH ON CONDITION	178	07	RR
BCT	BRANCH ON COUNT	180	46	RX
BCTR	BRANCH ON COUNT	180	06	RR
BE	BRANCH ON EQUAL			EX1
BH	BRANCH ON HIGH			EX1
BL	BRANCH ON LOW			EX1
BM	BRANCH ON MINUS			EX1
BNE	BRANCH ON NOT EQUAL			EX1
BNH	BRANCH ON NOT HIGH			EX1
BNL	BRANCH ON NOT LOW			EX1
BO	BRANCH ON OVERFLOW			EX1
BP	BRANCH ON PLUS			EX1
BR	BRANCH UNCONDITIONAL			EX2
BXH	BRANCH ON INDEX HIGH	181	86	RS
BXLE	BRANCH ON INDEX LOW OR EQUAL	182	87	RS
BZ	BRANCH ON ZERO			EX1
C	COMPARE WORD	124	59	RX
CD	COMPARE LONG	198	69	RX
CDR	COMPARE LONG	198	29	RR
CE	COMPARE SHORT	198	79	RX
CER	COMPARE SHORT	198	39	RR
CH	COMPARE HALFWORD	125	49	RX
CKC	CHECK CHANNEL PRIVIL	99	9F	SI
CL	COMPARE LOGICAL	157	55	RX
CLC	COMPARE LOGICAL	157	DK	SS2
CLI	COMPARE LOGICAL	157	95	SI
CLR	COMPARE LOGICAL	157	15	RR
CP	COMPARE DECIMAL	145	F9	SS1
CR	COMPARE WORD	124	19	RR
CVB	CONVERT TO BINARY	129	4F	RX
CVD	CONVERT TO DECIMAL	130	4E	RX
D	DIVIDE	128	5D	RX

Table E-2. 70/35-45-55 Instructions *(Cont'd)*

MNEMONIC	INSTRUCTION NAME		PROCESSOR MANUAL PAGE	MACH CODE	FORMAT TYPE
DD	DIVIDE LONG		202	6D	RX
DDR	DIVIDE LONG		202	2D	RR
DE	DIVIDE SHORT		202	7D	RX
DER	DIVIDE SHORT		202	3D	RR
DIG	DIAGNOSE	PRIVIL	91	83	SI
DP	DIVIDE DECIMAL		147	FD	SS1
DR	DIVIDE		128	1D	RR
ED	EDIT		167	DE	SS2
EDMK	EDIT AND MARK		170	DF	SS2
EX	EXECUTE		183	44	RX
HDR	HALVE LONG		199	24	RR
HDV	HALT DEVICE	PRIVIL	95	9E	SI
HER	HALVE SHORT		199	34	RR
IC	INSERT CHARACTER		162	43	RX
IDL	IDL	PRIVIL	90	80	SI
ISK	INSERT STORAGE KEY	PRIVIL	100	09	RR
L	LOAD WORD		111	58	RX
LA	LOAD ADDRESS		164	41	RX
LCDR	LOAD COMPLEMENT LONG		190	23	RR
LCER	LOAD COMPLEMENT SHORT		190	33	RR
LCR	LOAD COMPLEMENT		114	13	RR
LD	LOAD LONG		188	68	RX
LDR	LOAD LONG		188	28	RR
LE	LOAD SHORT		188	78	RX
LER	LOAD SHORT		188	38	RR
LH	LOAD HALFWORD		112	48	RX
LM	LOAD MULTIPLE		117	98	RS
LNDR	LOAD NEGATIVE LONG		192	21	RR
LNER	LOAD NEGATIVE SHORT		192	31	RR
LNR	LOAD NEGATIVE		116	11	RR
LPDR	LOAD POSITIVE LONG		191	20	RR
LPER	LOAD POSITIVE SHORT		191	30	RR
LPR	LOAD POSITIVE		115	10	RR
LR	LOAD WORD		111	18	RR
LSP	LOAD SCRATCH PAD	PRIVIL	86	D8	SS2
LTDR	LOAD AND TEST LONG		189	22	RR
LTER	LOAD AND TEST SHORT		189	32	RR
LTR	LOAD AND TEXT		113	12	RR
M	MULTIPLY WORD		126	5C	RX
MD	MULTIPLY LONG		201	6C	RX
MDR	MULTIPLY LONG		201	2C	RR
ME	MULTIPLY SHORT		201	7C	RX
MER	MULTIPLY SHORT		201	3C	RR
MH	MULTIPLY HALFWORD		127	4C	RX
MP	MULTIPLY DECIMAL		146	FC	SS1
MR	MULTIPLY WORD		126	1C	RR
MVC	MOVE		154	D2	SS2
MVI	MOVE		154	92	SI
MVN	MOVE NUMERICS		155	D1	SS2
MVO	MOVE WITH OFFSET		150	F1	SS1
MVZ	MOVE ZONES		156	D3	SS2

Table E-2. 70/35-45-55 Instructions *(Cont'd)*

MNEM ONIC	INSTRUCTION NAME		PROCESSOR MANUAL PAGE	MACH CODE	FORMAT TYPE
N	AND		158	54	RX
NC	AND		158	D4	SS2
NI	AND		158	94	SI
NOP	NO OPERATION				EX1
NOPR	NO OPERATION				EX2
NR	AND		158	14	RR
O	OR		159	56	RX
OC	OR		159	96	SI
OI	OR		159	D6	SS2
OR	OR		159	16	RR
PACK	PACK		148	F2	SS1
PC	PROGRAM CONTROL	PRIVIL	88	82	SI
RDD	READ DIRECT	PRIVIL	103	85	SI
S	SUBTRACT WORD		121	5B	RX
SD	SUBTRACT NORMALIZED SHORT		196	6B	RX
SDR	SUBTRACT NORMALIZED LONG		196	2B	RR
SDV	START DEVICE	PRIVIL	92	9C	SI
SE	SUBTRACT NORMALIZED SHORT		196	7B	RX
SER	SUBTRACT NORMALIZED SHORT		196	3B	RR
SH	SUBTRACT HALFWORD		122	4B	RX
SL	SUBTRACT LOGICAL		123	5F	RX
SLA	SHIFT LEFT SINGLE		134	8B	RS
SLDA	SHIFT LEFT DOUBLE		136	8F	RS
SLDL	SHIFT LEFT DOUBLE LOGICAL		174	8D	RS
SLL	SHIFT LEFT SINGLE LOGICAL		172	89	RS
SLR	SUBTRACT LOGICAL		123	1F	RR
SP	SUBTRACT DECIMAL		143	FB	SS1
SPM	SET PROGRAM MASK		106	04	RR
SR	SUBTRACT WORD		121	1B	RR
****	ASSEMBLY FORMATS				***
	R1,R2				RR
	R1,R3,D2(B2)				RS
	R1,D2(X2,B2)				RX
	D1(B1),I2				SI
	D1(L1,B1),D2(L2,B2)				SS1
	D1(L,B1),D2(B2)				SS2
	D2(X2,B2)				EX1
	R2				EX2

Table E-2. 70/35-45-55 Instructions *(Cont'd)*

MNEMONIC	INSTRUCTION NAME		PROCESSOR MANUAL PAGE	MACH CODE	FORMAT TYPE
SRA	SHIFT SINGLE RIGHT		135	8A	RS
SRDA	SHIFT RIGHT DOUBLE		137	8E	RS
SRDL	SHIFT RIGHT DOUBLE LOGICAL		175	8C	RS
SRL	SHIFT RIGHT SINGLE LOGICAL		173	88	RS
SSK	SET STORAGE KEY	PRIVIL	101	08	RR
SSP	STORE SCRATCH PAD	PRIVIL	87	D0	SS2
ST	STORE WORD		131	50	RX
STC	STORE CHARACTER		163	42	RX
STD	STORE LONG		200	60	RX
STE	STORE SHORT		200	70	RX
STH	STORE HALFWORD		132	40	RX
STM	STORE MULTIPLE		133	90	RS
SUR	SUBTRACT UNNORMALIZED SHORT		197	3F	RR
SVC	SUPERVISOR CALL		105	0A	RR
SW	SUBTRACT UNNORMALIZED LONG		197	6F	RX
SWR	SUBTRACT UNNORMALIZED LONG		197	2F	RR
SU	SUBTRACT UNNORMALIZED SHORT		197	7F	RX
TDV	TEST DEVICE	PRIVIL	97	9D	SI
TM	TEST UNDER MASK		161	91	SI
TR	TRANSLATE		165	DC	SS2
TRT	TRANSLATE AND TEST		166	DD	SS2
UNPK	UNPACK		149	F3	SS1
WRD	WRITE DIRECT	PRIVIL	102	84	SI
X	EXCLUSIVE OR		160	57	RX
XC	EXCLUSIVE OR		160	D7	SS2
XI	EXCLUSIVE OR		160	97	SI
XR	EXCLUSIVE OR		160	17	RR
ZAP	ZERO AND ADD DECIMAL		144	F8	SS1

APPENDIX F

SUMMARY OF 70/25 EXCEPTIONS

◆ The 70/25 Assembler is a subset of the POS/TOS/TDOS Assembly System language. The following alphabetically arranged list delineates the exceptions or restrictions of the 70/25 Assembler from the other Spectra Assemblers.

Address Constant	Maximum value of the calculated expression of an A-type constant is 2^{24}-1 on the 70/25.
AOPTN	(See page 5-6.) AOPTN functions of POS are applicable to 70/25.
CNOP	Not available on 70/25.
COM	Not available on 70/25.
DC	F-, H-, E-, D-type constants and related Scale and Exponent modifiers are not available on 70/25.
DS	F-, H-, E-, D-type operand entries are permitted to obtain appropriate boundary alignments.
Explicit Format	70/35-45-55 Compatibility can be maintained by specifying D2 (0, B2).
Extended Mnemonics	BR, NOPR not permitted. All others are acceptable.
Equipment Requirements:	70/25 Assembly System

Processor (one, 16K bytes)

Magnetic tapes (three work tapes with reverse read; one 9-channel system tape)

Input - card reader or magnetic tape

Listing - card punch or magnetic tape

Output - card punch or magnetic tape

Listing device - printer or magnetic tape

Literals	A duplicate literal is not generated for an address constant that contains a reference to the Location Counter.
Location Counter	Maximum value is 2^{19}-1 on the 70/25.
Macro Call	An inner macro call may contain up to 112 characters in the operand field on the 70/25.

SUMMARY OF 70/25 EXCEPTIONS (Cont'd)

Macro Format	The format of the macro definition can be altered by the ICTL instruction, if included in the calling programs source deck.
Macro Model Line	A model line may be continued on as many lines as necessary.
MCALL	Not available on 70/25.
MPRTY	Not available on 70/25.
MTRAC	Not available on 70/25.
NTRAC	Not available on 70/25.
Operand Field	The Operand Field may not extend through the "END" column. A blank "END" column must terminate the operand.
Stacked Assembly	Not permitted with a 16K assembly when SYS000 (worktape) and SYSOPT (generated output tape) are assigned to the same tape device.
Symbol Limits	1,024, 2,048, or 4,096 symbols are permitted with a 16K, 32K, or 65K assembly respectively.
XFR	See page 5-14. The XFR function of POS is applicable to 70/25. See also POS overlay methods, Appendix H.

APPENDIX G

SOURCE LANGUAGE MAINTENANCE

INTRODUCTION

◆ Source language maintenance is an extension of the TOS and TDOS Assemblers that provides the programmer with the capability to store and maintain Assembler language source programs on magnetic tape.

Depending on the options chosen, source language maintenance requires one or two additional tapes, which cannot be the devices assigned to the Assembly System (SYSUT1-3).

Additional maintenance facilities for programs stored on magnetic tape are provided by the Source Library Update. This utility routine is discussed in the Spectra 70 TOS Utility Routine manual, 35-302.

SOURCE LIBRARY TAPE

◆ The source library tape may contain a single program or multiple programs in any order, but is confined to a single reel. Each program consists of a number of blocks containing five 80-column source statement images, preceded by an 80-column *STARTC image and followed by a tape mark. The last program on the tape is followed by a double tape mark. Labels are not required on the input but if they are present they must be in standard format. Standard labels are written to the output tape.

To permit stacking of source coding for multiple assemblies neither tape is rewound unless rewinding is called for by a *STARTC Control Statement. Whenever output is written on magnetic tape, the Assembler writes two tape marks and backspaces one tape mark in anticipation of multiple assemblies.

Whenever a source statement is replaced the new statement and the first 38 bytes of the old statement are listed on SYSLST immediately preceding the External Symbol Dictionary of the assembly listing.

Maintenance Control Statement

◆ When source language maintenance is desired, a control statement, of the format below, must be the first statement read from SYSIPT. This statement cannot be continued.

Format

◆ OPERATION OPERAND

OPERATION	OPERAND
*STARTC	Progname, [option], [SEQ], [number], [size], [ID]

Operation Field

◆ Columns 1-7 must contain *STARTC.

Operand Field

◆ All operands except Progname are optional. A comma must be used to denote a missing operand unless no operands follow.

