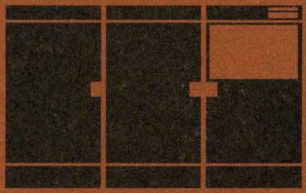
GENERAL ELECTRIC
COMPUTERS

DATANET-30 Communication Assembly Program



GENERAL ELECTRIC

CPB – 1074

DATANET-30 COMMUNICATION ASSEMBLY PROGRAM (COMMA)

Program Number
CDD30F2.001
CDD30F2.002

OCTOBER 1965

GENERAL ELECTRIC
COMPUTER DEPARTMENT

PREFACE

This manual is provided by General Electric Computer Department for the programmers and operators of DATANET-30 Communications Processing Systems. The manual describes the DATANET-30 Communication Assembly Program (COMMA) and is organized in two parts so that Part I can be used as a teaching aid and thereafter as a reference manual. It is recommended that a programmer have a working knowledge of the DATANET-30 system as described in the DATANET-30 Reference Manual, CPB-1019. Part II is directed to the more experienced programmer.

A glossary is included for readers who are not familiar with assemblers for data communications and symbolic programming.

Comments concerning this manual should be addressed to Technical Publications, General Electric Computer Department, P.O. Box 2561 Phoenix, Arizona, 85001.

TABLE OF CONTENTS

ILLUSTRATIONS

PART I

D E S C R I P T I O N   O F   T H E   P R O G R A M

GENERAL

The DATANET*-30 Communications Processor is a single-address, stored-program, special-purpose digital computer operating in straight binary but processing both alphanumeric (BCD or Baudot) and binary information. It can operate as a self-contained, free-standing terminal system; or it can become a part of a GE-600, -400, or -200 Series Information Processing System, sharing those peripherals with which the data processing systems are compatible.

With such a wide variety of peripheral configurations available, it is not practical to establish a standard peripheral configuration for an assembly program. Therefore, the DATANET-30 Communication Assembly Program (COMMA) never directly addresses a peripheral. Instead, it works with "files" in place of peripherals. Working with files, COMMA branches to predetermined memory locations which serve as assembly links to subroutines in the COMMA Input/Output Links (COIL). Constructed of "plug-in" subprograms, COIL permits changes to be made in peripheral configurations simply by changing these subprograms.

COMMA translates symbolic codes and instruction mnemonics used by the programmer into a ready-to-execute object program. The source program can be assembled into an object program from a punch card deck, perforated tape, magnetic tape (operating system tape) or from assemblies with remote terminals, such as Teletype equipment.

Since COMMA works with input and output (I/O) peripherals through predefined subroutines, a variety of peripheral inputs and outputs are provided. For example, when the assembler needs a new source document, it branches to a particular linkage point in COIL. When COIL returns to the assembler, the assembler expects an 80-column BCD record at a given memory location. It is immaterial whether the record--source document--came from an on-line card reader; from a magnetic tape written by a satellite computer system; from another computer over a high-speed transmission line or through a computer interface unit; from a dual-access disc storage unit (DSU) where it was placed by another

* DATANET is a registered trademark of General Electric Company.

computer; from Teletype; or from a perforated tape reader. Nor does it matter whether the record was always in BCD code or originated in some other code and was converted by COIL. The record may originally have been either in 80 columns or in a condensed record, spread and blank-filled by COIL.

Specialized COIL subprograms are provided with COMMA; however, specifications for writing a COIL subprogram are included in this manual (Chapter 4), so that users may modify or write their own. Not all COIL subprograms will make use of the full power of the assembler. For example, automatic updates of source programs or subroutine calls from library are not really practical unless one has access to bulk storage devices such as magnetic tapes or DSU. A basic assembly requires a minimum of one input device and one output device (capable of handling both BCD and binary data). The DATANET-30 can assemble with only a DSU, provided there is a way to get the source program on and the object listing off the DSU. (This can be accomplished by a conventional computer which shares the DSU with the DATANET-30.)

Object programs can be obtained from source programs written in either of two formats, old or new. Old format (described in DATANET-30 General Assembly Program, CPB-1180, is recognized by COMMA and translated to new format for processing. Object programs obtained in either format can be combined, assembled, or reassembled to operate a DATANET-30 system.

PROGRAM SEQUENCE

An assembly for communication processing differs from batched data processing in its required real-time operations concept. In batched data processing, assemblies are frequently required; and the terms "read a card" or "print a line" are completely dependent upon the source program memory utilization for extensive blocking or buffering of records and for obtaining peak efficiency from the peripherals. A DATANET-30 system is most often used in situations where the same program (or program set) runs day in and day out and an assembly is required infrequently.

COMMA performs only such tasks as formatting, packing and unpacking records, and all work independent of the I/O devices. COIL must contain the routines to perform all of the functions dependent upon the I/O devices. COIL, therefore, must include not only the provisions for reading and writing all records but also all error checks (and associated recovery procedures), end-of-file and end-of-tape checks, labeling, etc.

COMMA processes the source program in 80-column (punch card) format in two passes (three assembly phases) to obtain an object program. Figure 1 shows the assembly program's data flow. Inputs and outputs (files) are shown simply as boxes in the diagram, because they may come from any peripheral.

Analyzer Phase

During the analyzer phase, the instructions of the source program (possibly a merger of several files) are analyzed to generate the symbol tables and a work file. The work file contains the source card image and control words, and it serves as an input to the object phase.

Object Phase

The object phase reads the work file, completes the assembly's operations, and generates an object file and object listing file. All information required by the reports phase is left in memory, so that no additional pass of the work file is necessary.

Reports Phase

The reports phase sorts and lists the symbol table and generates the closing reports.

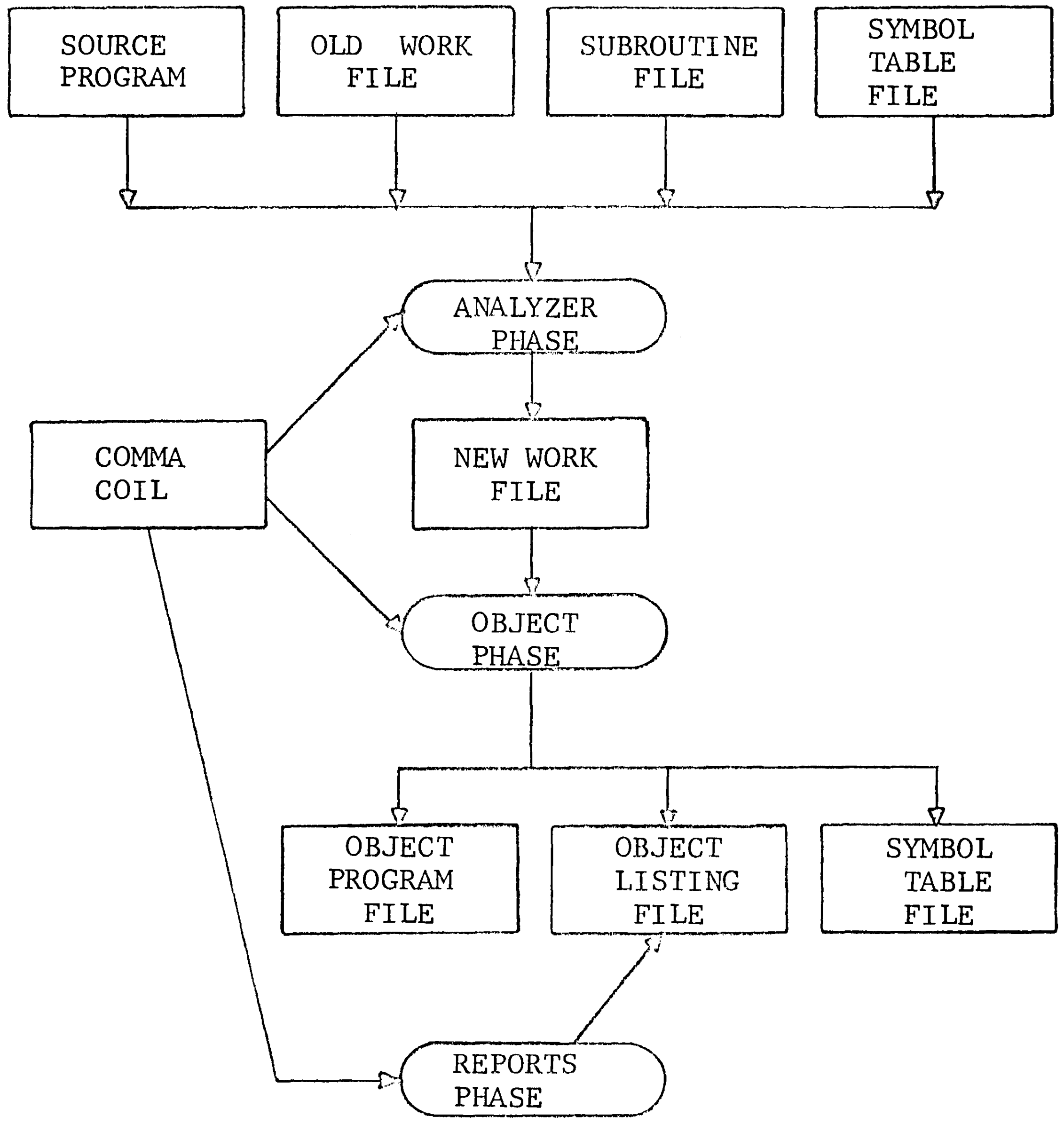


Figure 1. Diagram of the Assembly

A S S E M B L Y   L A N G U A G E

The source program for COMMA is written using pseudo-instructions, reference addresses, and mnemonics of operations to be performed. The cards punched for each line of the programmer's coding sheet become the source program card deck used in the translator phase of COMMA. (Refer to the DATANET-30 Reference Manual for the mnemonics of operations to be performed.)