The program named in the *STARTC card is always assembled, and the *STARTC card from SYSIPT always replaces the *STARTC card image on the output tape. The program ID may be updated by placing a version number after the last optional operand.

Progname

◆ The program name must be preceded by at least one space and can be any combination of characters except space and comma. Maximum length is eight characters.

After the program is on magnetic tape the program name may be changed by placing the new name in cols. 73-80 of the *STARTC card.

Option

◆ This operand may be unspecified or one of the five options listed below can be chosen.

1. Unspecified: The source program, which must be in SYSIPT, is assembled and written to the output tape. This option is used for initial creation of the source library tape.

2. ADD: All programs on the input tape are copied to the output tape, then the program to be added, which must be in SYSIPT, is assembled and appended to the output tape. Correction may not be applied. If more than one program is to be added, the succeeding programs must use option 1.

3. ASSEMBLE: The choice of this option causes the specified program on the input tape to be assembled with no corrections. An output tape is not produced and the *ENDC card must be omitted.

4. CORRECT: The source program from the input tape is updated with corrections from SYSIPT and assembled. An output tape is not produced.

5. COPY: The source program from the input tape is updated with corrections from SYSIPT, assembled, and written to the output tape. No other programs from the input tape are processed.

6. COPYALL: This function is identical to COPY, except that all programs on the source library input prior to the one to be assembled/corrected are first copied to the source output.

SEQ

◆ This operand is optional. If present, it instructs the assembly to insert sequence numbers in the updated source program. If this operand is blank, the contents of columns 73-80 of the source cards, or correction cards, are retained.

Number

◆ This operand is optional and should be used only in conjunction with the SEQ operand, above. If SEQ is not used, the number operand is ignored when present.

This operand specifies the first sequence number to be assigned; if the field is omitted, zeros are assumed. In any case, sequence numbers are incremented by 100 for each statement.

Size

◆ This optional field specifies the size of the sequence number and must be from four to eight digits in length. For example, if 4 is specified, the sequence number is placed in columns 77-80. If the field is not specified, an eight-character field (that is, columns 73-80) is assumed.

If the number of digits specified in the number field exceeds that specified by the size field or the implied size field, the rightmost digits of the number field are used.

Identification

◆ This operand is ignored if the SEQ operand is omitted. This operand specifies an identification field that will be reproduced into all source statements beginning in column 73. If SEQ is used, the ID field length is the difference between the maximum (8) less the number of characters specified in the SIZE operand.

Tape Positioning

◆ Columns 71 and 72 of the *STARTC message may be used to control positioning of the source input and source output tapes. The acceptable characters and their effect on the input and/or output tapes are summarized below.

Source Input

◆ If column 71 specifies repositioning of source input tape, the tape will be rewound and positioned following the first tape mark if the tape contains a Volume and Header label. If no labels are present, it will be positioned at BOT. If no repositioning is specified, the tape will not be rewound.

It should be noted that initial creation of a source library tape, using the "unspecified" option of the *STARTC card, will include a dummy Volume (VOL) and Header (HDR) label.

Source Output

◆ If column 72 specifies repositioning of the source output tape (SOURCE), the tape will be rewound and positioned according to the following rules:

1. If a Volume (VOL) label is not found as the first record on tape, a dummy label set (VOL, HDR, TM) will be written out.
2. If a Volume label is found, a search is made for a HDR label. The expiration date is checked and if found to be purgable, a dummy Header label and TM will be written out. If the purge check fails, the operator has the choice to continue or to mount a new tape and retry.

Source Output *(Cont'd)*

The purge control characters and their meanings are as follows:

COLUMN	CHARACTER	MEANING	SIGNIFICANCE
71	Y	Position source input tape regardless of whether or not it has been positioned.	Because source correction only searches tapes forward, this permits assembly of a program previously read from the tape.
71	N	Do not position the source input tape; even if it has never been positioned.	This provides a convenient method of utilizing multiple inputs (switched about by ASSGN cards).
71	Other than Y or N	Position source input if not yet positioned.	This is the standard TOS mode of operation.
72	N	Do not position the source output tape - even if it has never been positioned.	This provides a convenient way to switch output units and control positioning.
72	Other than N	Position if not yet positioned. Do not purge if already positioned.	This is the standard TOS mode of operation.

Correction Statements

◆ Correction statements are identified by exception; that is, if a statement does not begin with *STARTC, *DELETE, OR *ENDC it is processed as a correction. Correction statements must be in SYSIPT in ascending order by sequence number (columns 73-80).

Correction statements fall into two categories: replacement and insertion. If the sequence field of a correction statement is equal to the sequence number of a source library statement, then the source library statement is replaced by the correction statement. If a correction statement has a sequence number that is not equal to the sequence number of any statement on the tape, then the statement is inserted in proper numerical order. If a correction statement has a blank in column 80 it is considered to be an insertion and is inserted immediately. Thus, by utilizing dummy replacements or insertions to position the input tape, large sections of new coding may be inserted.

Delete Statement

♦ Whenever deletion of one or more cards is desired, a delete statement of the format shown below is required.

Format

♦

OPERATION	OPERAND
*DELETE [,]	d_1 [$,d_2$]

Operation Field

♦ *DELETE is punched in columns 1-7 to identify the delete statement.

An optional comma may appear in column 8.

Operand Field

♦ d_1 specifies the sequence number of the first card to be deleted and begins in column 9.

d_2 specifies the sequence number of the last card to be deleted and must be equal to or greater than d_1.

d_1 and d_2 are any combination of characters except blank or comma. If the field is greater than eight characters, the rightmost eight characters are used. If the field is less than eight characters, the sequence field is right-justified, and space-filled to the left. If d_2 is omitted, then it is set equal to d_1.

The comma in column 8 is optional. If present, the next eight characters regardless of value, are considered as the d_1 operand. This option allows correction of individual statements that contain invalid characters in the sequence number field. In order to properly position the source tape, a "dummy" correction should be given to the last preceding statement containing a valid sequence number.

End Statement

♦ The final statement for all programs being corrected must be the *ENDC statement unless the ASSEMBLE option is used. If corrections are present this statement must follow the last correction statement, however if no corrections are present, it must follow the *STARTC statement.

Format

♦

OPERATION	OPERAND
*ENDC	[COPY]

Operation Field

♦ *ENDC is punched in columns 1-5.

Operand Field

♦ The COPY operand is optional. If present, it directs the assembly to copy the remaining programs on the source library input to the updated source output. This copy option is allowed even if the program being assembled was on SYSIPT (that is, the *STARTC operand was ADD or blank).

If the *STARTC card specifies the CORRECT option, and the COPY operand is present in *ENDC, the COPY is ignored.

Error Messages

Message	Meaning	Action
ERROR	1. d_1 greater than d_2 in *DELETE card.	d_2 is set equal to d_1.
	2. Option operand in *STARTC card invalid.	Blank operand. Assumed-rest of card ignored.
*ERROR*FATAL	Program to be corrected cannot be located on source library input.	If COPY ALL is used in *STARTC the source input is copied to source output.
NO *ENDC CARD READ	1. No *ENDC card to terminate deck. 2. Correction cards out of sequence.	Correct and restart.

Example

◆
```
// STARTM
// ASSGN  SYSLST,L1
// ASSGN  SYSUT1,01
// ASSGN  SYSUT2,02
// ASSGN  SYSUT3,03
// ASSGN  SYSLIB,04
// JOB
// PARAM  INPUT=SYSUT6
// ASSGN  SYSUT5,05
// ASSGN  SYSUT6,06
// ASSMBL
*STARTC  PROG1,ADD,SEQ,01000,5,PG1
MAIN   START
       BALR  2,0
       USING *,2
         .
         .
         .
       END
*ENDC
```

APPENDIX H

OVERLAY (SEGMENTATION) METHODS

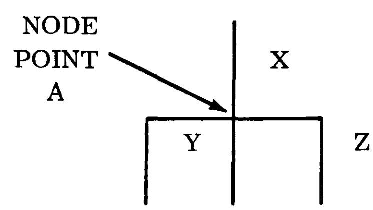
◆ Many programmers find themselves faced with the situation where available memory is not adequate for the entire program. They then must make the choice of reducing program size or developing a scheme where memory is overlaid when a particular segment of coding is needed.

This section describes the various methods of program overlays (segmentation) available to the POS and TOS/TDOS programmer. The design philosophy of the POS and TOS/TDOS systems requires that overlay planning be considered during program design and coding. The methods described herein are written at the source language level. The generated coding produced from these methods permits production of loadable object modules in the desired segmentation format.

Each logical coding entity is known as a segment. Reference to data and transfer of control between the modules within segments are accomplished by use of external referencing techniques. (See EXTRN and ENTRY, pages 4-16 and 4-15.)

A given program may contain multiple segments.

Overlay points within a program are known as "node" points. In the following diagram we show a single node point structure. Segment X is the root segment. Segments Y and Z share a common overlay point (node A). The housekeeping coding in X starting at node point A may be overlayed by either segment Y or Z.



Because there are differences in the POS and TOS/TDOS systems implementation, a separate discussion of overlay methods is presented for each system.

POS OVERLAY METHODS

◆ In POS a single assembly may create overlays. All overlays except the last must end with an XFR card. The last overlay ends with the normal END card.

To guarantee that an overlay is loaded in the desired address an ORG statement should be the first line of coding in the overlay.

POS OVERLAY METHODS (Cont'd)

Overlays are called into memory by use of the FETCH or LOAD macros (see POS FCP and Supervisor Communication Manual, No. 70-00-605).

Multiple assemblies may be combined into a program through use of the Linkage Editor.

An example of an overlay follows. This example is oriented toward the 70/25, however, the usage is applicable to POS also.

On page H-8, the DTFEN on line 0456 is coded with an OVLAY parameter. This parameter is applicable to the 70/25 only and causes the Open routine to be coded in line so that the Open may be overlaid with problem coding.

At object time the Loader reads the object cards until it encounters the XFR card. At this point (line 0490) control is transferred to the open routine at OPENRT. At the end of the open routine the FETCH macro (line 0480) is executed which overlays the open routine with the remainder of the object deck. At the end of this overlay the END (line 1870) transfers control to MAIN (0530) and the rest of the program is executed.

SAMPLE PROGRAM

Introduction

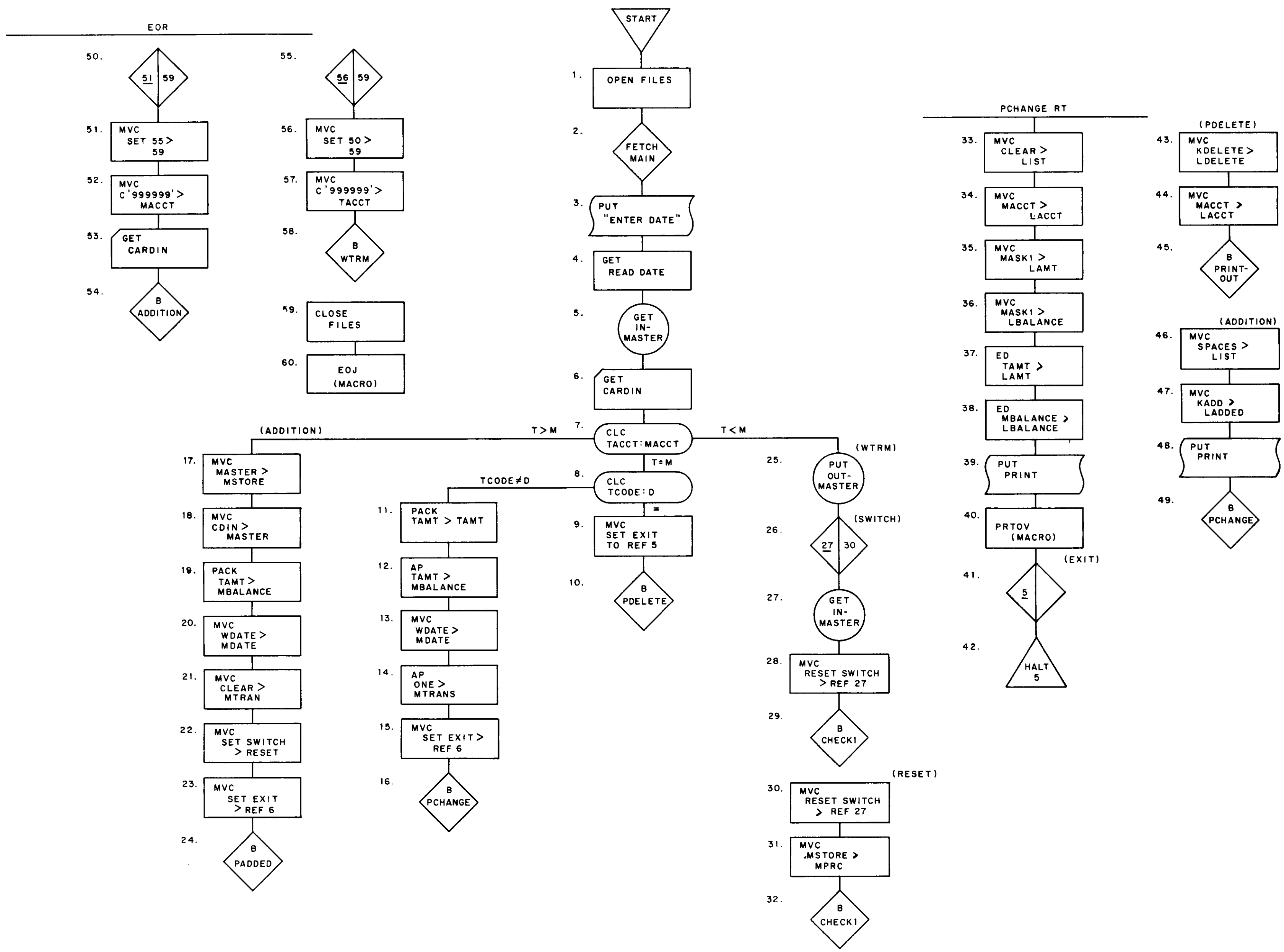
◆ This program has been included in this manual, not as an exercise in programming, but as a review of some of the assembly features previously discussed herein and contained in the related publication "POS File Control Processor and Supervisor Communication Macros Reference Manual," (No.70-00-605).