PROGRAMMING FORM

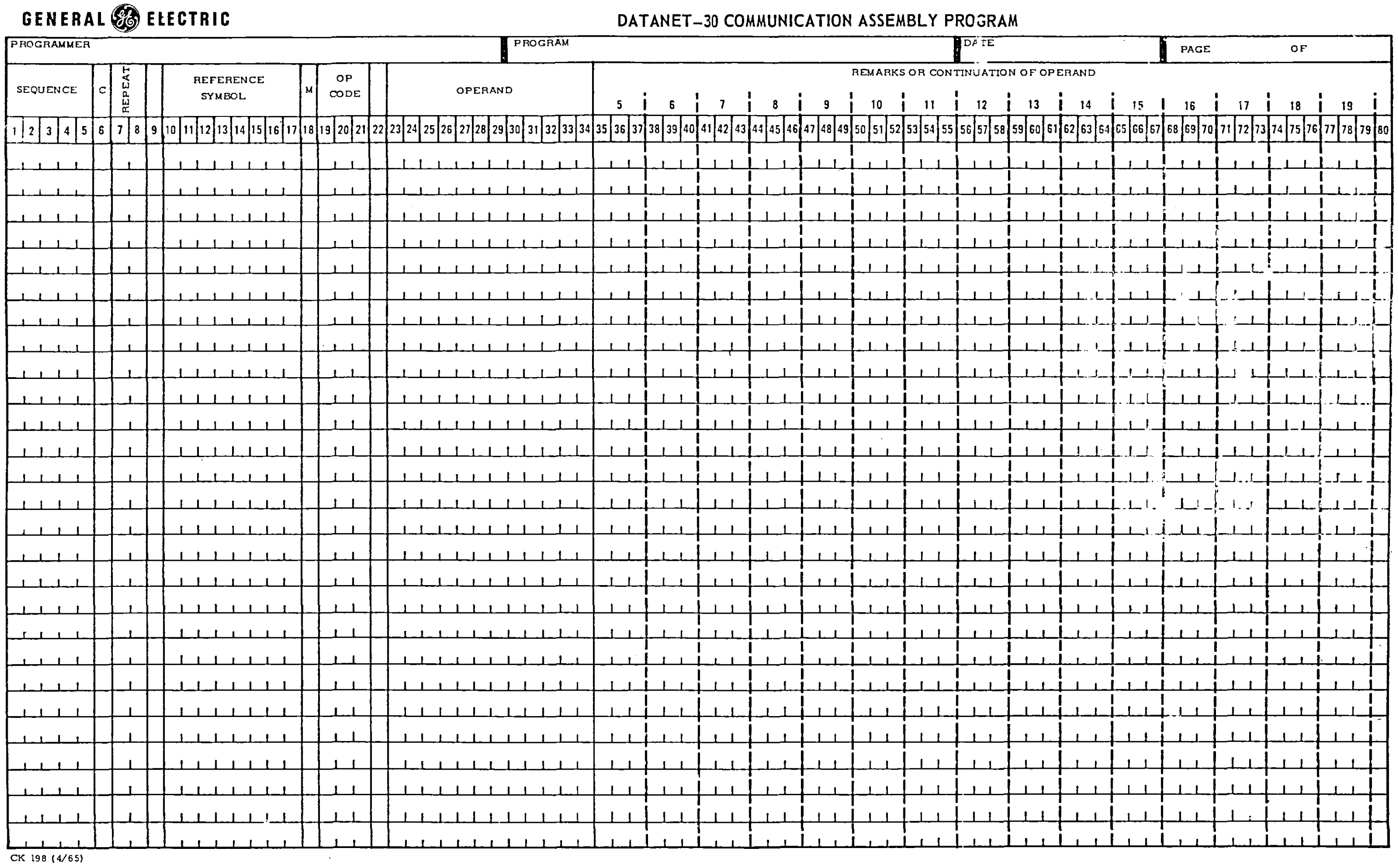
DATANET-30 COMMA programming form (CK-198), shown in Figure 2, can be used for source programs--new format. The form represents an 80-column card into which each line entry is punched.

In the paragraphs following, the field headings on the programming form are described according to their usage. Columns 9 and 22 are not used by the assembler for coding purposes.

Sequence Field (Columns 1-5)

To ensure proper sequence for the input of the source program, the programmer assigns sequence numbers in columns 1-5 of his programming form. During program assembly, the sequence of numbers is usually ignored; however, the assembler can be instructed to check the source cards for proper ascending sequence by issuing an SEQ pseudo-instruction.

Sequence numbers may be restarted as often as necessary. Some subroutines frequently contain their own sequence numbers, thus necessitating a restart of the sequence check for each new subroutine. If sequence checking is requested, the assembler flags any source card whose sequence number is not greater than that of the preceding card. All nonnumeric characters in the Sequence field are treated as zeros by the assembler. Sequence numbers become most important in maintaining work files (see "Update Assembly," p. 86), because they function as the identification key in updating or changing files without requiring complete reassembly of the file records.



GENERAL ELECTRIC DATANET-30 COMMUNICATION ASSEMBLY PROGRAM

PROGRAMMER	PROGRAM	DATE	PAGE OF

SEQUENCE	C	REPEAT		REFERENCE SYMBOL	M	OP CODE		OPERAND	REMARKS OR CONTINUATION OF OPERAND
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	35–80 (5 6 7 8 9 10 11 12 13 14 15 16 17 18 19)

CK 198 (4/65)

Figure 2. Sample Programming Form

Control Field (Column 6)

The Control field is used by the assembler to identify an operation to be performed according to the following codes:

C--Identifies a card as being an assembly control card, (see Chapter 3).

D--Identifies a deletion card. If console switch 17 is down, COMMA deletes this card from the object program. This code provides the user with the ability to delete coding inserted for debug purposes.

*--Identifies a card as being a message with information in columns 7-80 to be printed on the object program listing.

%--Identifies a card as being a message and causes the printer to slew 2 lines, print the message from columns 7-80, then slew 2 lines to the next print line. The % sign and sequence number of the instruction are inhibited from printing. This option is normally used to print title lines on the object program listing.

Any code in column 6 other than one of those listed above causes the assembler to indicate a C in the Flag field on the object listing, denoting an error.

Repeat Field (Columns 7-8)

The Repeat field is used by the assembler to generate identical units (words) for the instruction which follows. The field may be numeric (blank to 99) or symbolic.

If a symbol is used in the Repeat field, it is limited to two alphabetic characters and must be predefined by an EQU (Equal) or EQO (Equal Octal) equating the symbol to its numeric value. Failure to predefine a Repeat field symbol properly results in an error condition that is not correctable without reassembly. The assembler is not able to repeat for a symbol not previously defined.

If a numeric value is shown in this field it causes the assembler to repeat the instruction contained on the line for the indicated number of times. For example, to preset a 68-word block of memory to zero, the DEC 0 instruction would have 67 in the Repeat field, as shown below:

Memory Loc.	Instruction	Sequence	Repeat	Symbol	Opr	Operand
6000	000000	4840	67	TI	DEC	0
6001	000000					
6002	000000					
≀	≀					
6103	000000					
6104		4850				

To enter messages in BCD or Teletype codes, the Repeat field tells how many computer words are required for the message. For example, to enter the message OUT OF STOCK, the ALF pseudo-instruction would have a 4 in the Repeat field:

Memory Loc.	Instruction	Sequence	Repeat	Symbol	Opr	Operand
5000	466463	3220	4		ALF	OUT OF STOC
5001	604626					
5002	606263					
5003	462342					
5004		3221				

If the Repeat field is blank on a message-type pseudo-instruction, the assembler assumes a one-word (three-character) message.

Reference Symbol Field (Columns 10-17)

The Reference Symbol field is used to assign symbols to instructions, data, or pseudo-instruction constants. The symbol represents the relative memory location in which the data will be located when the program is loaded for execution. Reference symbols in COMMA are of two types, defined and redefinable.

- Defined Symbols. The following rules govern the construction and use of defined symbols:

  1. Symbols can vary from one to eight characters in length and be any combination of alphanumeric and special characters, except plus, minus, or comma.

  2. Symbols must contain at least one nonnumeric character (A-Z or special character except plus, minus, or comma).

3. Symbols may start at any point within the field. Leading or imbedded blanks are deleted by the assembler as it analyzes the Reference Symbol field.

4. Symbols of seven or fewer characters may be prefixed by the assembler program. (See PFX pseudo-instruction.)

5. A symbol must be defined only once within a source program. More than one such definition results in an error (code M) indication by the assembler each time the symbol is used.

o <u>Redefinable Symbols.</u> Symbols consisting exclusively of pound signs (#) are reserved for special usage. These symbols are never entered into the assembler symbol table but are kept in a separate table by the assembler. Redefinable symbols can be a set of as many as eight pound signs, and any number in a set may be reused as often as desired in a backward direction. Each time the symbol is reused, it takes a new value.

  Any time a redefinable symbol is named in an Operand field, it equals the last value assigned to that symbol and cannot be equated to other values. Redefinable symbols cannot be forced to odd or even locations and are illegal as operands with some pseudo-instructions.

  The main purpose for redefinable symbols is to reduce the demand on the symbol table of the assembler and to reduce the probability of symbol conflict between subroutines and program segments.

Example: Redefinable symbols used in lieu of the asterisk in tight loops.

REFERENCE SYMBOL	M	OP CODE		OPERAND	REMARKS
		PIC		0	MOVE INPUT TO OUTP
#		LDD		$INPUT	
		STD		$OUTPUT	
		AIC		2	
		XCZ		38	
		BNZ		#	
#		NIS		7	
		BNZ		#	
#		CSR		6	
		BEV		#	

Modifier Field (Column 18)

The Modifier field is used to tell the assembler to modify an operation or address. The following codes are used:

A--To set the absolute address in the MIC pseudo-instruction.

D--To place this instruction in an even-numbered location.

I--To indicate more than one instruction or an immediate-call option, as shown by the following examples:

a. The reference symbol refers to the first instruction only and generates the three instructions separated by slashes.

Symbol	Modifier	Opr	Operand
MOVE	I	PIC	0/LDDIN/STDOUT

b. If the instruction is an SBR, it means the subroutine is to be called immediately (immediate-call option).

Symbol	Modifier	Opr	Operand
SPREAD	I	SBR	BUFRD

M--To indicate multiple or maximum operands, as shown in the following examples:

a. The assembler will generate one instruction (multiple) for each operand on the card. The operands must be separated from each other by slashes.

Symbol	Modifier	Opr	Operand
	M	BRS	FILL/MOVE/READ/END

b. The assembler will choose the higher of two addresses (maximum) of each operand on the card. The operands must be separated from each other by commas. The selection of maximum addresses can be used only with BSS, EQO, EQU, LOC, or ORG pseudo-instructions.

Symbol	Modifier	Opr	Operand
	M	ORG	END1+1,END2+1

O--To indicate that the symbols in the Operand field are not to be prefixed.

S--To indicate that the reference symbols are not to be prefixed.

/--To indicate that the reference symbol on this card is to be ignored. This code can be used to label the rows in a table for documentation.

X--To indicate an MAC instruction with a partial operand or to indicate that the operand will be provided at a later time.

Op (Operation) Code Field (Columns 19-21)

The mnemonics of the operation or pseudo-operation are specified in the Operation Code field. These mnemonics indicate the operation to be performed. A blank or an illegal code in this field is flagged with 0 in the object program listing, and the instruction is replaced with a Halt (00_8) instruction.

Operand Field (Columns 23-34)

The Operand field is in free-floating format and terminated by a blank or, if indirect addressing is used, by a comma. Imbedded blanks are not permitted between elements in an expression; however, blanks are accepted in alphanumeric data-generating pseudo-operations.

An operand may start anywhere between columns 23 and 30, inclusive, and may extend to the end of the line (column 80).

Double asterisks can be used in the Operand field to indicate a "blank operand ok" when the operands are to be set at execution time.

Remarks Field (Columns 35-80)

Remarks may start anywhere after the operand. The only requirement is that there be at least one blank between the Operand and Remarks fields. Remarks should start in the same column throughout the program for neatness of documentation on the object program.

Lines containing only remarks can be obtained on the object program listing by placing an asterisk in the Control field and writing the information to be documented in columns 7 to 80. Note that columns 9 and 22 are not used for coding purposes; however, they are used on remarks cards.