The sample program illustrates the order in which the following features might be used to solve a problem.

1. Job-Control cards used to assemble.
2. Logical and physical FCP inclusion.
3. Possible overlaying techniques.
4. Some assembly controlling codes.
5. Basic assembly formats.
6. Literals, constants and working storage.
7. Supervisor calls.

This program shows only the coding necessary for assembling a 70/25 object program.

70/25 SAMPLE PROGRAM (POS)



CHARGE NO.

DATE REQ'D

SPECTRA 70
ASSEMBLY PROGRAM FORM

FLOW CHART REFERENCE

DATE 2/68 PAGE 1 OF 12

PROGRAM Sample Program

PROGRAMMER

NAME	OPERATION	OPERAND	COMMENTS	IDENTIFICATION
//	JOB	ASSEMB		
//	LOG			
//	ASSGN	SYS000,X'001',T2		
//	ASSGN	SYS001,X'002',T2		
//	ASSGN	SYS002,X'003',T2		
//	ASSGN	SYSIPT,X'20D',R1		
//	ASSGN	SYSOPT,X'20E',P1		
//	ASSGN	SYSLST,X'30F',L1		
//	DATE	65356		
//	CONFG	011		
//	EXEC			

CHARGE NO. ______________________

DATE REQ'D ______________________

SPECTRA 70
ASSEMBLY PROGRAM FORM

FLOW CHART REFERENCE ______________

DATE 2/68 PAGE 2 OF 12

PROGRAM Sample Program

PROGRAMMER ______________

NAME	OPERATION	OPERAND / COMMENTS	72	IDENTIFICATION
BEGIN	START	10000 SAMPLE ASSEMBLY PROBLEM (70/25)		0010
INMASTER	DTFSR	ALTTAPE=SYS000,	C	0020
		BLKSIZE=50,	C	0030
		DEVADDR=SYS001,	C	0040
		DEVICE=TAPE,	C	0050
		EOFADDR=END1,	C	0060
		ERROPT=SKIP,	C	0070
		FILABL=STD,	C	0080
		IOAREA1=MAST1,	C	0090
		IOAREA2=MAST2,	C	0100
		READ=FORWARD,	C	0110
		RECFORM=FIXUNB,	C	0120
		TYPEFLE=INPUT,	C	0130
		WORKA=YES		

```
BEGIN    START 10000          SAMPLE ASSEMBLY PROBLEM (70/25)                       0010
INMASTER DTFSR ALTTAPE=SYS000,                                                C    0020
               BLKSIZE=50,                                                    C    0030
               DEVADDR=SYS001,                                                C    0040
               DEVICE=TAPE,                                                   C    0050
               EOFADDR=END1,                                                  C    0060
               ERROPT=SKIP,                                                   C    0070
               FILABL=STD,                                                    C    0080
               IOAREA1=MAST1,                                                 C    0090
               IOAREA2=MAST2,                                                 C    0100
               READ=FORWARD,                                                  C    0110
               RECFORM=FIXUNB,                                                C    0120
               TYPEFLE=INPUT,                                                 C    0130
               WORKA=YES
```

CHARGE NO.

DATE REQ'D

SPECTRA 70
ASSEMBLY PROGRAM FORM

FLOW CHART REFERENCE

DATE 2/68 PAGE 3 OF 12

PROGRAM Sample Program

PROGRAMMER

NAME	OPERATION	OPERAND / COMMENTS	72	IDENTIFICATION
OUTMASTR	DTFSR	ALTTAPE=SYS002,	C	0150
		BLKSIZE=50,	C	0160
		DEVADDR=SYS003,	C	0170
		DEVICE=TAPE,	C	0180
		FILABL=STD,	C	0210
		IOAREA1=OUT,	C	0220
		IOAREA2=OUT1,	C	0230
		RECFORM=FIXUNB,	C	0240
		TYPEFLE=OUTPUT,	C	0250
		WORKA=YES		0260

28-00-119

CHARGE NO.
DATE REQ'D

SPECTRA 70
ASSEMBLY PROGRAM FORM

FLOW CHART REFERENCE

DATE 2/68 PAGE 4 OF 12
PROGRAM Sample Program
PROGRAMMER

NAME	OPERATION	OPERAND / COMMENTS	72	IDENTIFICATION
CARDIN	DTFSR	BLKSIZE=80,	C	0270
		DEVADDR=SYSRDR,	C	0280
		DEVICE=READER,	C	0290
		ALTDEV=TAPE,	C	0300
		EOFADDR=END3,	C	0310
		IOAREA1=CARD1,	C	0320
		IOAREA2=CARD2,	C	0330
		RECFORM=FIXUNB,	C	0340
		TYPEFLE=INPUT,	C	0350
		WORKA=YES		0360

28-00-119

CHARGE NO. ______________

DATE REQ'D ______________

SPECTRA 70
ASSEMBLY PROGRAM FORM

FLOW CHART REFERENCE ______________

DATE 2/68 PAGE 5 OF 12

PROGRAM Sample Program

PROGRAMMER ______________

NAME (1–8)	OPERATION (10–14)	OPERAND / COMMENTS (16–71)	72	IDENTIFICATION (73–80)
PRINT	DTFSR	BLKSIZE=132,	C	0380
		CONTROL=YES,	C	0390
		DEVADDR=SYSLST,	C	0400
		DEVICE=PRINTER,	C	0410
		IOAREA1=LIST,	C	0420
		PRINTOV=YES,	C	0430
		RECFORM=FIXUNB,	C	0440
		TYPEFLE=OUTPUT		0450
	DTFEN	OVLAY		0465

28-00-119

CHARGE NO. ____________

DATE REQ'D ____________

SPECTRA 70
ASSEMBLY PROGRAM FORM

FLOW CHART REFERENCE ____________

DATE 2/68 PAGE 6 OF 12

PROGRAM Sample Program

PROGRAMMER ____________

NAME	OPERATION	OPERAND	COMMENTS	IDENTIFICATION
OPENRT	OPEN	INMASTER,OUTMASTR,CARDIN,PRINT		0470
	FETCH	MAIN		0480
	XFR	OPENRT		0490
	REPRO			0500
PHASE MAIN				0510
	ORG	OPENRT		0520
MAIN	BALR	3,0	LOAD GENERAL REGISTERS	0530
	USING	*,3,4,5,6,7	START MAIN PATH	0540
LOAD	LM	4,7,BASE		0550
	B	HSKP		0560
BASE	DC	A(LOAD+4096)	REGISTER CONSTANTS	0570
	DC	A(LOAD+8192)		0580
	DC	A(LOAD+12288)		0590
	DC	A(LOAD+16384)		0600
HSKP	CNTRL	PRINT,SK,1	PAGE CHANGE FORM	0610
	MVC	LIST,KSPACE	CLEAR PRINT AREA	0620
TYPEDATE	EXCP	CCBTYPER		0630
	WAIT	CCBTYPER		0640
	B	POINT1		0650
CCBTYPER	CCB	SYSLOG,CCWTYPER	DATE CONSTANT	0660
CCWTYPER	CCW	WR,KDATE,KDATE+9		0670
KDATE	DC	C'ENTER DATE'		0680
POINT1	EXCP	CCBREAD		0690
	WAIT	CCBREAD		0700

28-00-119

CHARGE NO. ____________

DATE REQ'D ____________

SPECTRA 70
ASSEMBLY PROGRAM FORM

FLOW CHART REFERENCE ____________

DATE 2/68 PAGE 7 OF 12

PROGRAM Sample Program

PROGRAMMER ____________

NAME	OPERATION	OPERAND	COMMENTS	IDENTIFICATION
	B	POINT2		0710
CCBREAD	CCB	SYSLOG,CCWREAD		0720
CCWREAD	CCW	RDF,WDATE,WDATE+5		0730
WDATE	DS	CL6		0740
POINT2	GET	INMASTER,MPRC		0750
POINT3	GET	CARDIN,CDIN		0760
CHECK1	CLC	TACCT,MACCT	COMPARE ACCOUNT CODES	0770
	BL	ADDITION	BRANCH TO ADD	0780
	BH	WRTM	BRANCH TO WRITEOUT	0790
	CLC	TCODE,=C'D'	COMPARE TRANSACTION CODE	0800
	BNE	UPDATE	BRANCH TO UPDATE LOGIC	0810
	MVC	EXIT+2(2),ADA1	SET PRINT EXIT TO POINT1	0820
	B	PDELETE	BRANCH TO PRINT DELETE	0830
UPDATE	PACK	WTAMT(5),TAMT(9)	PACK TRANSACTION AMOUNT	0840
	AP	MBALANCE,WTAMT	ADD TRAN AMT TO BALANCE	0850
	MVC	MDATE,WDATE	MOVE CURRENT DATE TO MASTER	0860
	AP	MTRANS,=X'1C'	ADD ONE TO NUMBER OF TRANS	0870
	MVC	EXIT+2(2),ADB1	SET PRINT EXIT TO POINTS	0880
	B	PCHANGE	BRANCH TO PRINT CHANGE	0890
ADB1	DC	S(POINT3)	S TYPE CONSTANTS	0900
ADC1	DC	S(CHECK1)	S TYPE CONSTANTS	0910

CHARGE NO.
DATE REQ'D

SPECTRA 70
ASSEMBLY PROGRAM FORM

FLOW CHART REFERENCE

DATE 2/68 PAGE 8 OF 12
PROGRAM Sample Program
PROGRAMMER

NAME	OPERATION	OPERAND	COMMENTS	IDENTIFICATION
ADA1	DC	S(POINT2)	CONSTANT ADDRESS	0920
ADDITION	MVC	MSTORE,MPRC	MOVE MASTER TO WORK	0930
	MVC	MPRC(30),CDIN	MOVE TRANS TO MASTER	0940
	PACK	MBALANCE,TAMT	PACK AND MOVE AMT	0950
	MVC	MDATE,WDATE	MOVE CURRENT DATE TO MAS	0960
	MVC	MTRANS,=X'000000'	MOVE ZEROS TO NO OF TRANS	0970
	MVC	SWITCH+1(1),=X'F0'	SET WRITE SWITCH	0980
	MVC	EXIT+2(2),ADB1	SET PRINT EXIT TO CHK 1	0990
	B	PADDED	BRANCH TO PRINT ADDITION	0995
WRTM	PUT	OUTMASTR,MPRC	WRITE NEW MASTER	1000
SWITCH	BC	X'07',RESET	WRITE SWITCH	1010
	GET	INMASTER,MPRC	READ MASTER	1020
	MVC	SWITCH+1(1),=X'00'	SWITCH CONSTANT	1030
	B	CHECK1		1040
RESET	MVC	SWITCH+1(1),=X'00'	RESET WRITE SWITCH	1050
	MVC	MPRC,MSTORE	MOVE STORED MASTER	1060
	B	CHECK1	BRANCH TO CHECK1	1070
PCHANGE	MVC	LIST,KSPACE		1080
	MVC	LACCT,MACCT		1090
	MVC	LAMT,MASK1		1100

28-00-119

CHARGE NO. ____________

DATE REQ'D ____________

RCA

SPECTRA 70
ASSEMBLY PROGRAM FORM

FLOW CHART REFERENCE ____________

DATE 2/68 PAGE 9 OF 12

PROGRAM Sample Program

PROGRAMMER ____________

NAME	OPERATION	OPERAND	COMMENTS	IDENTIFICATION
	MVC	LBALANCE,MASK1		1110
	ED	LAMT,WTAMT		1120
	ED	LBALANCE,MBALANCE		1130
PRINTOUT	PUT	PRINT		1140
	PRTOV	PRINT,12	SENSE FOR OVERFLOW	1150
EXIT	B	HALT5	COMMON PRINT EXIT	1160
HALT5	HB	HALT5,X'05'	INVALID EXIT LOAD	1170
* PROGRAM CONSTANTS				1180
	DS	0F		1190
KDELETE	DC	C'DELETED'		1200
KADD	DC	C'ADDED'		1210
MASK1	DC	X'40206B2020206B2020214B202060'		1220
TEST	DC	P'+1:25'		1225
*DELETE PRINT PATH				1230
PDELETE	MVC	LDELETED,KDELETE	MOVE DELETED CONSTANT	1240
	MVC	LACCT,MACCT	MOVE ACCOUNT NUMBER	1250
	B	PRINTOUT	BRANCH TO PRINT	1260
*ADDITION PRINT PATH				1270
	MVC	LIST,KSPACE	MOVE SPACES TO PRINT	1280
	MVC	LADDED,KADD	MOVE CONSTANT TO PRINT	1290
	PUT	PRINT	PRINT LINE ONE	1300
	B	PCHANGE	BRANCH TO PRINT	1310

28-00-119

CHARGE NO.