CARD FORMAT

The format for the source card is shown in Figure 3.

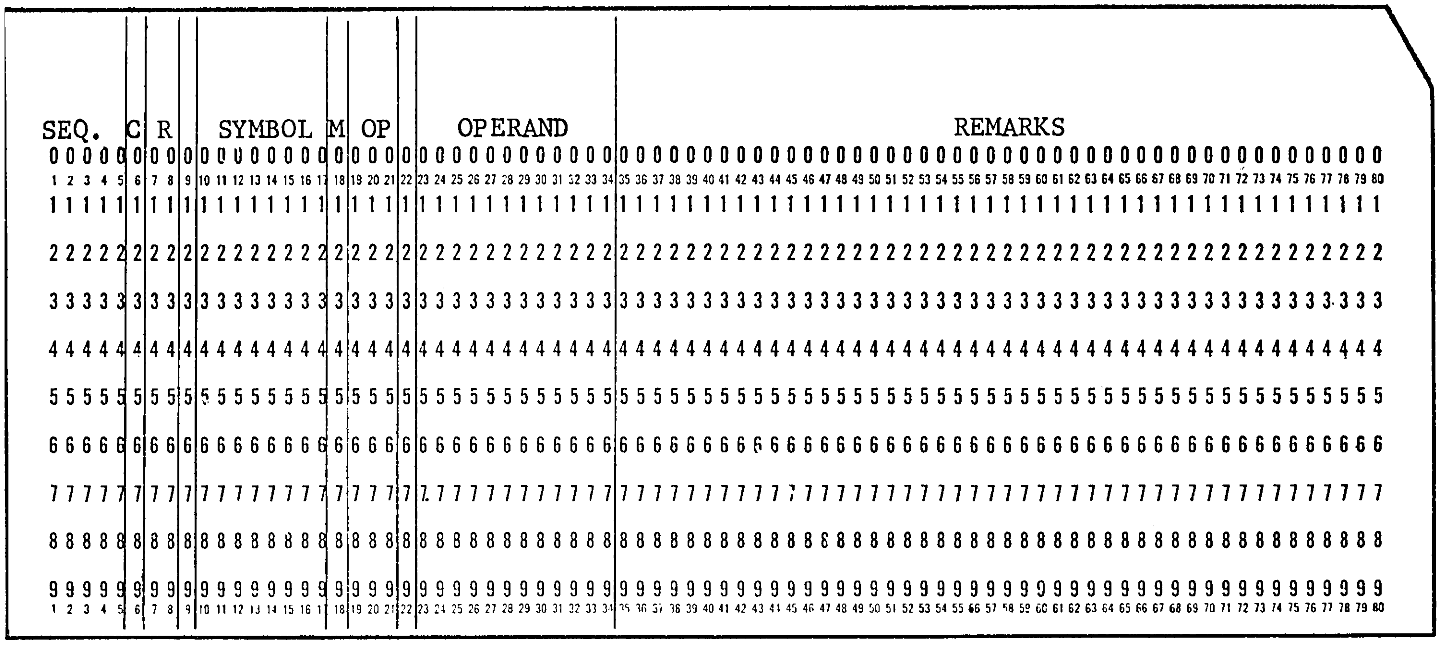


Figure 3. Source Card Format

PSEUDO-INSTRUCTIONS

Pseudo-instructions are not executed by the DATANET-30 but are used by COMMA to generate constants, control the assembly program, and provide information on the object listing. The COMMA pseudo-instructions are listed below under their classifications--data-generating and control--showing the type of function each performs. The instructions are then described in detail in alphabetic order by their mnemonics.

Data-Generating Instructions

Data-generating instructions are those which generate constants, messages, tables, or addresses.

o Constants. Each of the following instructions generates data from the information shown in the Operand field of the instruction and stores it in a memory location:

DEC	Generates one binary word from decimal data.
DDC	Generates two binary words from decimal data.
LNK	Generates a two-word linkage address.
MAC	Generates and defines a macro-instruction.
MIC	Generates and defines a micro-instruction.
OCT	Assembles one binary word from octal notation.

o Messages. Each of the following instructions generates a single data word or group of data words as shown in the Operand field of the instruction and stores the data in consecutive memory locations:

ALF	Assembles three alphanumeric characters per word.
NAL	Assembles three alphanumeric characters per word (2's complement).
T5L	Defines Teletype code--left justified--three characters per word.
T5R	Defines Teletype code--right justified--three characters per word.

o Tables. Processing communication messages usually require extensive use of tables. The following instructions help to minimize the work of constructing tables:

APO	Adds previous operands to the value in a table.
CDC	Constructs constants from CDN's.
CDN	Defines elements of a CDC.
FDN	Defines a field for MFC.
MFC	Defines multiple field constants.
SAP	Stops adding previous operands started by APO.

o Address. Accessing locations outside the current memory bank and indexing any instruction is accomplished only through indirect addressing. Indirect addressing may be done with no indexing or with indexing by either the A-, B-, or C-register. The indication of indexing and the indirect address as an absolute binary number are contained in one word, which may be set up by one of the following pseudo-operations:

INA	Indirect addresses using the A-register.
INB	Indirect addresses using the B-register.
INC	Indirect addresses using the C-register.
IND	Indirect addressing.

Control Instructions

Control instructions provide information to the internal operations of the assembly program. They do not become a part of the assembled program but tell the assembler how to assign memory, how to use peripherals during assembly, how to document the object program listing, and how to control the assembly program.

o Memory Allocation. Each of the following instructions tells the assembler how to assign memory locations in the object program:

BSS	Reserves memory blocks.
EQO	Equates a symbol to an octal memory address.
EQU	Equates a symbol to a decimal memory address.
EVN	Advances memory allocation register (MAR) to next even address.
INH	Inhibits movement of a symbol by a double-length reference.
LOC	Resets MAR from an octal number.
ODD	Advances MAR to next odd address.
ORG	Resets MAR from a decimal number.

o Documentation. Each of the following instructions assists a programmer in documenting his program:

EJT	Slews listing to next page.
LST	Resumes printer listing (after NLS).
NLS	Stops printer listing.
SEQ	Calls for sequence checking.
TOC	Prints table of contents.
TTL	Prints title line on each page.

- Program. Each of the following instructions tells the assembler how to control the operation of the object program:

DMP	Dumps the symbol table.
EMD	Terminates a macro-definition (MAC).
END	Terminates assembly.
LDS	Reads symbol table.
NEW	Switches assembler to new format.
OCR	Writes object file label record.
OLD	Switches assembler to old format.
PFX	Prefixes symbols with a character to make them unique.
SBR	Calls subroutine from a library file.
TCD	Transfers program control for execution.

- Peripheral. Unless otherwise instructed, the assembler builds object programs in card format, with an origin address, word count, and hash total on each card. However high-speed storage peripherals are available and are instructed by the following pseudo-instructions:

DLT	Deletes information in updating file procedures.
DSA	Addresses a DSU.
WOD	Output of the object program is generated in 64-word records with control words.

Alphabetical Listing

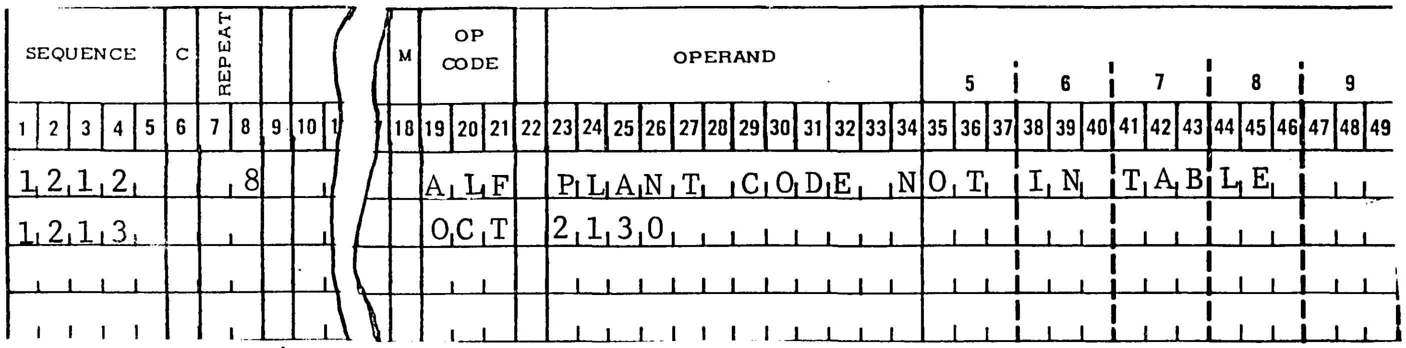
ALPHANUMERIC

ALF

The ALF instruction causes alphanumeric constants of a message to be entered into the object program. The characters in the message are converted to BCD and placed in memory locations determined by the assembly program.

The message must start in column 23 of the Operand field and may continue as far as necessary, through and including column 79. This provides 57 columns (19 three-character words) of message on one card. The number of three-character words in the message must be indicated in the Repeat field. If the Repeat field is blank, the assembler assumes that the message is only one 3-character word.

Example: The message PLANT CODE NOT IN TABLE may be entered in the object or assembled program by using the ALF instruction.



SEQUENCE (1-5)	C (6)	REPEAT (7-8)	M (18)	OP CODE (19-21)	OPERAND (23-)
1212		8		ALF	PLANT CODE NOT IN TABLE
1213				OCT	2130

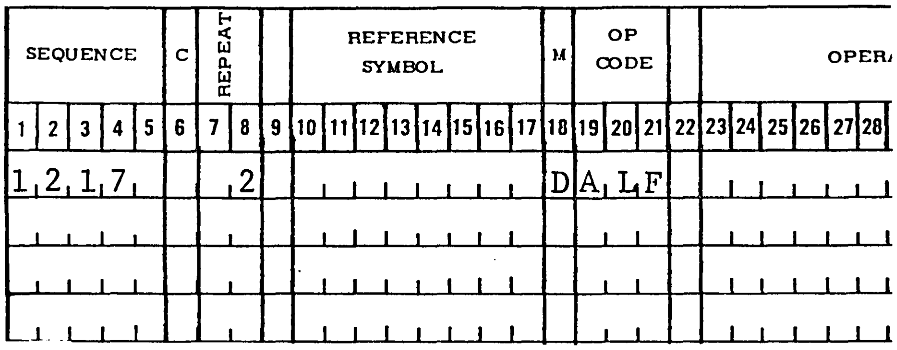
Note that a space is indicated by leaving the column blank.

The example appears in memory as follows:

Memory Location	Sequence	Instruction	Repeat	Opr	Operand
01505	1212	474321	8	ALF	PLANT CODE NOT IN TABLE
01506		456360			
01507		234624			
01510		256045			

Memory Location	Sequence	Instruction	Repeat	Opr	Operand
01511		466360			
01512		314560			
01513		632122			
01514		432560			
01515	1213	002130		OCT	2130

Example: To set information starting at an even-numbered location, use the Modifier field. Set six blanks in memory.



SEQUENCE	C	REPEAT		REFERENCE SYMBOL	M	OP CODE		OPER
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
1217		2			D	ALF		

If the next available memory location is 01505, the example appears in memory as follows:

Memory Location	Sequence	Instruction	Repeat	Mod.	Opr	Operand
01506	1217	606060	2	D	ALF	
01507		606060				

ADD PREVIOUS OPERAND

APO

The APO instruction is used to create tables which have a starting value and are to be incremented by the values following the APO instruction. An SAP must be used at the end of the table to stop adding previous operands.

Example: A table of DSU addresses is to start at 076000 and be incremented by 20, 40, and 20.

SEQUENCE	C	REPEAT		REFERENCE SYMBOL	M	OP CODE		OPERAND
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
100						OCT		76000
110						APO		
120						OCT		20
130						OCT		40
140						OCT		20
150						SAP		

Line 110	Add previous operand (76000)
Line 120	Generates 76020
Line 130	Generates 76060
Line 140	Generates 76100
Line 150	Stop

BLOCK STARTED BY SYMBOL

BSS

A BSS instruction causes the assembly program to increase the memory allocation register (MAR) by the number in the Operand field. This instruction is used to reserve a block of memory locations in the object program. The Operand field may be decimal or symbolic. If symbolic, the symbol must be predefined: if decimal, the operand is converted to binary by the assembly program before use. The BSS instruction can be used as often as needed. A negative decimal operand can be used to reduce the MAR, in effect equating a block of memory to another block already defined.

Example:

SEQUENCE	C	REPEAT		REFERENCE SYMBOL	M	OP CODE		OPERAND
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
422						ORG		144
423				CRD		BSS		28
424				PRINT		BSS		40
425				INDEX		BSS		-3

Line 422	Beginning location. MAR set of 144_{10}.
Line 423	MAR increased by 28. CRD is equated to 144_{10}, or 220_8.
Line 424	MAR increased by 40. PRINT is equated to 172_{10}, or 254_8
Line 425	MAR decreased by 3. INDEX is equated to 212_{10}, or 324_8
	MAR is set at 209_{10}, or 321_8.

A D in the Modifier field forces MAR to start the block at the next even-numbered location.

Example:

SEQUENCE	C	REPEAT		REFERENCE SYMBOL	M	OP CODE		OPERAND
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
						ORG		1024
				CRD		BSS		27
				PRINT	D	BSS		30

MAR would reserve the first 57 words, 2000_8 to 2032_8, for CRD; and PRINT would normally start at location 2033_8. But, because D is in the Modifier field, the assembler would force the starting address to the first even-numbered location, 2034_8, increasing MAR by 58.

Two operands can be used with the BSS instruction when an M is used in the Modifier field. The assembler selects the operand with the largest value.

Example:

SEQUENCE	C	REPEAT		REFERENCE SYMBOL	M	OP CODE		OPERAND
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
					M	BSS		NOW,LATER

NOW and LATER having previously been defined as 4 and 10, respectively, the assembler reserves 10 words.

CONDITIONAL CONSTANT

CDC

The CDC instruction is used to generate constants or words in a constant table. It constructs words from conditions stated in associated CDN instructions. The operand can be octal and/or symbolic. CDN and CDC are used to construct tables in which the value assumed by each element of the table is dependent on certain conditions. These conditions are given names and are defined with a CDN instruction.

A CDC instruction is similar to an OCT, with the following exceptions:

1. All symbols used should be defined by a CDN instruction.
2. All constant elements are octal.
3. All elements are ORed together.

Special tables can be packed to contain only a few conditions (perhaps only two), each item defined by a CDN instruction. A word or constant is then generated by a related CDC instruction. For example, assume that a tape input/output routine requires a word in the file parameter list as follows:

Bits 1-3	Tape unit number
4	Rewind on first call
5	Wait on rewind
6	Wait for end-of-file indication at close
7	Rewind at close
8	Input file
9	Binary tape

The tape routine would contain the following CDN and CDC instructions.

SEQUENCE	C	REPEAT	REFERENCE SYMBOL	M	OP CODE	OPERAND	REMARKS OR CONTINUATION OF OPERAND
			RWDOPEN		CDN	10	BIT 4 REWIND ON FIRST CALL
			WAITOPEN		CDN	20	BIT 5 WAIT FOR REWINDING TAPES ON OPEN
			WEFCLOSE		CDN	40	BIT 6 WAIT EOF AT CLOSE
			RWDCLOSE		CDN	100	BIT 7 REWIND AT CLOSE
			READTAPE		CDN	200	BIT 8 THIS IS AN INPUT FILE
			BINARY		CDN	400	BIT 9 BINARY TAPE
					CDC	6+READTAPE+B	INARY+RWDOPEN+RWDCLOSE

The CDC instruction for the file parameter list specifies to read a binary tape on unit 6, rewind on first call, and rewind on close. The symbol on the CDN instruction is entered into the assembler symbol table along with the bit configuration described and tells the assembler what bits to turn on when the symbol is called for later in some other operation. The CDC instruction generates words or constants for the table

Sometimes certain conditions in a table can be combined under still another name (symbol) to reduce repetition. For instance, in the above example assume that several words in the file parameter list require a READTAPE+BINARY configuration. Rather than repeat READTAPE+BINARY over and over, this bit combination can be lumped together under still another symbol to reduce the repetition. A CDN instruction would read as follows:

Symbol	Opr	Operand	
COMBO	CDN	600	(Combined 400 and 200)

A CDN is not restricted to use with just YES/NO (single-bit) conditions. For example, assume that bits 3-1 are to be used to indicate color. CDN could be used as follows:

Symbol	Opr	Operand
RED	CDN	0
GREEN	CDN	1
BLUE	CDN	2
YELLOW	CDN	3
PURPLE	CDN	4
BLACK	CDN	5

It is not really necessary to define a condition which represents zeros in the field; however, it is possible and assists in documentation. In the example, if no colors were specified, the result would be RED, so it is not necessary to define RED; but calling for RED in an operand gives more positive documentation.

CONDITION DEFINITION

CDN

The CDN instruction is used to construct a special kind of packed table, one in which each item can represent one of only a few conditions. It is used with the conditional constant (CDC) instruction to describe the bit configuration to be presented by the name (symbol). The instruction does not generate data but tells the assembly what bits to turn on when the symbol is called by some other pseudo-instruction later in the program. The operand must be octal, and the instruction must have a name in the symbol field.

Examples for the use of the CDN instruction are included in those for CDC.

DECIMAL

DEC

The DEC instruction places the binary equivalent of a decimal constant in the object program. The constant is assigned a memory location determined by the assembly program. The operand portion of the constant can be symbolic or decimal. If symbolic, at least one character must be used other than 0-9, plus, or minus.

If no sign is present, the number is assumed to be positive. A minus sign, specifying a negative number, results in the 2's complement of the number being placed in memory. Leading zeros are ignored and the number right justified.

Example(positive numbers):

16 17	M 18	OP CODE 19 20 21	22	OPERAND 23–34	APPEARS IN MEMORY 35–43 (5 | 6 | 7)
		DEC		5	000 005
		DEC		73738	220 012

Example (negative numbers):

16 17	M 18	OP CODE 19 20 21	22	OPERAND 23–34	APPEARS IN MEMORY 35–43 (5 | 6 | 7)
		DEC		-1	777 777
		DEC		-19	777 755
		DEC		-5	777 773
		DEC		-33	777 737
		DEC		-73738	557 766

Example (symbolic notation): A decimal constant can be expressed in the Operand field as a symbol or as sums and/or differences of symbols or numbers. These symbols may represent memory cells or be defined by an EQU or EQO.

SEQUENCE	C	REPEAT		REFERENCE SYMBOL	M	OP CODE		OPERAND
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
						DEC		SYNTX
				SYNTX		EQU		143

Example: Set a 12-word block in memory to 1 (uses the Repeat field option).

C	REPEAT		REFERENCE SYMBOL	M	OP CODE		OPERAND	REMARKS OR CONTINUATION OF OPE								
								APPEARS IN MEMORY							LOCATION	
								5	6	7	8	9	10	11	12	13
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
	11		SYNTX		DEC		1	000	001						114	1
								000	001						114	2
								000	001						114	3
								000	001						114	4
								000	001						115	3

DOUBLE-LENGTH DECIMAL

DDC

DDC, like DEC, is used to enter a decimal constant in the object program. This constant is assigned two sequential memory locations, starting with the first even-numbered location available. The allowable number range on a DDC is -34,359,738,367 to +34,359,738,367.

Example:

	M	OP CODE		OPERAND	1st Word Even Loc. 5	6	7	2nd Word Odd 8	Loc 9	REMARKS OR 10	1
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 5
		DDC		12	000	000			000	014	
		DDC		786432	000	011			000	000	
		DDC		-24	777	777			777	750	

DELETE

DLT

The DLT instruction causes card images to be deleted from master files when updating existing current files. If a DLT is used in any program other than an update procedure, it will be ignored by the assembler.

As an instruction in an update procedure program, a DLT deletes either a single card (record) or a block of cards (records) as identified by their sequence number (first five characters of a record). If only one record is to be deleted, the Operand field is left blank; and the sequence number of the item to be deleted is shown in the Sequence field. If a block of cards is to be deleted, the Operand field must show the sequence number of the last item to be deleted.

Example (for a single-record delete):

Sequence	Symbol	Opr	Operand
1115		DLT	

Example (for a block delete):

Sequence	Symbol	Opr	Operand
1115		DLT	1139

DUMP SYMBOL TABLES

DMP

The DMP instruction causes the assembler to dump every symbol between the parameter addresses shown in the Operand field. The operand may be either decimal or symbolic. If decimal, the operand is converted to binary; if symbolic, the operand must be predefined.

Example: Dump all symbols whose addresses are in the common data bank.

Symbol	Opr	Operand
DUMP#1	DMP	0,512

Example: Dump all symbols whose addresses fall between the addresses assigned to the two symbols HERE and THERE.

Symbol	Opr	Operand
DUMP#2	DMP	HERE,THERE

The DMP instruction does not dump symbols defined by CDN instructions or symbols which have been modified.

All other symbols are dumped (including prefixed symbols) together with their address and control bits. The first six columns of the Symbol field are written on each record of the symbol table file for identification.

DISC STORAGE ADDRESS

DSA

The DSA pseudo-instruction allows the assembler to set a beginning and ending address (range) in the DSU address table to be used in accessing the COIL files during phase 2 of the assembly. (See "COIL Files, p. 96.)

When several files are stored as disc storage records, the assembler must be given the addresses of the files in order to access them when called upon by the source program. The addresses can vary from one assembly to another, and the DSA pseudo-instruction enables the programmer to specify the addresses to be used.

The programmer specifies on the DSA the file number in the Repeat field and the octal code configuration for the beginning and ending frame address (CW3 of the PRF instruction) in the Operand field. The two addresses must be separated by a comma.