DATE REQ'D

SPECTRA 70
ASSEMBLY PROGRAM FORM

FLOW CHART REFERENCE

DATE 2/68 PAGE 10 OF 12
PROGRAM Sample Program
PROGRAMMER

```
NAME     OPERATION OPERAND / COMMENTS                                      IDENTIFICATION
* ALLOCATION OF STORAGE AREAS FOR INMASTER                                 1320
         DS     0F                                                         1330
MAST1    DS     CL50      INPUT AREA ONE FOR MASTER                        1340
MAST2    DS     CL50      INPUT AREA TWO FOR MASTER                        1350
MPRC     DS     0CL50     WORKA FOR INPUT MASTER                           1360
MACCT    DS     CL6                                                        1370
MNAME    DS     CL24                                                       1380
MBALANCE DS     CL5       PACKED FIELD                                     1390
MDATE    DS     CL6                                                        1400
MTRANS   DS     CL5                                                        1410
MSPACE   DS     CL4                                                        1420
OUT      DS     CL50      ALLOCATION OF STORAGE FOR CARD IN                1425
OUT1     DS     CL50                                                       1426
* ALLOCATION OF STORAGE AREAS FOR CARDIN
         DS     0F                                                         1430
CARD1    DS     CL80      INPUT AREA ONE FOR CDIN                          1440
CARD2    DS     CL80      INPUT AREA TWO FOR CDIN                          1450
CDIN     DS     0CL80                                                      1460
TACCT    DS     CL6                                                        1470
TNAME    DS     CL24                                                       1480
TAMT     DS     CL9                                                        1590
TCODE    DS     CL1                                                        1500

WTAMT    DS     CL5                                                        1510
```

28-00-119

CHARGE NO. ____________

DATE REQ'D ____________

SPECTRA 70
ASSEMBLY PROGRAM FORM

FLOW CHART REFERENCE ____________

DATE 2/68 PAGE 11 OF 12

PROGRAM Sample Program

PROGRAMMER ____________

NAME	OPERATION	OPERAND	COMMENTS	IDENTIFICATION

```
*  ALLOCATION OF STORAGE AREAS FOR PRINTOUT                                 1520

         DS    0F                                                           1530
KSPACE   DC    X'40'                                                        1540
LIST     DS    CL132                                                        1550
         ORG   LIST                                                         1560
LSPACE   DS    CL10                                                         1570
LACCT    DS    CL6                                                          1580
LSPACE2  DS    CL7                                                          1590
LAMT     DS    CL12                                                         1600
LSPACE3  DS    CL6                                                          1610
LBALANCE DS    CL12                                                         1620
         ORG   LIST+10                                                      1630
LADDED   DS    CL5                                                          1640
         ORG   LIST+19                                                      1650
LDELETED DS    CL7                                                          1660
         ORG   LIST+132                                                     1670

*  ALLOCATION OF MASTER STORAGE                                             1680

         DS    0F                                                           1690
MSTORE   DS    CL50                                                         1700

*  START END OF RUN ROUTINES                                                1710
```

28-00-119

CHARGE NO. ______________________

DATE REQ'D ______________________

SPECTRA 70
ASSEMBLY PROGRAM FORM

FLOW CHART REFERENCE ______________

DATE 2/68 PAGE 12 OF 12

PROGRAM Sample Program

PROGRAMMER ______________________

NAME	OPERATION	OPERAND / COMMENTS	IDENTIFICATION
PADDED	MVC	LIST,KSPACE	1720
	MVC	LADDED,KADD	1730
	PUT	PRINT	1740
	B	PCHANGE	1750
END1	BC	X'0',FINAL END OF INMASTER FILE	1760
	MVC	END3+1(1),=X'F0'	1770
	MVC	MACCT,=X'FFFFFFFFFFFFFFFF'	1780
	GET	CARDIN,CDIN	1790
	B	ADDITION	1800
END3	BC	X'00',FINAL END OF CARDIN FILE	1810
	MVC	END1+1(1),=X'F0'	1820
	MVC	TACCT,=X'FFFFFFFFFFFFFFFF'	1830
	B	WRTM	1840
FINAL	CLOSE	INMASTER,OUTMASTR,CARDIN,PRINT	1850
	EOJ		1860
	END	MAIN	1870

28-00-119

TOS/TDOS OVERLAY METHODS

◆ TOS overlays are created from separately assembled object modules by the Linkage Editor.

Overlay segments in the program can be loaded when desired by using the LPOV macro (described in the TOS/TDOS FCP Manual, No. 70-00-608) or either the CALL or SEGLD macro described below. The latter two methods require the Linkage Editor to produce an Overlay Control Module and two tables (SEGTAB and ENTAB) and bind them into the user's program. This overlay control module accesses tables that reflect the status of segments presently in memory and of the overlay structure of the program. These tables provide the facility for a single overlay macro call by the user to bring a particular segment and all segments in the same path between it and the root segment into memory. The facility is also available for an overlay call to become a branching action when the requested load is already in memory.

It is recommended that the LPOV macro not be used in a program that also uses either CALL or SEGLD. The LPOV macro interfaces directly with the TOS Executive and thus does not update the overlay status tables. Since this status information is required by the CALL macro, which loads a segment only if the segment is not already in memory, invalid results could occur.

Furthermore, if either CALL or SEGLD is used, the NOCTL parameter must not be specified to Linkage Editor

CALL
Call Segment

General Description

◆ The CALL macro is used to effect a transfer of control between segments. When CALL is issued the Linkage Editor tables are checked to determine if the requested segment is already in memory. If the segment is in memory, a branch is performed to the symbol specified. If the segment is not in memory, an overlay request is issued which causes the requested segment and any other segments in its path to be brought into memory. Then the branch is performed.

Format

◆ The format is as follows:

NAME	OPERATION	OPERAND
Symbol or Blank.	CALL	Symbol.

Specification Rules

Name

◆ Symbol or blank.

Operation

◆ CALL.

Operand

◆ Symbolic name of the entry point within the called segment to which control is transferred. This symbolic name must appear as an ENTRY in the segment to be loaded.

Register 15 is used by this macro and its previous contents will be destroyed.

SEGLD
Segment Load

General Description

◆ The SEGLD macro causes loading of the segment containing the referenced entry point (SYMBOL1) including all segments within its path. The requested segment is loaded unless it is higher in the path than the requesting segment.

Format

◆ The format is as follows:

NAME	OPERATION	OPERAND
Symbol or Blank.	SEGLD	Symbol 1, [Symbol 2]

Specification Rules

Name

◆ Symbol or blank.

Operation

◆ SEGLD.

Operand

◆ SYMBOL1 - Names an entry point within the segment to be loaded.

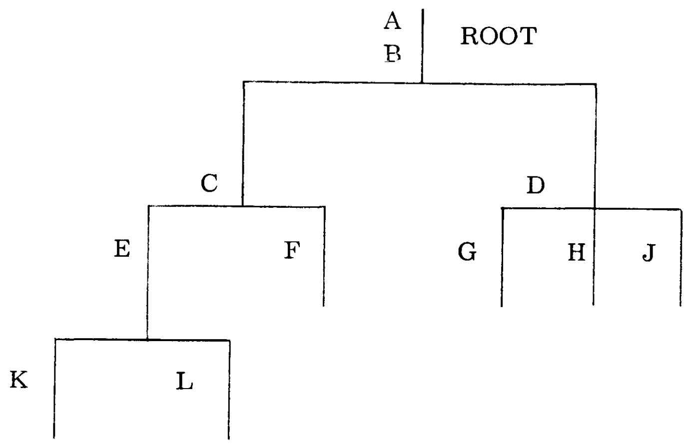
SYMBOL2 (OPTIONAL) - Specifies the instruction that is to be executed upon completion of the loading process. If no symbol specified, control goes to next sequential instruction.

1. If SYMBOL2 is an external reference, it is the user's responsibility to establish the appropriate ENTRY and EXTRN statements and to ensure that the module that contains SYMBOL2 is in memory upon completion of the loading process.

2. Register 15 is used by this macro and its previous contents will be destroyed.

PROGRAM EXAMPLE

◆ The following example is intended to represent a program structure of numerous object modules. Each object module was assembled separately and bound by the Linkage Editor into the logical structure as shown. All module to module references were made by use of CALL or SEGLD macros.



The following names were available in the indicated modules.

1. In program A the following statement caused only a branch to TESTER because it is contained in the root load.

NAME	OPERATION	OPERAND
	CALL	TESTER

2. In Program B the statement

NAME	OPERATION	OPERAND
RHOM	CALL	NET

caused segments L,E, and C to be called into memory giving the following use of memory with control transferred to NET.

Entry Points

MODULE NAMES				
A		HOME		ROOT SEGMENT
B		TESTER RHOM		ROOT SEGMENT
C				
E		TREAT		
L		NET		

PROGRAM EXAMPLE (Cont'd)

3. In program L, the following statement

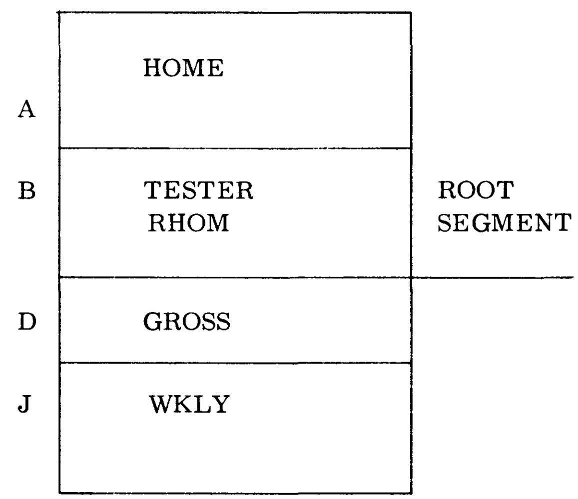
NAME	OPERATION	OPERAND
REINT	SEGLD	TREAT

caused segments E and C to be called into memory again and control transferred to the next sequential instruction.

4. In Program L, the statement

NAME	OPERATION	OPERAND
	SEGLD	WKLY, GROSS

caused segments J and D to be called into memory and control transferred to GROSS which is a tag in segment D.



APPENDIX I

MACRO LANGUAGE TERMINOLOGY

Call Line

◆ See Macro Call Statement.

Character String

◆ A sequence of character values that are combined at generation time into a final character value. (See Substring.)

Conditional Commands

◆ The conditional commands permit the programmer to control the sequence of the executed or generated statements based on values present in the macro call.

Expressions

◆ See Logical or Relational Expressions.

Global Symbols

◆ Assigned values of SETA, SETB, and SETC variable symbols which remain in effect for all references to the variable symbol throughout the assembly unless changed. SETC variable symbols must be global.

Header Statement

◆ The first statement of a macro definition. Indicates the beginning of a macro definition.

Inner Macro Call Statement

◆ Name given to a macro call that is contained within a macro definition. The type of inner macro call (that is, positional or keyword) is independent of the type of the <u>containing</u> macro definition.

Keyword Macros

◆ Values to be substituted (generated) are paired with keywords in the macro definition so that if a required value is omitted from the macro call line, the keyword associated with that value will be substituted. Parameters may appear in random order within the macro call. (See also Positional Macros.)

Local Symbols

◆ Assigned values of SETA and SETB variable symbols which remain in effect for all references to the variable symbol within the macro in which the variable symbol is defined unless changed by a SET command. SETC variable symbols <u>cannot</u> be local. After the macro is generated, the values are reset to zero or false.

Logical Expressions

◆ A series of terms connected with one or more logical operators (AND, OR, and NOT) that controls the combining of the component terms into a final value. Each expression is enclosed in parentheses, and no more than three levels of parentheses are allowed. Logical Expressions are only used with the SETB, AIF, and AIFB macro commands.

Macro Definition — ◆ The series of statements that comprise the macro. The definition consists of a Header Statement, a Prototype Statement, Model Statements, and a Trailer Statement.

Macro Expansion — ◆ The substitution of variable symbol values in the model statements during their generation in place of the macro call statement.

Macro Call Statement — ◆ The line(s) of coding that contains the parameters that are substituted within the generated model statements. Also referred to as: Call Line, Macro Call, and Macro Instruction.

Model Statements — ◆ Statements that make up the macro definition which are executed or generated. The Name, Operation, and Operand fields can contain symbols defined in the macro call or variable symbols used in the macro definition. The variable symbols are, in turn, replaced by the values they represent.