Example:

Repeat	Symbol	Mod.	Opr	Operand
3			DSA	104450,176240

More than one assembly file may be required in the assembly, in which case an M in the Modifier field tells the assembler to continue analyzing pairs of addresses (separated by slashes) until it finds a pair not terminated by a slash in the Operand field. Each time a slash is detected, the assembler adds 1 to the file number shown in the Repeat field. If the assembly file numbers are not in consecutive sequence in their assignment of numbers, they cannot be specified on the same DSA instruction but will require separate DSA's.

Example:

Repeat	Mod.	Opr	Operand
3	M	DSA	104450,176240/442006,544264

The addresses shown in the Operand field must show the starting address and the ending address. The two addresses are separated from each other by a comma and/or one or more blanks. Addresses may be either octal or symbolic. If symbolic, they must be previously defined by a CDN command.

If a DSU is used for COIL file 9, it is not necessary to provide addresses with a DSA, because the assembler automatically uses the addresses assigned to COIL file 0.

EJECT PRINTER PAPER

EJT

The EJT instruction causes the printer to slew to the top of the next page after printing the line causing the slew.

Normally, the assembly program prints 54 lines per page and then ejects to the top of the next page. When the EJT instruction is encountered, the line is printed, the line count is reset, and the paper is immediately slewed to the top (first printing line) of the next page.

END OF MACRO DEFINITION

EMD

The EMD instruction causes the assembly program to terminate the macro-definition.

END OF PROGRAM

END

The END instruction causes the assembly program to generate an instruction transferring control to the location specified in the Operand field upon execution of the object program. The operand may be decimal or symbolic. If decimal, the operand is converted to binary; if symbolic, the symbol must be predefined.

The END instruction signifies an end of program and termination of the assembly. This instruction may be used only once in a source program and must be the last instruction to be executed by the program. If no END instruction is used, an error comment results, and the assembler terminates the program by an end-of-deck condition.

EQUALS OCTAL

EQO

The EQO instruction equates a new symbol to a memory location or to an old symbol already known to the assembly program. The operand (octal or symbolic) indicates the specific memory location to be used. The instruction does not affect the MAR; thus, it may be used as often as necessary and at any point within the source program without disturbing the memory assignment sequence. If the operand is symbolic, the symbol must be predefined. Leading zeros in the Operand field are ignored.

Example:

	Symbol	Mod.	Opr	Operand
1.	CRD2		EQO	400
2.	AREA		EQO	PRINT
3.	SAME	M	EQO	00512,START

Line 1	Symbol CRD2 address is assigned to octal location 400.
Line 2	Symbol AREA address is assigned to a predefined location called PRINT.
Line 3	Leading zeros are ignored; symbol SAME address is assigned to octal location 512 or to START, whichever is greater.

Multiple operands may also be used with EQO to select the larger of two operands, if preceded by an M in the Modifier field.

EQUALS DECIMAL

EQU

The EQU instruction is identical with EQO, except that the Operand field must be decimal or symbolic.

Example:

Symbol	Opr	Operand
CRD2	EQU	256

Symbol CRD2 address is assigned to decimal location 256 (octal 400). MAR is not affected.

Multiple operands may also be used with EQU to select the larger of two operands if preceded by an M in the Modifier field.

EVEN LOCATION

EVN

The EVN instruction causes the symbol named in the Operand field to be forced to an even-numbered location in memory and must precede the instruction defining the symbol. The MAR is increased by 1 or 2, dependent upon its status.

Example:

Symbol	Opr	Operand
	EVN	MASTER
MASTER	DEC	0

EVN forces MASTER symbol to be assigned to the next even-numbered location in memory.

FIELD DEFINITION

FDN

The FDN instruction does not generate data; it is used to describe one of the parameters to be placed in a table by an associated MFC instruction. An FDN tells the assembly program how to interpret data on following MFC instructions and where to put that data.

Fields of a table are numbered from 1-31 inclusive, written in the Repeat field on the coding sheet. (The FDN instruction cannot be used to generate repeated instructions such as DEC and OCT.) The parameter-type literals in columns 10-12 tell how the field should be interpreted. The Operand field must contain the number of the leftmost bit position to be occupied by the field (bit 18 on the first coding line in the following example), followed by a comma or a space, followed by the rightmost bit position (bit 14 on the first coding line).

Example: For a table consisting of one word per Teletype station and each station requiring four fields of data, one FDN instruction is made for each field.

C	REPEAT	REFERENCE SYMBOL	M	OP CODE	OPERAND	Defined as
	1	DEC		FDN	18,14	Line number
	2	T5L		FDN	12,7	Call-directing code (CDC)
	3	ALF		FDN	6,1	ID code
	4	SYM		FDN	13,13	Line type (full or half)

INDIRECT ADDRESS INDEXED BY A-REGISTER

INA

The INA instruction is used to generate a constant which is a memory address. When used as an indirect address, the contents of the A-register are added to the address portion of the instruction.

Example:

Memory Location	Instruction	Symbol	Opr	Operand
00177	130000	B1	INA	MEMSIZE-KSYM-KSYM
00200	134000	B2	INA	MEMSIZE-KSYM
00201	100201	XARDFN	INA	*
		MEMSIZE	CDN	40000
		KSYM	EQO	4000

INDIRECT ADDRESS INDEXED BY B-REGISTER

INB

The INB instruction is identical with INA, except that the B-register is used as an index register.

Example:

Memory Location	Instruction	Symbol	Opr	Operand
00411	200411	XBRFDN	INB	*

INDIRECT ADDRESS INDEXED BY C-REGISTER

INC

The INC instruction is identical with INA, except that the C-register is used as an index register.

Example:

Memory Location	Instruction	Symbol	Opr	Operand
00473	300473	XCRFDN	INC	*

INDIRECT ADDRESS

IND

The IND instruction is used to generate a constant that is a memory address. The operand may be a number, a symbol, or a sum-and-difference expression. If numeric, it is assumed to be a decimal number and is converted to binary. If symbolic, the address of the symbol is used.

Memory Address	Instruction	Symbol	Opr	Operand
05304		MNEMONIC		
⁞		⁞		
05547		LASTOP		
⁞				
05600	000243		IND	LASTOP-MNEMONIC
⁞	⁞		⁞	
02234	002234		IND	CHANGE

INHIBIT

INH

The INH instruction inhibits the assembly's freedom to move a symbol. Whenever the assembly finds a symbol in the operand of a double-length instruction (for example, LDD, BRS) the assembly usually forces that symbol to the next available even location. When the programmer does not want the symbol to be placed in an even location, he places that symbol in the Operand field of the INH instruction.

Example:

Symbol	Opr	Operand
PRINT	LOC	1000
	INH	XYZ
	LDD	XYZ
XYZ	LDC	

Symbol XYZ will be at memory location 1001_8, not forced to the next even location.

LOAD SYMBOL TABLE

LDS

The LDS instruction causes the assembler to load every symbol into memory addresses for the parameters shown in the Operand field. The operands may be either numeric or symbolic. If symbolic, the symbol must have been previously defined.

The assembler accepts only those records from the file whose identification matches that on the LDS card. The first six columns of the Symbol field must contain the identification written on each record of the symbol table file (see DMP).

If prefixing is in effect when the LDS card is executed, symbols loaded from the symbol table file are prefixed.

Symbols should not be called from the file if they have been previously referenced. In this case, they will be set up as multidefined symbols on the object program listing.

SUBROUTINE LINK

LNK

The LNK instruction generates two words; the first is zero, and the second is the address specified in the Operand field. The constant is assigned two sequential memory locations, starting with the first even-numbered location available.

Example: A subroutine, whose calling name will be READ, starts at location READ1, (1000_8).

Symbol	Opr	Operand	Appears in Memory	
READ	LNK	READ1	Location	Instruction
			01442	000000
			01443	001000

Unless told to do otherwise, the assembler forces the LNK pseudo-instruction to an even location. Note that a DDC pseudo-instruction would accomplish the same thing.

Example: A subroutine identical to the one above.

Symbol	Opr	Operand
READ	DDC	READ1

LOCATION IN OCTAL

LOC

The LOC instruction performs the same functions as an ORG; however, the contents of the Operand field must be an octal number or be symbolic. The assembly program ignores leading zeros. The MAR is reset in the same way as with an ORG pseudo-instruction.

Example:

Opr	Operand
LOC	1000

Multiple operands may also be used with the LOC instruction to select the larger of two arguments, if preceeded by an M in the Modifier field.

LIST

LST

If the object program has been inhibited from listing by an NLS (No List) instruction, listing can be resumed with an LST instruction.

MACRO PROGRAMMING

MAC

The MAC pseudo-instruction provides macro capability to COMMA and is useful to a programmer where defined macros are needed. The MAC instruction saves the programmer from having to write in-hand codes repetitively. The operands can be provided at definition time or calling time or both. Each MAC instruction requires two basic elements:

1. Defining the macro (definition time)
2. Calling the macro (calling time)

DEFINING A MACRO. There are three basic parts to a macro-definition, as follows:

1. Header: MAC--one card
2. Body: one or more instruction cards
3. End: EMD--one card

All cards following the header are considered to be part of the definition until the EMD card appears.

Header. The Symbol field of a MAC instruction must contain the name by which the programmer will call the macro later. The symbol can be any valid 3-character alphabetic or alphanumeric configuration in columns 10-12, providing it is not the same as any machine instruction; pseudo-instruction; or shift, circulate, or subtract macro-instruction.

Body. Instructions within the body of the definition cannot be labeled with ordinary symbols. They can, however, be redefinable. Using redefinable symbols requires caution, since the symbol must be redefined each time the macro is called.

Macro-definitions can contain any machine instruction and the following pseudo-instructions:

CDC	INC
DEC	IND
DDC	LNK
INA	OCT
INB	

No other pseudo-instructions are permitted, nor are shift, circulate, or subtract macro-instructions.

One MAC cannot call on another MAC.

Each instruction within the definition may contain a complete operand, a partial operand, or no operand:

- o If the complete operand is provided at definition time, the Modifier field should be left blank.
- o If only a partial operand is provided at definition time, the partial operand should be written normally and the Modifier field should show an X.
- o If no operand is provided at definition time, the Modifier field should show an X, and the operand must contain a double asterisk. The double asterisk denotes "blank operand ok."

End. Definition time is terminated by the pseudo-instruction EMD (denoting end-of-macro-definition time). No other parameters are needed on the EMD card.

CALLING A MACRO. A macro is called with the symbol assigned to it in the macro definition. The Operand field must contain an operand for each X shown in the Modifier field during definition time, and each operand must be in the same sequence. Each operand must be terminated by a comma or a blank, depending on whether or not indirect addressing is desired. Operands must be separated by not more than eight blanks.