Null Parameter — ◆ A parameter that is not included in the macro call when a symbolic parameter has been included in a prototype statement.

Operand Values — ◆ See Values.

Positional Macros — ◆ One of the two types of macros (see Keyword Macros). Values to be substituted for symbolic parameters in the Prototype statement <u>must appear in a prescribed order in the macro call.</u>

Prototype Statement — ◆ Defines the format and the mnemonic operation code of the macro call. The Operand field contains symbolic parameters used during generation of model statements. This statement must appear as the second statement in the macro definition.

Relational Expressions — ◆ Consist of two terms connected by a relational operator (EQ, NE, LT, GT, LE, GE). Each expression is enclosed within parentheses, and no more than three levels of parentheses are allowed. Used only with the SETB, AIF, and AIFB macro commands.

Set Macro Commands — ◆ Allow character manipulation, arithmetic calculation, and the setting and testing of binary switches on the basis of logical and relational expressions. The Set commands are: SETA, SETC, and SETB, which assign arithmetic, character and binary values, respectively, to Set variable symbols.

Sequence Symbols — ◆ Identifies a model statement as the destination of a conditional or unconditional macro branch command (AIF, AIFB, AGO, or AGOB).

Set Variable Symbols — ◆ Symbols that are associated with the Set commands. Character, arithmetic, and binary values are assigned to them and may be altered by the programmer at any time using the Set commands.

Substring — ◆ Used in the SETC or SETA statements to obtain a portion of a value.

Symbolic Parameters

◆ Name given to the generalized parameters defined in the prototype statement. Values contained in the macro call that correspond to the prototype's symbolic parameters (either positionally or by keyword) are substituted for the identical symbolic parameters in the model statements at generation time.

System Variable Symbols

◆ Local variable symbols that are assigned values by the Assembler at generation time. They can be used in the Name field or Operand field of macro definition statements. The system variable symbols are &SYSNDX, &SYSECT, and &SYSLST.

Trailer Statement

◆ Signifies the end of a macro definition. Must be the last statement of a macro definition.

Values

◆ The character string of up to eight characters which is assigned by either a Set macro command or a macro call statement to a variable symbol. Each call value must have been represented in the prototype statement as a symbolic parameter.

Variable Symbols

◆ Symbols representing varying values, which may be assigned, changed, or tested at any time during macro generation, by the programmer and/or the assembler. Current values are examined to determine what model statements are to be generated. Variable symbols can either be: 1) symbolic parameters, 2) System variable symbols, or 3) Set variable symbols.

APPENDIX J

SUMMARY OF MACRO DEFINITION OPERATION CODES

Operation Codes	Name Field	Operand Field
AGO	A sequence symbol or blank.	A sequence symbol of a statement following the AGO.
AGOB	A sequence symbol or blank.	A sequence symbol of a statement preceding the AGOB.
AIF	A sequence symbol or blank.	A logical or relational expression enclosed within parentheses, immediately followed by a sequence symbol of a statement following the AIF.
AIFB	A sequence symbol or blank.	A logical or relational expression enclosed within parentheses, immediately followed by a sequence symbol of a statement preceding the AIFB.
ANOP	A sequence symbol.	Not used.
MACRO	Not used.	See page 7-2.
MEND	A sequence symbol or blank.	Not used.
MEXIT	A sequence symbol or blank.	Not used.
MNOTE	A sequence symbol or blank.	An optional error code followed by a combination of characters enclosed within quotation marks.
SETA	&AGn or &ALn, where n is 0 - 15.	An arithmetic expression.
SETB	&BGn, or &BLn, where n is 0 - 127.	A logical expression or a relational expression enclosed within parentheses.
SETC	&CGn, where n is 0 - 15.	Up to eight characters enclosed within a pair of single quote marks with substrings allowed. Concatenation of enclosed terms allowed to form the final eight characters.

SUMMARY OF MACRO DEFINITION OPERATION CODES (Cont'd)

Operation Codes	Name Field	Operand Field
Model Statement (any assembly mnemonic operation code, symbolic parameter, or assembly command except END, ICTL, ISEQ, START, and sequence symbols).	A symbol parameter, a symbol, a variable symbol, a sequence symbol, or a combination of variable symbols and other characters that are equivalent to a symbol.	Any combination of characters (including symbolic parameters and variable symbols).
Prototype Statement	Mnemonic operation code.	Comma(,) or a maximum of 49 symbolic parameters, separated by commas.
Macro Call Statement	A valid mnemonic operation code.	Comma(,) or a maximum of 49 operands, separated by commas.

APPENDIX K

SUMMARY OF MACRO EXPRESSIONS

Comment	Arithmetic	Character	Logical	Relational
Can Contain:	1. Positive decimal self-defining terms.	1. Up to eight characters enclosed by a pair of single quote marks.	1. 0, 1, or SETB variable symbols.	1. Two arithmetic expressions.
	2. SETA and SETB variable symbols.	2. Any SET variable symbol or previously defined symbolic parameter enclosed by a pair of single quotes.	2. NOT &BL$\underline{n}$ or NOT &BG$\underline{n}$ where n=0-127.	2. Two character expressions.
	3. SETC variable symbols if the value assigned is a positive-decimal, self-defining term.	3. A combination (concatenation) of variable symbols, symbolic parameters, and other characters enclosed by a pair of single quotes with substrings allowed to form the final 8 characters (16 intermediate characters).	3. Two or more SETB variable symbols and the associated logical operators.	
	4. Symbolic parameters if the corresponding operand is a positive decimal self-defining term.		4. 0 and 1 can be used only in single-term expressions.	
	5. &SYSLIST(n) if the corresponding operand is a positive-decimal, self-defining term.		5. Combination of logical and/or relational expressions enclosed in parentheses and nested to a maximum of three levels.	
	6. &SYSNDX.			

SUMMARY OF MACRO EXPRESSIONS (Cont'd)

Comment	Arithmetic	Character	Logical	Relational
Operators Are:	+, -, *, and/.	Concatenation with a period (.).	AND, OR, and NOT.	EQ, NE, LT, GT, LE, and GE.
Range of Values Are:	0 to 2^{24}-1.	Zero to eight characters.	0(false) or 1(true).	0(false) or 1(true).
Can Be Used In:	1. SETA operands.	1.SETC operands.	1. SETB operands.	1. SETB operands.
	2. Relational expressions.	2. Relational expressions.	2. AIF operands.	2. AIF operands.
	3. SETC operands.	3. SETA operands if the assigned value is a positive-decimal, self-defining term.	3. AIFB operands.	3. AIFB operands.

APPENDIX L

SUMMARY OF MACRO SYMBOLIC PARAMETERS AND VARIABLE SYMBOLS

Symbol	Defined By	Initialized or Set To	Value Changed By	Can Be Used
Symbolic parameter.	Prototype statement.	Corresponding macro call operand value.	Constant throughout definition.	1. Arithmetic expressions if operand is self-defining, positive-decimal term. 2. Character expressions. 3. Model statements. 4. Relational expressions.
SETA	Predefined.	0	SETA command.	1. Arithmetic expressions. 2. Character expressions. 3. Model statements. 4. Relational expressions.
SETB	Predefined.	0	SETB command.	1. Arithmetic expressions. 2. Character expressions. 3. Logical expressions. 4. Relational expressions. 5. Model statements.

SUMMARY OF MACRO SYMBOLIC PARAMETERS AND VARIABLE SYMBOLS (Cont'd)

Symbol	Defined By	Initialized or Set To	Value Changed By	Can Be Used
SETC	Predefined.	Null character value.	SETC command.	1. Arithmetic expressions if operand is self-defining positive-decimal term. 2. Character expressions. 3. Model statements. 4. Relational expressions.
&SYSNDX	The assembly.	Macro instruction index.	Constant throughout definition; different for each macro call.	1. Arithmetic expressions. 2. Character expressions. 3. Model statements. 4. Relational expressions.
&SYSECT	The assembly.	Control section in which macro call appears.	Constant throughout definition; set by CSECT, DSECT, and START.	1. Character expressions. 2. Model statements. 3. Relational expressions.
&SYSLIST(n) Where n is an arithmetic expression.	The assembly.	Corresponding macro call operand value.	Constant throughout definition for a given value of n.	1. Arithmetic expressions if operand is self-defining, positive-decimal term. 2. Character expressions. 3. Model statements. 4. Relational expressions.

APPENDIX M

HEXADECIMAL-DECIMAL CONVERSION TABLE

General

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

Hexadecimal	*Decimal*
000 to FFF	0000 to 4095

Hexadecimal-Decimal Number Conversion Table

◆ In the table, the decimal value appears at the intersection of the row representing the most significant hexadecimal digits (16^2 and 16^1) and the column representing the least significant hexadecimal digit (16^0).

Example: $C21_{16} = 3105_{10}$

HEX	*0*	*1*	*2*
C0	3072	3073	3074
C1	3088	3089	3090
C2	3104	3105	3106
C3	3120	3121	3122

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

Hexadecimal	*Decimal*	*Hexadecimal*	*Decimal*
1000	4,096	C000	49,152
2000	8,192	D000	53,248
3000	12,288	E000	57,344
4000	16,384	F000	61,440
5000	20,480	10000	65,536
6000	24,576	20000	131,072
7000	28,672	30000	196,608
8000	32,768	40000	262,144
9000	36,864	50000	327,680
A000	40,960	60000	393,216
B000	45,056	70000	458,752

Example: $1C21_{16} = 7201_{10}$

Hexadecimal	*Decimal*
C21	3105
+1000	+4096
1C21	7201

HEXADECIMAL-DECIMAL CONVERSION TABLE (Cont'd)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
01	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
02	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
03	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
04	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
05	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
06	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
07	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
08	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
09	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
10	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
11	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
12	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
13	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
14	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
15	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
16	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
17	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
18	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
19	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
20	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
21	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
22	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
23	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
24	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
25	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
26	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
27	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
28	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
29	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
30	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
31	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
32	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
33	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
34	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
35	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
36	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
37	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
38	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
39	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023

HEXADECIMAL-DECIMAL CONVERSION TABLE (Cont'd)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
40	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
41	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
42	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
43	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
44	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
45	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
46	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
47	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
48	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
49	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
50	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
51	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
52	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
53	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
54	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
55	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
56	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
57	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
58	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
59	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
60	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
61	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
62	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
63	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
64	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
65	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
66	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
67	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663
68	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
69	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
70	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
71	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
72	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
73	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
74	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
75	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
76	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
77	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
78	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
79	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047

HEXADECIMAL-DECIMAL CONVERSION TABLE (Cont'd)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
80	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
81	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
82	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
83	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
84	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
85	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
86	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
87	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
88	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
89	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
90	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
91	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
92	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
93	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
94	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
95	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
96	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
97	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
98	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
99	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A0	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A1	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A2	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A3	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A4	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A5	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A6	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A7	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A8	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A9	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC0	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
AD0	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE0	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF0	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
B0	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B1	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B2	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B3	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B4	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B5	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B6	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B7	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B8	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B9	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071

HEXADECIMAL-DECIMAL CONVERSION TABLE (Cont'd)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C0	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C1	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C2	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C3	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C4	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C5	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C6	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C7	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C8	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C9	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
D0	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D1	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D2	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D3	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D4	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D5	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D6	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D7	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D8	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D9	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
E0	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E1	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E2	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E3	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E4	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E5	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E6	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E7	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E8	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E9	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
F0	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F1	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F2	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F3	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F4	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F5	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F6	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F7	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F8	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F9	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

APPENDIX N

SAMPLE PROGRAM

INTRODUCTION

◆ This sample program is included in the manual to illustrate the TOS Monitor job stream necessary to assemble a source program and bind the output using Linkage Editor. The loadable module is then executed without the use of Monitor.

The card deck composition to accomplish this is as follows:

```
//ΔSTARTM
//ΔASSGN SYSLST,L1
//ΔASSGN SYSUT1,01
//ΔASSGN SYSUT2,02
//ΔASSGN SYSUT3,03
//ΔASSGN SYSLIB,04
//ΔJOB TOS MONITOR
//ΔPARAM XREF=YES
//ΔASSMBL
        (Optional Assembly Codes)         }
ΔSTART                                    }
        (Reader and printer DTFSR's)      }  Assembly
        (User source deck macro)          }  Source
        (Remainder of source)             }  Program
ΔEND                                      }
//ΔLNKEDT
        (Various // COMM cards - optional)
//ΔENDMON
//ΔASSGN SYS001,R1    }  Run-time parameters for
//ΔASSGN SYS002,L1    }  reader and printer not
//ΔEND                }  shown on listing.
        (Data cards)
/*
```

INTRODUCTION (Cont'd)

The following computer output from these runs are shown on the succeeding pages:

1. Listing of Monitor control statements;
2. Assembler listing (see note);
3. Linkage Editor map;
4. Sample output from program execution;
5. Console typewriter sheet.

Note

◆ Only a small portion of the cross reference listing has been included (XREF=YES). The user macro MOVE has been allowed to expand (PRINT=GEN), while all other macro expansions have been suppressed (PRINT=NOGEN).

```
TOS MONITOR

// ASSGN SYSLST,L1

// ASSGN SYSUT1,01

// ASSGN SYSUT2,02

// ASSGN SYSUT3,03

// ASSGN SYSLIB,04
```

```
// JOB  TOS MONITOR

// PARAM XREF=YES

// ASSMBL
```

EXTERNAL SYMBOL DICTIONARY

	SYMBOL	TYPE	ID	ADDR	LENGTH
	SAMPL	SD	01	00000	010C8
(DUMMY)	IDUMMY	SD	FF	00000	00198
	IN	LD	01	00000	
	INB	LD	01	0005C	
	OUT	LD	01	000C0	
	OUTB	LD	01	0011C	
	IFCP	LD	01	001B8	
	IFCPOV	LD	01	00918	

SYMBOL	CSECT	VALUE	L	SYMBOL	CSECT	VALUE	L	SYMBOL	CSECT	VALUE	L	SYMBOL	CSECT	VALUE	L
BEGIN	01	00EF8	02	CTR	01	00FEE	02	EF	01	00FD0	04	IAFINAL	FF	00084	04
IAFTER	FF	00070	02	IAFTERID	FF	00074	04	IALTTAPE	FF	00062	06	IBLKSIZE	FF	00078	04
IBUFFER	FF	000B0	01	IBUFFERI	FF	000B4	04	IBYTCNT	FF	00038	02	IB1STAT	FF	0008A	01
IB2STAT	FF	0008B	01	ICAPREC	FF	00088	01	ICARDS	FF	000C0	04	ICCB	FF	00000	28
ICCBAFI	FF	00024	04	ICCBAH	FF	0000E	02	ICCBASR	FF	0001C	04	ICCBCAR	FF	00010	04
ICCBCCR1	FF	00018	04	ICCBCCR2	FF	00014	04	ICCBCCW	FF	00008	06	ICCBDT	FF	00006	01
ICCBEXF	FF	00023	01	ICCBSB	FF	00020	03	ICCBSDN	FF	00000	06	ICCBUF	FF	00007	01
ICCW1	FF	00028	08	ICCW12	FF	00130	08	ICCW13	FF	00138	08	ICCW14	FF	00140	08
ICCW15	FF	00148	08	ICCW16	FF	00150	08	ICCW17	FF	00158	08	ICCW18	FF	00108	08
ICCW19	FF	00110	08	ICCW2	FF	00030	08	ICCW20	FF	00118	08	ICCW21	FF	00120	08
ICCW22	FF	00128	08	ICCW23	FF	00130	08	ICCW24	FF	00138	08	ICCW25	FF	00140	08
ICCW26	FF	00148	08	ICCW27	FF	00150	08	ICCW28	FF	00158	08	ICCW29	FF	00160	08
ICCW3	FF	00108	08	ICCW32	FF	00168	08	ICCW33	FF	00170	08	ICCW34	FF	00178	08
ICCW35	FF	00180	08	ICCW36	FF	00188	08	ICCW37	FF	00190	08	ICCW6	FF	00110	08
ICCW7	FF	00118	08	ICCW8	FF	00120	08	ICCW9	FF	00128	08	ICHECKPT	FF	0008B	01
ICKPTREC	FF	00088	01	ICLOSE	01	0022A	01	ICLOSE1B	01	0022A	04	ICONTROL	FF	0008C	04
ICRDERR	FF	00070	01	ICTLCHR	FF	00088	01	IDAIOA	FF	000C0	04	IDARESER	FF	00090	04
IDASTOR	FF	000D0	38	IDTFTYPE	FF	0003E	01	IDUMMY	FF	00000	01	IEOFADDR	FF	00070	01
IEOVIPT	01	002D6	04	IEOVREV	01	002E6	04	IERRBYTE	FF	000A8	04	IERROPT	FF	000C0	01
IERRORB	FF	0003B	01	IEXPAND	FF	00060	02	IFCP	01	001B8	40	IFCPOV	01	00918	08
IFEOV	01	002EE	04	IFEOVCAL	01	0032A	04	IFEOVILP	01	00344	04	IFEOVIPT	01	00338	04
IFEOVRPT	01	002F8	04	IFILABL	FF	0003A	01	IFILL	FF	00070	02	IFILSTAT	FF	0003A	01
IFIRSTIM	FF	0003F	01	IFPIND1	FF	000AC	01	IFPIND2	FF	000AD	01	IFPIND3	FF	000AE	01
IFPSWA	FF	00094	04	IFPSWB	FF	00098	04	IFPSWC	FF	0009C	04	IFPSWD	FF	000A0	04
IFPSWLB	FF	000A4	04	IFPSWRD	FF	0008C	04	IFPSWTR	FF	000A8	04	IFPSWWR	FF	00090	04
IGET	01	00394	04	IIAFTIND	FF	000AF	01	IIDALEN	FF	0008A	02	IIDLER	FF	00054	04
IIDLOC	FF	000A0	04	IILENAMB	FF	0005C	04	IILENAME	FF	0004C	07	IIOAREA1	FF	00048	04
IIOAREA2	FF	00074	04	IIOREG	FF	000AC	01	IISRKEY	FF	00053	01	IISRSEEK	FF	000CC	04
IKEYARG	FF	0009C	04	IKEYLEN	FF	00089	01	ILABADDR	FF	00040	04	ILABELRW	FF	00070	01
ILABNAME	FF	00044	04	ILHECON	FF	00080	01	ILPABT	01	003F8	04	ILPAFIN	01	00690	04
ILPAFINL	01	005F0	04	ILPAFT	01	004D2	04	ILPAH	01	00610	04	ILPBATCH	01	00720	04
ILPBAVA	01	00592	04	ILPBLOCK	01	00654	04	ILPBOUT	01	00746	04	ILPCARDR	01	00826	04
ILPCCWW	01	0070C	04	ILPCHECK	01	00686	06	ILPCKPTR	01	003F8	04	ILPCNTS	01	00738	06
ILPCOCCW	01	0060C	04	ILPETWOT	01	00570	04	ILPEXBLK	01	0068C	04	ILPEXEC	01	0070A	02
ILPEXECA	01	00702	01	ILPFINI	01	005E8	04	ILPFPUT	01	0074A	02	ILPFTERM	01	00664	04
ILPGETLA	01	0053A	06	ILPHDO	01	005A8	04	ILPHD1	01	005B4	04	ILPHD2	01	005C6	04
ILPLHEX	01	00602	02	ILPMONEF	01	00834	04	ILPOR	01	00788	06	ILPOTEST	01	0040C	04
ILPPBLK	01	007CC	04	ILPPCNTR	01	007B6	04	ILPPFIX	01	00810	06	ILPPIOR	01	007FC	04
ILPPOPUT	01	007C4	04	ILPPRET	01	007E2	04	ILPPRIAD	01	00852	04	ILPPRICC	01	008A4	04
ILPPRIN	01	004DE	04	ILPPRINT	01	0083E	04	ILPPRIRE	01	00882	02	ILPPRISA	01	00714	08
ILPPRIZ	01	00532	04	ILPPRNON	01	00526	04	ILPPRSEN	01	004E6	04	ILPPRSEX	01	0050C	04
ILPPRSW	01	0085E	02	ILPPRSWP	01	00864	04	ILPPRVAA	01	00886	04	ILPPRVAB	01	00892	04
ILPPUSER	01	007AA	04	ILPREGS	01	0054A	02	ILPRES	01	00618	04	ILPREST	01	0048C	04
ILPRET	01	0075E	04	ILPRETA	01	0076E	06	ILPRETB	01	00778	04	ILPRETC	01	00766	04
ILPREVA	01	00754	02	ILPRINV	01	008C2	06	ILPROUT	01	00750	04	ILPSETSW	01	005A0	04
ILPSKOV	01	0051A	04	ILPSOUT	01	006AC	04	ILPSUBR	01	006A2	04	ILPSUBR1	01	006D6	04
ILPSUBR2	01	006BE	06	ILPTCCWB	01	003E0	04	ILPTERES	01	00500	04	ILPTERM	01	003E4	01
ILPTERMA	01	004BE	01	ILPTERME	01	005D6	04	ILPTERMN	01	00550	04	ILPTERMW	01	003F4	04
ILPTERM1	01	003D6	04	ILPTERRE	01	003DA	04	ILPTEX	01	00430	04	ILPTMSG1	01	004BC	02
ILPTRSAV	01	0071C	04	ILPTSK	01	0047E	04	ILPTSKIP	01	00494	06	ILPTUN	01	005E0	04
ILPTUSE	01	00446	04	ILPTUSER	01	0046C	04	ILPTWLFB	01	004AA	04	ILPTWLFR	01	00438	02
ILPTWLP	01	00442	04	ILPTWLV	01	00434	01	ILPUNFRP	01	0072E	04	ILPUPLHE	01	00798	01