Example: The following is a subroutine call which can be replaced by a macro:

M	OP CODE		OPERAND	REMARKS OR CONTINUATION OF OPERAND
18	19 20 21	22	23 24 25 26 27 28 29 30 31 32 33 34	35 … 80
	BRS		PACK	
	LNA			STARTING ADDRESS FROM WHICH DATA IS TO BE MOVED
	LNB			STARTING ADDRESS TO WHICH DATA IS TO BE MOVED
	DEC			NUMBER OF WORDS TO BE PACKED

The calling sequence is to be defined as a macro, with parameters t be provided at call time. First, the macro must be defined as follo

REFERENCE SYMBOL (10–17)	M (18)	OP CODE (19–21)	22	OPERAND (23–34)
PCK		MAC		
		BRS		PACK
	X	INA		**
	X	INB		**
	X	DEC		**
		EMD		

Now call on the macro to pack 32 words from INPUT to OUTPUT:

M (18)	OP CODE (19–21)	22	OPERAND (23–34)	5 (35–37)	6 (38–40)
	PCK		INPUT,OUTPUT	,32	

Or 20 words can be moved from CARD+7 to HDLTITLE:

M (18)	OP CODE (19–21)	22	OPERAND (23–34)	5 (35–37)	6 (38–40)	7 (41–42)
	PCK		CARD+7,HDLTI	TLE	,20	

Macro-definitions are stored in the symbol table of the assembler, reducing the total number of symbols permitted. A moderate number of macros does not seriously affect the number of symbols permitted, because the definitions are stored as binary control words and not as card images.

MULTIFIELD CONSTANT

MFC

The MFC instruction permits packing several fields into one word of memory by converting, shifting, and ORing the information shown by its related FDN instruction. The parameters are listed in the Operand field in field number sequence as specified by the FDN. Field 1 must appear as the first parameter, then field 2, etc. The parameters (fields) are separated from each other by commas or up to eight spaces, as desired. The last parameter must be followed by a blank.

Example 1: The FDN instruction previously defined field 1 to be a line number; field 2, a call directing code; field 3, an ID code; and field 4, a line type.

Opr	Operand
MFC	17,C,K,FULL
MFC	3,Y,5,HALF
MFC	21,2,A,HALF

Example 2: To extend the example used above, suppose that each station requires another word containing three parameters--for example, transmitter start code, line number for alternate routing, and CDC for alternate routing. Instead of putting this control word into a separate column, the programmer may desire to put both words for each station together. Thus, two sets of field definitions are needed. The second set can be defined by simply using other field numbers, as shown below.

Field definitions for first word, each station:

Repeat	Symbol	Opr	Operand	
1	DEC	FDN	18,14	Line number
2	T5L	FDN	12,7	CDC, Primary routing
3	ALF	FDN	6,1	Computer ID code
4	SYM	FDN	13,13	Line type

Field definitions for second word, each station:

Repeat	Symbol	Opr	Operand	
5	T5L	FDN	12,7	Transmitter stop code
6	DEC	FDN	5,1	Line number, alternate route
7	T5L	FDN	18,13	CDC, alternate route

The programmer may now use either set of field definitions. Obviously, some way is needed to specify on each MFC card which set of field definitions to use. If the set to be used starts with field 1, no indication is needed; the assembler automatically starts with field 1. If the set to be used starts with some other field number, the programmer must place the number of the first field in the set into the Repeat field (MFC cards cannot be repeated). Below is an extension of example 2, in which every station has two words adjacent to each other in the table:

```
     MFC   17,C,K,FULL        CHICAGO
5    MFC   L,3,U
     MFC   3,Y,S,FULL         NEW YORK
5    MFC   B,13,T
```

Note that there is nothing in the FDN cards which tells the assembler that fields 1-4 constitute one set and fields 5-7 another. It is the number of parameters on the MFC card which determines the size of this set.

The programmer may err and provide a field too big to fit into the number of bits specified on the FDN cards. In the example above, only 5 bits are provided for line number. If the programmer calls for a line number greater than 31 (largest possible in 5 bits) the assembler chops off all high-order bits in the oversize number and flags the word as an error.

An MFC instruction cannot be modified for multiple operands (Modifier field, column 18). If M is used, the result is a bad assembly and the error is not flagged.

MICRO PROGRAMMING

MIC

The MIC instruction is used to set the operation bits of the assembled instruction to any desired configuration. The operands can be decimal or symbolic. The first operand forms a "base" with which the second operand is combined. The first operand may be an octal number or a numeric/symbolic sum-and-difference expression. If it is an expression, the elements are ORed together The first operand must be followed by a space, a comma, or both. A comma in this position does not cause indirect addressing.

The second operand is an address which is calculated as a unit and then ORed together with the base described by the first operand. The second operand may be a decimal number, a symbol, or a sum-and-difference expression. If the second operand is an expression, the elements are added. If the second operand is terminated by a comma, it causes indirect addressing.

After the second operand has been calculated, if the Modifier field contains an A (denoting an absolute address), the address will be in absolute binary. If, however, the Modifier field is blank, the address will be converted to a relative address (program-bank, common-data-bank, or channel table) and will then be ORed with the base provided.

Example: Construct a word containing bits 17 and 18 and the address READ+18.

Opr	Operand
MIC	600000,READ+18

Example: Construct a word containing bits 17 and 18 and the address READ+18 in absolute binary form.

M	Opr	Operand
A	MIC	600000,READ+18

NEGATIVE ALPHANUMERIC

NAL

The NAL instruction is used to enter the 2's complement of an alphanumeric constant in the object program. The characters in the message are converted and placed in memory locations determined by the assembly program. The message must start in column 23 of the Operand field and may continue as far as necessary, through and including column 79. The number of 3-character words in the message must be indicated in the Repeat field. If the Repeat field is blank, the assembly assumes that the message is only one 3-character word.

This instruction is used to generate constants in tables where an ADD command will be used for a three-way compare.

Example: The 2's complement of codes A14, AB2, and ABF are to be placed in the object program.

Repeat	Opr	Operand
3	NAL	A14AB2ABF

NEW FORMAT

NEW

The NEW pseudo-instruction is a format control card and causes the assembler to switch from old format to new. It must be the last card of the old-format group of data cards and be punched NEW in columns 8-10 (Operation Code field of old format). See Chapter 3.

NO LIST

NLS

The NLS instruction stops listing of the object program. To resume listing, an LST instruction must be issued.

OBJECT CONTROL RECORD

OCR

The OCR instruction is used to create identifying label records on the object program (file 7). OCR causes the operand of the instruction to be written on the object program file, assisting the user in locating a particular program in an operating system. There is no limit to the number of labels a programmer can make to a file. Each label, however, must start with a TCD instruction. Both special labels and standard labels can be generated as shown in the following examples.

Example 1 (standard labels): In the following example a standard label is generated for the operand. An S in column 10 of the Symbol field causes the assembler to generate a standard four-word label--three BCD words and the address of the operand in the fourth word.

Symbol	Opr	Operand	Appears in Memory	
	TCD			
S	OCR	PHOENIX	473046	PHO
			254531	ENI
			676060	Xb̸b̸
			003500	Operand address

Example 2 (special labels): To write a special label, the programmer lists consecutive OCR's containing whatever information he desires. An E in column 10 of the Symbol field is required for the last OCR to show the end of the label.

Symbol	Opr	Operand	Appears in Memory	
	TCD	START		
	OCR	707000	707000	
	OCR	002000	002000	Location to go to
	OCR	212223	212223	Label ABC
E	OCR	242526	242526	Label DEF

OCTAL

OCT

The OCT instruction enters octal constants in the object program. The octal number specified in the Operand field may be numeric, symbolic, or a sum-and-difference expression of numerics and/or symbols. Octal constants are used primarily for establishing particular bit configurations in memory. When a numeric operand is used, the number is interpreted by the assembler as octal and must not contain any digits greater than 7. All octal numbers are right justified by the assembler. Leading zeros in the Operand field are ignored and, therefore, need not be supplied. A leading minus in the operand sets the sign bit of the constant to 1.

Example: Set OCT 77 in location READ1.

Symbol	Opr	Operand	Appears in memory
READ1	OCT	77	000077

Example: To set various octal instructions in sequential locations, use the Modifier field to indicate multiple operands.

Sequence	Symbol	Mod.	Opr	Operand	Appears in memory	
					Location	Instruction
1932		M	OCT	17/21/13/200	01001	000017
					01002	000021
					01003	000013
					01004	000200

ODD LOCATION

ODD

The ODD instruction causes the symbol named in the Operand field to be forced to an odd-numbered location in memory and must precede the instruction defining the symbol. The MAR is increased by 1 or 2, dependent upon the status of MAR.

Example:

Symbol	Opr	Operand
	ODD	PRINT
PRINT	DEC	10

ODD forces PRINT to be assigned to next odd-numbered location in memory.

OLD FORMAT

OLD

The OLD pseudo-instruction is a format control card and causes the assembler to switch from new format to old. It must be the first card of an old-format group of data cards and be punched OLD in columns 19-20. (See Chapter 3.)

ORIGIN

ORG

The ORG instruction controls the initial memory assignment performed by the assembly program. When an ORG instruction is encountered, the assembly program uses the contents of the Operand field to reset an internal counter in the assembly program referre to as the "memory allocation register" (MAR). Normally, the MAR is increased by 1 for each instruction encountered.

If no ORG is included, the assembly of the program automatically begins at location 00000. Any number of ORG instructions can be used in one assembly.

If the operand of the ORG instruction is decimal, it is converted to binary by the program before being used. If the operand is symbolic, the symbol must be predefined before being used. A symbol is defined by placing its name in the Reference Symbol field (columns 10-17) once, and only once, in a given program. The assembly ignores all fields but the Operand field on an ORG instruction.

Example:

```
     Symbol     Opr    Operand

1.              ORG    1024
2.   NEXTCARD   TRA    S,Z
3.              BOD    NEXTCARD
4.              LDZ    CHANGEMEM
```

Line 1	Assembly of object program starts at location 2000_8 (1024_{10}). MAR is set to location 2000_8.
Line 2	MAR is increased by 1
Line 3	MAR is increased by 1
Line 4	MAR is increased by 1

An M in the Modifier field instructs the assembler to choose the higher of two addresses. For example, suppose two overlays use the same memory area and both refer to a table which must be kept in memory at all times. The table is to be placed at the end of the longest overlay, but it is not known which overlay is the longest. A name can then be assigned to the last location in each overlay, such as END1 and END2. Now the programmer can origin the table as follows:

Mod.	Opr	Operand
M	ORG	END1+1,END2+1

An asterisk operand can also be used, since one can choose the highest of any number of addresses. For example, see the coding below for a program segment that is to be started at A, B, C, or 4096_{10}, whichever is highest.

Mod.	Opr	Operand
M	ORG	A,B
M	ORG	*,C
M	ORG	*,4096

PREFIX

PFX

The PFX instruction is used to make all symbols within a program segment uniquely different from symbols used in other segments. When the assembler encounters a PFX instruction, it automatically attaches the character shown in column 10 of the Reference Symbol field to the front of all symbols which follow on the coding sheet until it is replaced by another PFX instruction. Prefix characters are not given in these instances:

1. Symbols containing eight characters.
2. Redefinable symbols (#).
3. Symbols in the Reference Symbol or Operand fields which contain an S or O in the Modifier field.

Prefixing may be started, stopped, or changed as often as desired. Prefixing is stopped by any PFX instruction containing either a zero or a blank in column 10.

STOP ADDING PREVIOUS OPERAND

SAP

The SAP pseudo instruction is used with APO to stop adding previous operands.

SUBROUTINE CALL

SBR

Subroutines are called from a library file with an SBR instruction. The Reference Symbol field of the SBR instruction must contain the identification data (label), left-justified, for the subroutine. The symbol for the label must exactly match the label on the library file, character by character; for the assembler does only a double-word compare, with no analysis of characters.

The assembler adds the request for a subroutine to its request list but does not immediately call the subroutine from the library. A maximum of 15 subroutines may be requested at any one time. When the assembler reaches the END instruction of the source program, it calls all requested subroutines which have not yet been called and assembles them at the end of the program.

If it is necessary to call a subroutine before the end of the source program, the immediate-call option (I in the Modifier field) causes the assembler to call all subroutines from the request list table. Calling in subroutines indiscriminately during a program causes the subroutine library file to pass from the beginning every time a request for immediate call is given.

Each time a request exercising the immediate-call option is given, the request list table is cleared; and 15 more subroutines may be requested through the table.

RESET SEQUENCE COUNTER

SEQ

The SEQ instruction causes the sequence counter of the assembler to be reset to zero and the sequence numbers of the source program to be checked. The SEQ pseudo-instruction may be used as often as desired in a source program, and each time it is used the sequence counter is reset to zero. SEQ is most useful when each subroutine of a program contains its own set of sequence numbers.

TELETYPE LEFT-JUSTIFIED

T5L

The T5L instruction stores three characters per word of a Teletype five-level code message, left-justified (with start bit). The number of 3-character words in the message must be indicated in the Repeat field to a limit of 19 words. If the Repeat field is blank, the assembler assumes the message is only one 3-character word.

Example:

C	REPEAT		REFERENCE SYMBOL	M	OP CODE		OPERAND	5	6	7	8	
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
	8				T5L		YOU ARE NOW	ON	INT	ERC	EPT	

TELETYPE RIGHT-JUSTIFIED

T5R

The T5R instruction stores three characters per word of a Teletype five-level code message, right-justified (no start bit). The three high-order bits of the word are zero. The number of 3-character words in the message must be indicated in the Repeat field to a limit of 19 words. If the Repeat field is blank, the assembler assumes the message is only one 3-character word.

Example:

C	REPEAT		REFERENCE SYMBOL	M	OP CODE		OPERAND	5 6 7
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 4
	7				T5R		RESEND MSG N	O SEVEN

TRANSFER CONTROL

TCD

The TCD instruction generates a command which causes a program loader to transfer to the location specified in the Operand field. It is used only when a programmer desires to execute a segment of the program before loading is complete and should be the last instruction of a program segment. When the program is loaded for execution, the TCD directs the assembler to the starting location of the program to be executed. The TCD should not be used in place of an END instruction for the end of a source program, because the assembler looks for additional source program instructions following a TCD.

The operand of a TCD instruction may be decimal or symbolic. If decimal, the operand is converted to binary; if symbolic, the symbol must be predefined. A TCD instruction may be used as often as necessary in a source program, since it does not affect the MAR.

TABLE OF CONTENTS

TOC

The TOC pseudo-instruction provides a means of obtaining a contents listing of sections of the source program. Columns 22-78 should be used for descriptive information of the portion of the program where the TOC is used. This information is printed at the end of the object listing with the page number where the TOC was located in the listing. The contents listing can be detached and placed in front of the assembly listing as a table of contents.

Both a TOC and a TTL pseudo-instruction can be used for each section. They differ in that a TTL card does not appear in the table of contents, but TOC data (columns 22-78) appears at the end of the listing.

The number of TOC cards permitted depends upon the number of symbols used by the program, since the assembler stores the TOC information in the unused space in the symbol table area after all symbols are stored. Any TOC card which cannot be stored is ignored by the assembler.

TITLE

TTL

The TTL instruction causes the information shown in columns 22-78 to be printed at the top of each page. It is not necessary for a title instruction to be the first instruction of a program.

All printed pages of the object program listing are printed with the contents of the title instruction. The contents of the first title instruction are changed only when another title instruction is encountered by the assembler. Any number of title instruction lines may be used in a source program.

WRITE OBJECT PROGRAM

WOD

The WOD instruction causes the assembler to build the object program in 64-word records, with a record origin, word count, and hash total on each record for storage on either a DSU or magnetic tape subsystem.

DETECTED CODING ERRORS

As an aid to the programmer, the assembly program detects certain types of coding errors. Since the source program deck may contain cards punched in both old format (Program Library No. CD225F1.001) and new format (Program Library No. CD030F2.001 the following list of error codes applies to errors in both formats, except where otherwise specified:

A--<u>Error or suspected error in the operand address.</u> One of the following has occured:

1. An Operand field is blank in a coding line normally requiring an address.

2. An entry is in the Operand field of a coding line which normally should be blank.

3. The numeric value of the operand does not meet the requirement of the coding line in which it was used.

In any instance, the value of the operand address is logically ORed into the instruction.

<u>Examples:</u>

```
    Memory   Machine
    Address  Instruct. Sequence  Symbol    Opr    Operand

1.  02000A   010000    1234                ADD
         ↖
          Error code:    The blank Operand field is a
                         possible error. (The address
                         may be added later in the
                         program.)

2.  02331A   000000    1524                EJT     17
         ↖
          Error code:    The Operand field should be
                         blank.

3.  02110A   000000    1324                OCT     28
         ↖
          Error code:    The operand should be octal an
                         not decimal.
```

B--<u>Backed-up allocation register.</u> The assembler MAR was backed up (reset to a lower number than it currently contained) by an ORG or LOC instruction. This may not be an error but is flagged as a potential error.

<u>Example:</u>

Memory Address	Machine Instruct.	Sequence	Symbol	Opr	Operand
04030	100200	01234		BRU	SPREE
02000B	002000	01235		ORG	1024

↖Error code: MAR has been backed up.

C--<u>Control field illegal.</u> The Control field or Modifier field has been used illegally.

E--<u>Message words exceeded.</u> More words were specified on a message coding line than can fit on one card (19 three-character words).

F--<u>Symbol table full.</u> The assembly symbol table was full. The symbol is omitted from the symbol table, having the same effect as an undefined symbol. (See error code U.)

L--<u>Illegal Reference Symbol field.</u> A Reference Symbol field error, such as the use of an illegal character or an all-numeric symbol, has occurred.

M--<u>Multidefined symbol.</u> Either the Reference Symbol field or the Operand field contains a symbolic name which appears in the Reference Symbol field of two or more different instruction coding lines. If the error detected was in the Reference Symbol field, the assembler will assign to that symbol the present setting of the MAR. If the error detected was in the Operand field, the value assigned to the symbol the last time it appeared will be used as the operand address in the assembled instruction.

O--<u>Illegal mnemonic operation.</u> The mnemonic is unknown to the assembler. The program generates a 00 octal (HLT) operation code as a substitute.

S--<u>Size error.</u> An error occurred in the size of DEC, DDC, or OCT; or a number was too large to fit the indicated field size.

T--<u>Error or suspected error in X-field.</u> (Applies to old-forma coding only.) The X-field contains an entry in an instruc tion which does not access memory, or it contains any character other than X or a blank.

U--<u>Undefined symbol.</u> A symbolic name appears in the Operand field which does not appear in the Reference Symbol field of any instruction. The address assigned to this symbol is 0000.

Q--<u>Source card out of sequence.</u> If sequencing of source cards is requested (an optional provision of the object program), this error character indicates the source card is out of sequence.

$--<u>Channel table usage.</u> Either the specified address was not modulo 16 or not less than 8192 or both.

-(minus)--<u>Skipped memory location.</u> COMMA skipped a memory location because something was forced to an even location. This is not necessarily an error but is flagged as a potential error.

/--<u>Assembler error.</u> Indicates a work file record was lost.

P--<u>Peripheral error.</u> An unrecoverable peripheral error (such as parity error) was detected on this or a preceding line. Each peripheral error is flagged on only one line of the listing, even though the error may pertain to a record containing data for more than one line (such as a blocked record or the object program, which contains several instructions). It is not possible to tell where the error occurred--that is, whether on the source file, on the work file, or elsewhere.

ADDRESSING LOCATIONS IN MEMORY

COMMA provides the facility for assigning symbolic or binary addresses to locations in memory. The location of an instruction in memory can be designated by a reference symbol. An instruction with a symbolic address does not have a definite location in memory until assembly of the object program; however, an instruction does have a relative location designated by the reference symbol. Addresses can be referenced to some starting point or be indexed (modified by the contents of an index register) in any of the ways discussed in the following paragraphs.

Direct Address

A direct address indicates the location where the referenced operand can be found or stored within a program bank.

Example:

Memory	Instruction	Sequence	Symbol	Opr	Operand
00200		00374	AMT	EQU	128

(This instruction equates the symbol AMT to location 128_{10}.)

Memory	Instruction	Sequence	Symbol	Opr	Operand
01472	400200	01030		LDA	AMT

(This instruction assigns the operand the value of the symbol address--location 128_{10}.)

Relative Address

A relative address indicates a value which must be added to the base address to find the referenced address.

Example:

Memory	Instruction	Sequence	Symbol	Opr	Operand
00200		00374	AMT	EQU	128
01472	400205	01030		LDA	AMT+5

(This instruction equates the operand to the symbol address plus 5.)

Indirect Address

The DATANET-30 provides indirect addressing. This mode of addressing refers an instruction to another memory location for its final reference address.

Example:

Memory	Instruction	Sequence	Symbol	Opr	Operand
00200		00374	AMT	EQU	128
01074	345472	01421		ADO	AMTA,
(This instruction refers the DATANET-30 to location 1472 for its operand.)					
01472	000200	01930	AMTA	IND	AMT
(This instruction furnishes its operand (AMT) to replace AMTA in LDA AMTA.)					

The machine instruction at memory location 1074 would normally read 341472; but, because the symbol in the operand is followed by a comma, indirect addressing is generated and bit position 12 is set to 1. When the instruction is executed, a 1 will be added (ADO) to the contents at memory location 200 in the common data bank.

Indexed Address

An indexed address is an indirect address in which the contents of a register are added to the reference address to obtain the final reference address.

Example:

Memory	Instruction	Sequence	Symbol	Opr	Operand
00200		00374	AMT	EQU	128
01074	345472	01421		ADO	AMTA,
(This instruction refers the DATANET-30 to location 1472 for its operand.)					
01472	100200	01930	AMTA	INA	AMT
(This instruction furnishes its operand (AMT) plus the contents of register A to replace AMTA in LDA AMTA.)					

The current contents of register A is added to the base address AMT (00200) to obtain the final reference address. If register A contains 120, then the final reference address will be 200 + 120, or 320.

Note that the only difference in indirect addressing and indexed addressing is in the reference to IND, INA, INB, or INC.

ADDRESS MODIFICATION

Instructions assigned to common data or channel tables (program bank 1) can be addressed either in a direct or indirect mode from any location in memory. However, those instructions assigned to program banks 2 through 8 can only be addressed from another location in the same program bank or indirectly from one program bank to another.