SYMBOL	CSECT	VALUE	L	SYMBOL	CSECT	VALUE	L	SYMBOL	CSECT	VALUE	L	SYMBOL	CSECT	VALUE	L
ILPUSEBY	01	007BC	04	ILPVAREX	01	008D8	04	ILPWAIT	01	006E6	01	ILPWAITT	01	006FE	04
ILPWAITX	01	006F2	04	ILPWAITZ	01	006EE	04	ILPWORT	01	004BC	01	ILPY	01	0084A	04
ILPYSREG	01	00794	04	ILPZERO	01	00742	01	ILSAV14	01	004A6	04	ILSTBLK	FF	000B0	04
ILSTNTRY	FF	000B8	04	ILSTRLTR	FF	000BC	03	IMRKCTR	FF	0003C	02	IN	01	00000	06
INB	01	0005C	04	INONEED	FF	000C4	04	INPUT	01	01078	50	IONTROLP	FF	000BF	01
IOPEN	01	00222	01	IOPEN1A	01	00222	04	IOVERFLO	FF	00088	01	IOVLYNAM	01	0021C	06
IOVRTGR	01	00384	04	IOVRTN	FF	00058	04	IOVRTT1	01	00388	04	IOVRT1A	01	00352	04
IOVRT1A1	01	00368	06	IOVRT1B	01	00376	04	IPRINTOV	FF	00072	01	IPUT	01	0038C	04
IREAD	FF	00088	01	IREADID	FF	00094	04	IREADKY	FF	00098	04	IRECFORM	FF	00088	01
IRECSIZE	FF	0007C	04	IRELADDR	FF	00088	01	IRELSE	01	0081A	04	IRELSE1	01	0081E	06
IRESERV	FF	0003F	01	IREWIND	FF	0003F	01	ISEEKADR	FF	000A4	04	ISRCHM	FF	00088	01
ISSAFTER	FF	00073	01	ITESTSW0	FF	00089	01	ITESTSW1	FF	00089	01	ITESTSW2	FF	00089	01
ITESTSW3	FF	00089	01	ITESTSW4	FF	00089	01	ITESTSW5	FF	00089	01	ITESTSW6	FF	00089	01
ITESTSW7	FF	00089	01	ITLEOV	01	00296	04	ITLEOVRT	01	002CA	04	ITLEOV1A	01	002AA	04
ITPMARK	FF	00088	01	ITRANS	FF	000B0	01	ITRUNC	01	0039C	04	ITRUNCEX	01	003D2	04
ITRUNC1	01	003A8	02	ITRUNC4	01	003C8	04	ITYPEFLE	FF	0003A	01	IUNUSED1	FF	00088	01
IUNUSED2	FF	00089	01	IUNUSED3	FF	000AC	01	IUNUSED9	FF	00088	01	IVAL1A	01	0022E	04
IVAL1B	01	0023C	04	IVAL1D	01	0027E	04	IVAL1E	01	00260	04	IVAL1G	01	0034C	04
IVAL1J	01	00286	04	IVAL1L	01	0028E	04	IVAL1M	01	0026E	04	IVARBLD	FF	000C4	04
IVBLKCNT	FF	00062	04	IWHATSIT	FF	00068	08	IWLRERR	FF	000C8	04	IWORKA	FF	00088	01
IWRITEID	FF	00080	04	IWRITEKY	FF	00084	04	LIST	01	00FF4	84	LOOP	01	00F3C	04
MAX	01	00FEC	02	OFLOW	01	00FC0	04	ONE	01	00FF0	01	OUT	01	000C0	06
OUTB	01	0011C	04	PRINTC	01	00FF3	01	RESET	01	00FF1	02	SAMPL	01	00000	01
WRITE	01	00F94	04												

```
SYMBOL     REFERENCES                         CROSS   REFERENCE   LISTING

BEGIN      00812*  00890
CTR        00852   00863   00869   00884*
EF         00052   00873*
IAFINAL    00197*  00576   00611   00698   00717
IAFTER     00188*
IAFTERID   00192*
IALTTAPE   001[illegible]
[illegible]
IWRITEKY   00198*
LIST       00822   00823   00823   00843   00844   00845   00846   00847   00848   00849   00850   00864   00865   00865
           00888*
LOOP       00835*  00866
MAX        00852   00883*
OFLOW      00853   00868*
ONE        00863   00885*
OUT        00079   00082*  00821   00832   00862   00878
OUTB       00080   00108*
PRINTC     00089   00103   00115   00118   00119   00824   00854   00868   00887*
RESET      00869   00886*
SAMPL      00005*  00291*
WRITE      00857*  00870
```

```
TOS ASSEMBLY PROGRAM                                                                                   PAGE 0016

FLAGS  LOCTN OBJECT CODE      ADDR1 ADDR2 STMNT M  SOURCE STATEMENT

                                          00001              ISEQ  76,80                                                TOS00000
                                          00002              PRINT NOGEN                                                TOS00010
                                          00003              MCALL DTFSR,DTFEN,OPEN,CLOSE,GET,PUT,TERM                  TOS00020
                                          00004              TITLE 'TOS ASSEMBLY PROGRAM'                               TOS01000
       00000                              00005   SAMPL      START                                                      TOS01010
                                          00006   *                                                                     TOS01020
                                          00007   * THIS SAMPLE PROGRAM ILLUSTRATES AN EDIT RUN FROM CARDS TO PRINTER   TOS01030
                                          00008   * USING STANDARD FCP AND ONE SOURCE DECK MACRO, WHICH IS INCLUDED TO  TOS01040
                                          00009   * SHOW HOW A USER MACRO CAN BE INCLUDED IN THE SOURCE DECK.           TOS01050
                                          00010   *                                                                     TOS01060
                                          00011   *                                                                     TOS01070
                                          00012   IN         DTFSR DEVADDR=SYS001,DEVICE=READER,                       CTOS01080
                                          00013                    TYPEFLE=INPUT,RECFORM=FIXUNB,                       CTOS01090
                                          00014                    IOAREA1=INPUT,BLKSIZE=80,                           CTOS01100
                                          00015                    EOFADDR=EF              READER - KEYWORD MACRO CALL  TOS01110


                                          00074   OUT        DTFSR DEVADDR=SYS002,DEVICE=PRINTER,                      CTOS01130
                                          00075                    TYPEFLE=OUTPUT,RECFORM=FIXUNB,                      CTOS01140
                                          00076                    IOAREA1=PRINTC,BLKSIZE=133,                         CTOS01150
                                          00077                    CTLCHR=YES,                                         CTOS01160
                                          00078                    ALTDEV=TAPE             PRINTER - KEYWORD MACRO CALL TOS01170
                                          00144              DTFEN                                                      TOS01180
```

```
FLAGS  LOCTN OBJECT CODE      ADDR1 ADDR2 STMNT M  SOURCE STATEMENT

         *                                                                  TOS02000
         * THIS IS THE MOVE MACRO DEFINITION, WHICH MOVES UP TO 80 BYTES(&LN), TOS02010
         * FROM ONE AREA(&FR) TO ANOTHER AREA(&TO), IN GROUP SIZES(&GP) FROM   TOS02020
         * 7 TO 80. FIVE SPACES ARE GIVEN BETWEEN EACH GROUP.                  TOS02030
         *                                                                  TOS02040
                  MACRO                            MACRO HEADER             TOS02050
         &NAME    MOVE   &FR,&TO,&LN,&GP           POSITIONAL PROTOTYPE     TOS02060
                  AIF    ('&LN' EQ '').MVC80       NO LENGTH? - MOVE 80     TOS02070
                  AIF    (&GP LT 7).ERROR          GROUPING LESS THAN 7? - ERROR TOS02080
                  AIF    ('&GP' EQ '').ONEMV       GROUPING NOT DESIRED? - 1 MOVE TOS02090
                  AIF    (&LN GT 80).ERROR         INPUT LENGTH GREATER 80 - ERROR TOS02100
         &AL1     SETA   0                         OUTPUT POSITION FOR NEXT GROUP TOS02110
         &AL2     SETA   0                         INPUT POSITION TO BE MOVED TOS02120
         &CG1     SETC   '&NAME'                   NAME FOR FIRST MOVE      TOS02130
         .LOOP    ANOP                             ENTRY POINT FOR EACH MOVE TOS02140
         &CG1     MVC    &TO+&AL1(&GP),&FR+&AL2    GENERATED MOVE INSTRUCTION TOS02150
         &CG1     SETC   ''                        REMOVE NAME ON SUBSEQUENT MOVES TOS02160
         &AL1     SETA   &AL1+&GP+5                UPDATE NEXT OUTPUT POSITION TOS02170
         &AL2     SETA   &AL2+&GP                  UPDATE NEXT INPUT POSITION TOS02180
                  AIFB   (&LN-&AL2 GE &GP).LOOP    CAN ANOTHER FULL GROUP BE MOVED? TOS02190
                  AIF    (&LN-&AL2 EQ 0).END       DID GROUPINGS COME OUT EVEN? TOS02200
                  MVC    &TO+&AL1(&LN-&AL2),&FR+&AL2    GEN. MOVE - LESS THAN GRP TOS02210
         .END     MEXIT                                                     TOS02220
         .MVC80   ANOP                                                      TOS02230
         &NAME    MVC    &TO(80),&FR               GENERATED MOVE OF 80     TOS02240
                  MEXIT                                                     TOS02250
         .ONEMV   ANOP                                                      TOS02260
         &NAME    MVC    &TO(&LN),&FR              GENERATED MOVE OF ACTUAL LENGTH TOS02270
                  MEXIT                                                     TOS02280
         .ERROR   MNOTE  6,'SPECIFICATIONS EXCEEDED - NO GENERATION'        TOS02290
                  MEND                             MACRO TRAILER            TOS02300
```

```
FLAGS  LOCTN OBJECT CODE      ADDR1 ADDR2 STMNT M  SOURCE STATEMENT

                                        00810     * MAIN ROUTINE                                                   TOS03000
                                        00811     *                                                                TOS03010
       00EF8 05 50                      00812     BEGIN     BALR  5,0                    REGISTER 5 COVERS CODING    TOS03020
       00EFA                            00813               USING *,5                                              TOS03030
                                        00814               OPEN  IN,OUT                 OPEN FILES                  TOS03040
       00F14 92 40 50FA      00FF4      00822               MVI   LIST,X'40'             CLEAR PRINT AREA            TOS03050
       00F18 D2 82 50FB 50FA 00FF5 00FF4 00823              MVC   LIST+1(131),LIST                                   TOS03060
       00F1E 92 C1 50F9      00FF3      00824               MVI   PRINTC,X'C1'           ADVANCE TO TOP OF FORM      TOS03070
                                        00825               PUT   OUT                                              TOS03080
                                        00833     LOOP      GET   IN                     READ CARD                   TOS03090
                                        00841               PRINT GEN                                              TOS03100
                                        00842               MOVE  INPUT,LIST,80,10       CALL TO MOVE 8 GROUPS OF 10 EACH TOS03110
       00F54 D2 09 50FA 517E 00FF4 01078 00843 M1           MVC   LIST+0(10),INPUT+0  GENERATED MOVE INSTRUCTION
       00F5A D2 09 5109 5188 01003 01082 00844 M1           MVC   LIST+15(10),INPUT+10  GENERATED MOVE INSTRUCTION
       00F60 D2 09 5118 5192 01012 0108C 00845 M1           MVC   LIST+30(10),INPUT+20  GENERATED MOVE INSTRUCTION
       00F66 D2 09 5127 519C 01021 01096 00846 M1           MVC   LIST+45(10),INPUT+30  GENERATED MOVE INSTRUCTION
       00F6C D2 09 5136 51A6 01030 010A0 00847 M1           MVC   LIST+60(10),INPUT+40  GENERATED MOVE INSTRUCTION
       00F72 D2 09 5145 51B0 0103F 010AA 00848 M1           MVC   LIST+75(10),INPUT+50  GENERATED MOVE INSTRUCTION
       00F78 D2 09 5154 51BA 0104E 010B4 00849 M1           MVC   LIST+90(10),INPUT+60  GENERATED MOVE INSTRUCTION
       00F7E D2 09 5163 51C4 0105D 010BE 00850 M1           MVC   LIST+105(10),INPUT+70  GENERATED MOVE INSTRUCTION
                                        00851               PRINT NOGEN                                            TOS03120
       00F84 F9 11 50F2 50F4 00FEC 00FEE 00852              CP    MAX,CTR                TEST FOR OVERFLOW           TOS03130
       TXT CARD # IS 0052.
       00F8A 47 80 50C6      00FC0      00853               BE    OFLOW                                            TOS03140
       00F8E 92 01 50F9      00FF3      00854               MVI   PRINTC,X'01'           SINGLE SPACE CHARACTER      TOS03150
                                        00855     WRITE     PUT   OUT                    PRINT LINE                  TOS03160
       00FAC FA 10 50F4 50F6 00FEE 00FF0 00863              AP    CTR,ONE                ADD 1 TO COUNTER            TOS03170
       00FB2 92 40 50FA      00FF4      00864               MVI   LIST,X'40'             CLEAR PRINT AREA            TOS03180
       00FB6 D2 82 50FB 50FA 00FF5 00FF4 00865              MVC   LIST+1(131),LIST                                   TOS03190
       00FBC 47 F0 5042      00F3C      00866               B     LOOP                                             TOS03200
                                        00867     * OVERFLOW AND CLOSE ROUTINES                                    TOS03210
       TXT CARD # IS 0053.
       00FC0 92 C1 50F9      00FF3      00868     OFLOW     MVI   PRINTC,X'C1'           PAGE CHANGE                 TOS03220
       00FC4 D2 01 50F4 50F7 00FEE 00FF1 00869              MVC   CTR(2),RESET                                       TOS03230
       00FCA 47 F0 509A      00F94      00870               B     WRITE                                            TOS03240
                                        00871     EF        CLOSE IN,OUT                                           TOS03250
                                        00879               TERM                                                   TOS03260
                                        00882     * CONSTANTS,COUNTERS,AND I/O AREAS                               TOS04010
       00FEC 060C                       00883     MAX       DC    X'060C'                MAXIMUM PAGE SIZE           TOS04030
       00FEE 000C                       00884     CTR       DC    X'000C'                LINE COUNTER                TOS04040
       00FF0 1C                         00885     ONE       DC    P'+1'                  SINGLE SPACE VALUE          TOS04050
       00FF1 000C                       00886     RESET     DC    X'000C'                ZERO FOR CLEARING LINE COUNTER TOS04060
       TXT CARD # IS 0054.
       00FF3                            00887     PRINTC    DS    CL1                    PRINT CONTROL CHARACTER     TOS04070
       00FF4                            00888     LIST      DS    CL132                  PRINT AREA                  TOS04080
       01078                            00889     INPUT     DS    CL80                   CARD INPUT AREA             TOS04090
       00EF8                            00890               END   BEGIN                                            TOS99999

  FLAGS IN 000000 STMNTS. VERSION NUMBER IS V009. ( TOS)

// LNKEDT
```

```
PROGRAM

      NAME OF PROGRAM        SAMPL                COMPUTED LENGTH          00004296       MAXIMUM LENGTH              00004296

                                                  NUMBER OF REGIONS             001       NUMBER OF OVERLAY POINTS         000
                                                  NUMBER OF SEGMENTS            001       NUMBER OF ENTRY POINTS         00007
                                                  NUMBER OF MODULES             001       STARTING EXECUTION ADDR.      000EFB
                                                  BLANK COMMON LENGTH      00000000       BLANK COMMON LOAD ADDR.       000000


    SEGMEN

         NAME OF SEGMENT     (ROOT)     NUMBER  001   SEGMENT LENGTH           00004296       STARTING ADDRESS              000000
                                                  SYMBOLIC OVERLAY POINT     (ROOT)       REGION NUMBER                    001
                                                  NEXT SEGMENT IN PATH       (ROOT)       NUMBER OF MODULES IN SEGMENT 001

        MODULES              NAME OF        LOAD         MODULE        NUMBER OF      METHOD USED TO
                                MODULE        ADDRESS      LENGTH        ENTRYS        BIND MODULES
                             ---------     --------     --------      ---------     -------------

                             SAMPL           000000     00004296        00007          EXPLICIT
***END LNKEDT


// COMM  OBJECT MODULE IS ON SYSUT2, NAMED SAMPL.

// COMM  THIS OBJECT MOD. MAY BE LOADED AND RUN UNDER EXEC. BY TYPING

// COMM  E LOD SAMPL,02,,R1

// ENDMON
```

```
  LISTING              OF                SAMPLE               DATA                CARDS

  FIELD1              FIELD2              FIELD3              FIELD4              FIELD5              FIELD6              FIELD7              FIELD8

  1 TO 10            11 TO 20            21 TO 30            31 TO 40            41 TO 50            51 TO 60            61 TO 70            71 TO 80

0000000001          1111111112          2222222223          3333333334          4444444445          5555555556          6666666667          7777777778
1234567890          1234567890          1234567890          1234567890          1234567890          1234567890          1234567890          1234567890

X                   X                   X                   X                   X                   X                   X                   X
XX                  XX                  XX                  XX                  XX                  XX                  XX                  XX
XXX                 XXX                 XXX                 XXX                 XXX                 XXX                 XXX                 XXX
XXXX                XXXX                XXXX                XXXX                XXXX                XXXX                XXXX                XXXX
XXXXX               XXXXX               XXXXX               XXXXX               XXXXX               XXXXX               XXXXX               XXXXX
XXXXXX              XXXXXX              XXXXXX              XXXXXX              XXXXXX              XXXXXX              XXXXXX              XXXXXX
XXXXXXX             XXXXXXX             XXXXXXX             XXXXXXX             XXXXXXX             XXXXXXX             XXXXXXX             XXXXXXX
XXXXXXXX            XXXXXXXX            XXXXXXXX            XXXXXXXX            XXXXXXXX            XXXXXXXX            XXXXXXXX            XXXXXXXX
XXXXXXXXX           XXXXXXXXX           XXXXXXXXX           XXXXXXXXX           XXXXXXXXX           XXXXXXXXX           XXXXXXXXX           XXXXXXXXX
XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX
XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX
XXXXXXXXX           XXXXXXXXX           XXXXXXXXX           XXXXXXXXX           XXXXXXXXX           XXXXXXXXX           XXXXXXXXX           XXXXXXXXX
XXXXXXXX            XXXXXXXX            XXXXXXXX            XXXXXXXX            XXXXXXXX            XXXXXXXX            XXXXXXXX            XXXXXXXX
XXXXXXX             XXXXXXX             XXXXXXX             XXXXXXX             XXXXXXX             XXXXXXX             XXXXXXX             XXXXXXX
XXXXXX              XXXXXX              XXXXXX              XXXXXX              XXXXXX              XXXXXX              XXXXXX              XXXXXX
XXXXX               XXXXX               XXXXX               XXXXX               XXXXX               XXXXX               XXXXX               XXXXX
XXXX                XXXX                XXXX                XXXX                XXXX                XXXX                XXXX                XXXX
XXX                 XXX                 XXX                 XXX                 XXX                 XXX                 XXX                 XXX
XX                  XX                  XX                  XX                  XX                  XX                  XX                  XX
X                   X                   X                   X                   X                   X                   X                   X
         X                   X                   X                   X                   X                   X                   X                   X
        XX                  XX                  XX                  XX                  XX                  XX                  XX                  XX
       XXX                 XXX                 XXX                 XXX                 XXX                 XXX                 XXX                 XXX
      XXXX                XXXX                XXXX                XXXX                XXXX                XXXX                XXXX                XXXX
     XXXXX               XXXXX               XXXXX               XXXXX               XXXXX               XXXXX               XXXXX               XXXXX
    XXXXXX              XXXXXX              XXXXXX              XXXXXX              XXXXXX              XXXXXX              XXXXXX              XXXXXX
   XXXXXXX             XXXXXXX             XXXXXXX             XXXXXXX             XXXXXXX             XXXXXXX             XXXXXXX             XXXXXXX
  XXXXXXXX            XXXXXXXX            XXXXXXXX            XXXXXXXX            XXXXXXXX            XXXXXXXX            XXXXXXXX            XXXXXXXX
 XXXXXXXXX           XXXXXXXXX           XXXXXXXXX           XXXXXXXXX           XXXXXXXXX           XXXXXXXXX           XXXXXXXXX           XXXXXXXXX
XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX
XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX
 XXXXXXXXX           XXXXXXXXX           XXXXXXXXX           XXXXXXXXX           XXXXXXXXX           XXXXXXXXX           XXXXXXXXX           XXXXXXXXX
  XXXXXXXX            XXXXXXXX            XXXXXXXX            XXXXXXXX            XXXXXXXX            XXXXXXXX            XXXXXXXX            XXXXXXXX
   XXXXXXX             XXXXXXX             XXXXXXX             XXXXXXX             XXXXXXX             XXXXXXX             XXXXXXX             XXXXXXX
    XXXXXX              XXXXXX              XXXXXX              XXXXXX              XXXXXX              XXXXXX              XXXXXX              XXXXXX
     XXXXX               XXXXX               XXXXX               XXXXX               XXXXX               XXXXX               XXXXX               XXXXX
      XXXX                XXXX                XXXX                XXXX                XXXX                XXXX                XXXX                XXXX
       XXX                 XXX                 XXX                 XXX                 XXX                 XXX                 XXX                 XXX
        XX                  XX                  XX                  XX                  XX                  XX                  XX                  XX
         X                   X                   X                   X                   X                   X                   X                   X
XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX
 XXXXXXXX            XXXXXXXX            XXXXXXXX            XXXXXXXX            XXXXXXXX            XXXXXXXX            XXXXXXXX            XXXXXXXX
  XXXXXX              XXXXXX              XXXXXX              XXXXXX              XXXXXX              XXXXXX              XXXXXX              XXXXXX
   XXXX                XXXX                XXXX                XXXX                XXXX                XXXX                XXXX                XXXX
    XX                  XX                  XX                  XX                  XX                  XX                  XX                  XX
    XX                  XX                  XX                  XX                  XX                  XX                  XX                  XX
   XXXX                XXXX                XXXX                XXXX                XXXX                XXXX                XXXX                XXXX
  XXXXXX              XXXXXX              XXXXXX              XXXXXX              XXXXXX              XXXXXX              XXXXXX              XXXXXX
 XXXXXXXX            XXXXXXXX            XXXXXXXX            XXXXXXXX            XXXXXXXX            XXXXXXXX            XXXXXXXX            XXXXXXXX
XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX          XXXXXXXXXX
```

```
  LISTING         OF           SAMPLE          DATA           CARDS

  FIELD1         FIELD2         FIELD3         FIELD4         FIELD5         FIELD6         FIELD7         FIELD8

  1 TO 10       11 TO 20       21 TO 30       31 TO 40       41 TO 50       51 TO 60       61 TO 70       71 TO 80

0000000001     1111111112     2222222223     3333333334     4444444445     5555555556     6666666667     7777777778
1234567890     1234567890     1234567890     1234567890     1234567890     1234567890     1234567890     1234567890

XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX
 XXXXXXXX       XXXXXXXX       XXXXXXXX       XXXXXXXX       XXXXXXXX       XXXXXXXX       XXXXXXXX       XXXXXXXX
  XXXXXX         XXXXXX         XXXXXX         XXXXXX         XXXXXX         XXXXXX         XXXXXX         XXXXXX
   XXXX           XXXX           XXXX           XXXX           XXXX           XXXX           XXXX           XXXX
    XX             XX             XX             XX             XX             XX             XX             XX
    XX             XX             XX             XX             XX             XX             XX             XX
   XXXX           XXXX           XXXX           XXXX           XXXX           XXXX           XXXX           XXXX
  XXXXXX         XXXXXX         XXXXXX         XXXXXX         XXXXXX         XXXXXX         XXXXXX         XXXXXX
 XXXXXXXX       XXXXXXXX       XXXXXXXX       XXXXXXXX       XXXXXXXX       XXXXXXXX       XXXXXXXX       XXXXXXXX
XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX
X              X              X              X              X              X              X              X
XX             XX             XX             XX             XX             XX             XX             XX
XXX            XXX            XXX            XXX            XXX            XXX            XXX            XXX
XXXX           XXXX           XXXX           XXXX           XXXX           XXXX           XXXX           XXXX
XXXXX          XXXXX          XXXXX          XXXXX          XXXXX          XXXXX          XXXXX          XXXXX
XXXXXX         XXXXXX         XXXXXX         XXXXXX         XXXXXX         XXXXXX         XXXXXX         XXXXXX
XXXXXXX        XXXXXXX        XXXXXXX        XXXXXXX        XXXXXXX        XXXXXXX        XXXXXXX        XXXXXXX
XXXXXXXX       XXXXXXXX       XXXXXXXX       XXXXXXXX       XXXXXXXX       XXXXXXXX       XXXXXXXX       XXXXXXXX
XXXXXXXXX      XXXXXXXXX      XXXXXXXXX      XXXXXXXXX      XXXXXXXXX      XXXXXXXXX      XXXXXXXXX      XXXXXXXXX
XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX
XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX
XXXXXXXXX      XXXXXXXXX      XXXXXXXXX      XXXXXXXXX      XXXXXXXXX      XXXXXXXXX      XXXXXXXXX      XXXXXXXXX
XXXXXXXX       XXXXXXXX       XXXXXXXX       XXXXXXXX       XXXXXXXX       XXXXXXXX       XXXXXXXX       XXXXXXXX
XXXXXXX        XXXXXXX        XXXXXXX        XXXXXXX        XXXXXXX        XXXXXXX        XXXXXXX        XXXXXXX
XXXXXX         XXXXXX         XXXXXX         XXXXXX         XXXXXX         XXXXXX         XXXXXX         XXXXXX
XXXXX          XXXXX          XXXXX          XXXXX          XXXXX          XXXXX          XXXXX          XXXXX
XXXX           XXXX           XXXX           XXXX           XXXX           XXXX           XXXX           XXXX
XXX            XXX            XXX            XXX            XXX            XXX            XXX            XXX
XX             XX             XX             XX             XX             XX             XX             XX
X              X              X              X              X              X              X              X
         X              X              X              X              X              X              X              X
        XX             XX             XX             XX             XX             XX             XX             XX
       XXX            XXX            XXX            XXX            XXX            XXX            XXX            XXX
      XXXX           XXXX           XXXX           XXXX           XXXX           XXXX           XXXX           XXXX
     XXXXX          XXXXX          XXXXX          XXXXX          XXXXX          XXXXX          XXXXX          XXXXX
    XXXXXX         XXXXXX         XXXXXX         XXXXXX         XXXXXX         XXXXXX         XXXXXX         XXXXXX
   XXXXXXX        XXXXXXX        XXXXXXX        XXXXXXX        XXXXXXX        XXXXXXX        XXXXXXX        XXXXXXX
  XXXXXXXX       XXXXXXXX       XXXXXXXX       XXXXXXXX       XXXXXXXX       XXXXXXXX       XXXXXXXX       XXXXXXXX
 XXXXXXXXX      XXXXXXXXX      XXXXXXXXX      XXXXXXXXX      XXXXXXXXX      XXXXXXXXX      XXXXXXXXX      XXXXXXXXX
XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX
XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX     XXXXXXXXXX
 XXXXXXXXX      XXXXXXXXX      XXXXXXXXX      XXXXXXXXX      XXXXXXXXX      XXXXXXXXX      XXXXXXXXX      XXXXXXXXX
  XXXXXXXX       XXXXXXXX       XXXXXXXX       XXXXXXXX       XXXXXXXX       XXXXXXXX       XXXXXXXX       XXXXXXXX
   XXXXXXX        XXXXXXX        XXXXXXX        XXXXXXX        XXXXXXX        XXXXXXX        XXXXXXX        XXXXXXX
    XXXXXX         XXXXXX         XXXXXX         XXXXXX         XXXXXX         XXXXXX         XXXXXX         XXXXXX
     XXXXX          XXXXX          XXXXX          XXXXX          XXXXX          XXXXX          XXXXX          XXXXX
      XXXX           XXXX           XXXX           XXXX           XXXX           XXXX           XXXX           XXXX
       XXX            XXX            XXX            XXX            XXX            XXX            XXX            XXX
        XX             XX             XX             XX             XX             XX             XX             XX
         X              X              X              X              X              X              X              X
```

```
E LOD MON,,,R1,,4Ø5ØØ
V MON     Ø2L6 ØØ2Ø52
U MON    Ø3RT  ØØ:Ø1:26
U MON    ØØ:21:Ø8 JOB  TOS MONITOR
V ASSMBL Ø2L6 ØØ2116
U ASSMBL Ø399
V LNKEDT Ø2L6 ØØ2715
6 LNKEDT Ø899 ***END LNKEDT
U LNKEDT Ø399
U MON    OBJECT MODULE IS ON SYSUT2, NAMED SAMPL.
U MON    THIS OBJECT MOD. MAY BE LOADED AND RUN UNDER EXEC. BY TYPING
U MON    E LOD SAMPL,Ø2,,R1
U MON    Ø3RT  ØØ:Ø6:5Ø
V MON    Ø2NH ØØ28ØØ

E LOD SAMPL,Ø2,,R1
V SAMPL  Ø2L6 ØØ2824
6 SAMPL  5Ø86  IN      R1 ØØØØ355
6 SAMPL  5Ø86  OUT     L1 ØØØØ356
V SAMPL  Ø2NH ØØ2949 ØØØ121
```

Spectra 70 70-00-602

POS/TOS/TDOS Assembly

System Reference Manual

Your comments, accompanied by answers to the following questions help us produce better publications. If your answer to a question is "no," or requires qualification, please explain on a separate sheet of paper. Please give specific page and line references with comments when appropriate. If you desire a reply, be sure to include your name and address.

	YES	NO
Does this publication meet your needs?		
Did you find the material:		
Easy to read and understand?		
Organized for convenient use?		
Complete?		
Well illustrated?		
Written for your technical level?		

What is your occupation? ____________________

STAPLE STAPLE

FOLD FOLD

FIRST CLASS
PERMIT NO. 16
CAMDEN, NEW JERSEY

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY—

RADIO CORPORATION OF AMERICA
ELECTRONIC DATA PROCESSING
CAMDEN, NEW JERSEY 08101

ATTN: Manager, Systems Software Services
Cherry Hill
Building 204-2

CUT ALONG LINE

FOLD FOLD