Addressing Common Data

All instructions that refer to an address in the common data bank are assigned mode 2 by COMMA, as a part of the instruction address

Example: Load the A-register with contents of TAX stored in location 20_{10}.

Machine Instruction	Opr	Operand
402024	LDA	TAX

Locations in the common data bank 0000 to 0511_{10} are addressed as 2000_8 to 2777_8.

Addressing Channel Tables

Symbols starting with a $ sign denote channel tables and are assigned mode 3 by COMMA, as a part of the instruction address.

Example: Load A-register with scan word 1 stored in location 32_{10}.

Machine Instruction	Opr	Operand
403002	LDA	$SW3

Addressing Program Banks

All instructions that refer to an address within the 1024 memory locations of a program bank are assigned mode 0 by COMMA as a part of the instruction address.

Example: Originate program at location 1024_{10}; reference to constant stored at location 1159_{10}.

```
Location    Instruction    Opr    Operand

            002000         ORG    1024
02000       400207         LDA    CHNGEMEM
```

Example: Originate program at location 1024_{10}; reference to constant stored at location 3072_{10}.

```
Location    Instruction    Opr    Operand

            002000         ORG    1024
* 02000 A   400000         LDA    FLXPT
```

An error tag A with an address 0000 appears, because FLXPT is stored in a location in another program bank. Each program bank has upper and lower limits in which direct addressing may be used.

Changing Program Banks

When it is necessary to go from one program bank to another, a branch instruction must be given in the indirect mode (mode 4). When a program approachs the upper limit of a program bank, the P-counter contains the address of the first instruction in the next program bank; and, upon execution of the instruction in the last location, the next program bank is addressed. If a branch instruction is in the last location, the program branches to the corresponding address in the next program bank.

Example: The following constants are stored in program banks 2 and 3.

Location	Symbol	
02050	FIRST	(Program bank 2)
03107	SECOND	(Program bank 2)
04670	THIRD	(Program bank 3)
05740	FOURTH	(Program bank 3)

The program is to originate in location 1024_{10} for the following operations.

```
    Location     Instruction   Opr    Operand

                 002000        ORG    1024
    02000        400050        LDA    FIRST
    02001        520107        AAM    SECOND
  * 02002  A     500000        STA    THIRD
  * 02003  A     400000        LDA    FOURTH
```

Errors are tagged A as address error, because they exceed the program bank in which the program originated. The program properly coded would be:

```
    Location     Instruction   Symbol    Opr    Operand

                 002000                  ORG    1024
    02000        004670        #         IND    THIRD
    02001        005740        ##        IND    FOURTH
    02002        400050                  LDA    FIRST
    02003        521107                  AAM    SECOND
    02004        504000                  STA    #,
    02005        404001                  LDA    ##,
```

Instructions # and ## are executed indirectly.

3. ASSEMBLY OPERATIONS

COMMA is a three-phase assembler, operating in conjunction with specialized COIL subprograms related to the peripheral configuration specified by the user. Each COMMA/COIL program routine is described in the DATANET-30 Programming Routines Manual. This chapter describes the operations of only the COMMA assembler.

COMMA CONTROL CARDS

Control cards are used by COMMA to establish the format and optional assembly operations to be performed.

Assembly Control Card

The assembly control card is optional. If it is used, it must be the first card in the source deck. If it appears later in the deck, the assembler will halt.

COIL may require that some other card--such as a DSA--be the first card of the source deck. If so, all control information required on the control card is placed in columns not used by the DSA. In this fashion the control card information may be placed on the same card with other pseudo-operations. Control information is specified as follows:

Column	Set to	Operation to be performed
6	C	Identifies the card as a control record.
10	U	Old master file will be updated and a new master file created.
	S	Old master file will be updated and a new master file created and resequenced.
	R	Resequencing during initial assembly.

Columns 12-17 are available for a date, which is printed on the title line of every page of the object program listing. If a date is used, the assembler inserts a slash between the second and third characters and between the fourth and fifth characters. Thus, 070464 becomes 07/04/64.

OLD Format Control Card

COMMA assumes the source program to be in new format. However, old format can be used. (See "Program Form" in DATANET-30 General Assembly Program, CPB-1180.) This feature allows source programs written and assembled on a GE-200 Series central processor to be included in the source file, provided the format is identified by a control card.

All source media data cards following the OLD format control card are interpreted by COMMA according to the pseudo-instructions provided by the DATANET-30 General Assembly Program. Any data coded with a NEW format pseudo-instruction are flagged on the object program listing with an 0, denoting an illegal mnemonic operation.

The mnemonic operation code OLD must be punched in columns 19-21, and any remarks that appear in columns 35-80 are printed on the object listing preceding the group of instructions in old format. All of the pseudo-instructions in the old format work in COMMA, except as follows:

- *+* prints before it ejects, rather than after.
- Double decimal numbers which extend onto a second card do not assemble properly in COMMA. Double decimal numbers exceeding 8 digits must be entered in new format.
- COMMA does not scale decimal or double numbers.
- The Z-option has been replaced by the MIC (new format) instruction.

NEW Format Control Card

The source program can switch back and forth between old and new format as often as necessary, provided the assembler is advised of the switch. In order to switch from old to new format, the assembler is advised by a NEW format control card following the old format group of instructions so that it procedes the first instruction in new format. The mnemonic operation code NEW must be punched in columns 8-10, and any remarks must start in column 31.

SYSTEM CONFIGURATION

COMMA operates with an 8k memory. However, it adjusts itself to a 16k memory.

INITIAL ASSEMBLY

During an initial assembly COIL, COMMA, and the source program are processed to obtain a work file and an object program file. Each instruction of the source media is written in the source card image on the work file and sequence-numbered as directed by the user's source program. COMMA assumes all source media to be in new format unless specifically instructed by an OLD format control card. COMMA operates as follows:

1. A loader routine is used to read COIL initially into memory. After COIL is read, the loader transfers control to COIL.

2. COIL reads phase 1 (analyzer) of COMMA.

3. After phase 1 has been read, COIL turns control over to COMMA.

4. COMMA requests COIL to read the first source program card. COIL then formats and stores the card image in a buffer area shared by both COMMA and COIL and returns control to COMMA.

5. COMMA analyzes the first source data image to determine the following:

   <u>If the first card is an assembly control card</u>, COMMA sets flags for the option specified.

   <u>If the first card is not an assembly control card</u>, COMMA assumes an initial assembly for new format with no updating or resequencing.

6. COMMA assembles the record for COIL and transfers control to COIL to write the record on the master work file.

7. COMMA analyzes and assembles each source record and transfers control to COIL for each instruction to be written onto the master work file.

8. When COMMA detects an END instruction and after it has been written by COIL, COMMA transfers control back to COIL to read phase 2 (object).

9. After phase 2 is read into memory, COIL transfers control to COMMA to execute phase 2.

10. During phase 2, COMMA requests COIL to read each record from the master work file. After assembling each record into a machine-coded instruction for COIL, COMMA requests COIL to write the object program file.

11. COIL reads phase 3 (reports), and COMMA sorts the symbol table for COIL to print or store on a peripheral.

UPDATE ASSEMBLY

During an update assembly COIL, COMMA, and a source program are processed to obtain a new master work file, with or without new sequence numbers, and an updated object program. An update assembly provides the user with the ability to change, delete, or add to previous assemblies, thus eliminating reassembly time. The sequence number of the instruction (record) on the master work file is used as the key for updating and creating a new master work file. Figure 4 shows the required sequence of an update source deck.

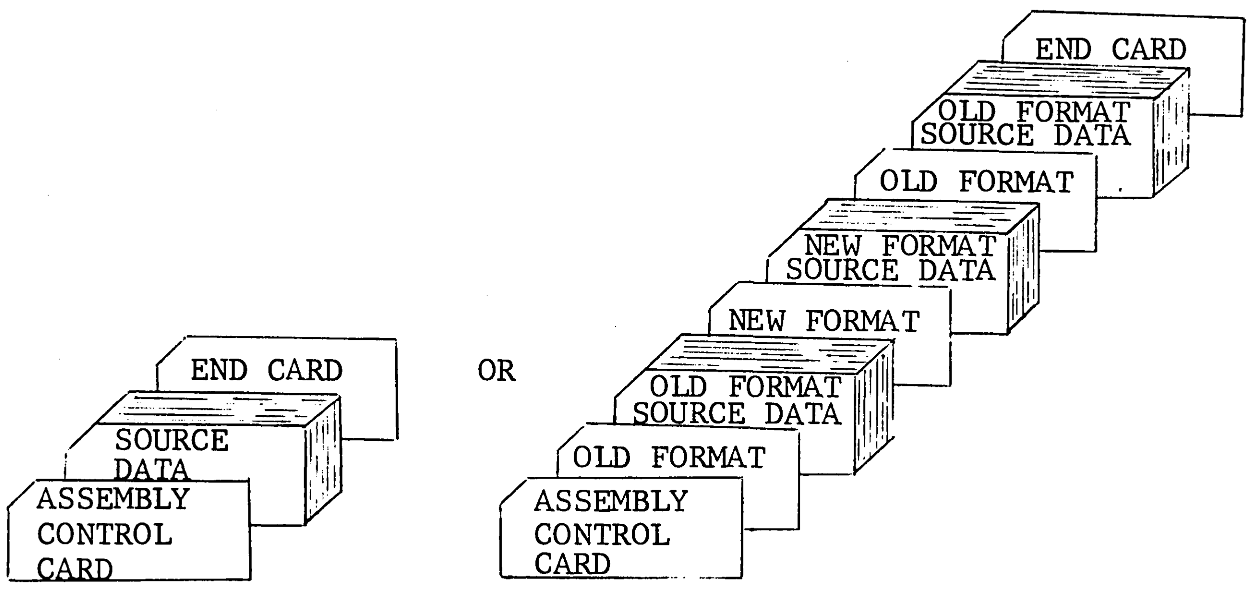


Figure 4. Sequence of Update Source Deck

An assembly control card must be the first card of the source update deck. The source program deck can consist of instructions to add new instructions to the object program, to delete instructions from the object program, or to change existing instructions in the object program. The deck must terminate with an END card.

During an update assembly, NEW and OLD format cards are not required, because COMMA determines the format of the card and processes it correctly.

The old master work file will already have an END card (from a previous assembly). However, of the two, the one with the lower sequence number will terminate the assembly.

A simplified flow diagram of the update procedure is shown in Figure 5.

Changing

Replacing an existing record in the work file with a new one requires only a new record with the sequence number of the old instruction. For example, an old record with sequence number 1115 can be corrected as follows:

	Sequence	Opr	Operand
Old record	1115	CDC	2001+REOPEN+INFILE
New record	1115	CDC	2002+RWDOPEN+WAIT+INFILE

Zeros as sequence number are not valid to records being changed.

Deleting

To remove an existing record from a work file, a Delete (DLT) instruction containing the sequence number of the old record is used. For example, to delete a single record with sequence number 1115, the operand of the DLT must be blank, as shown below.

Sequence	Opr	Operand
1115	DLT	

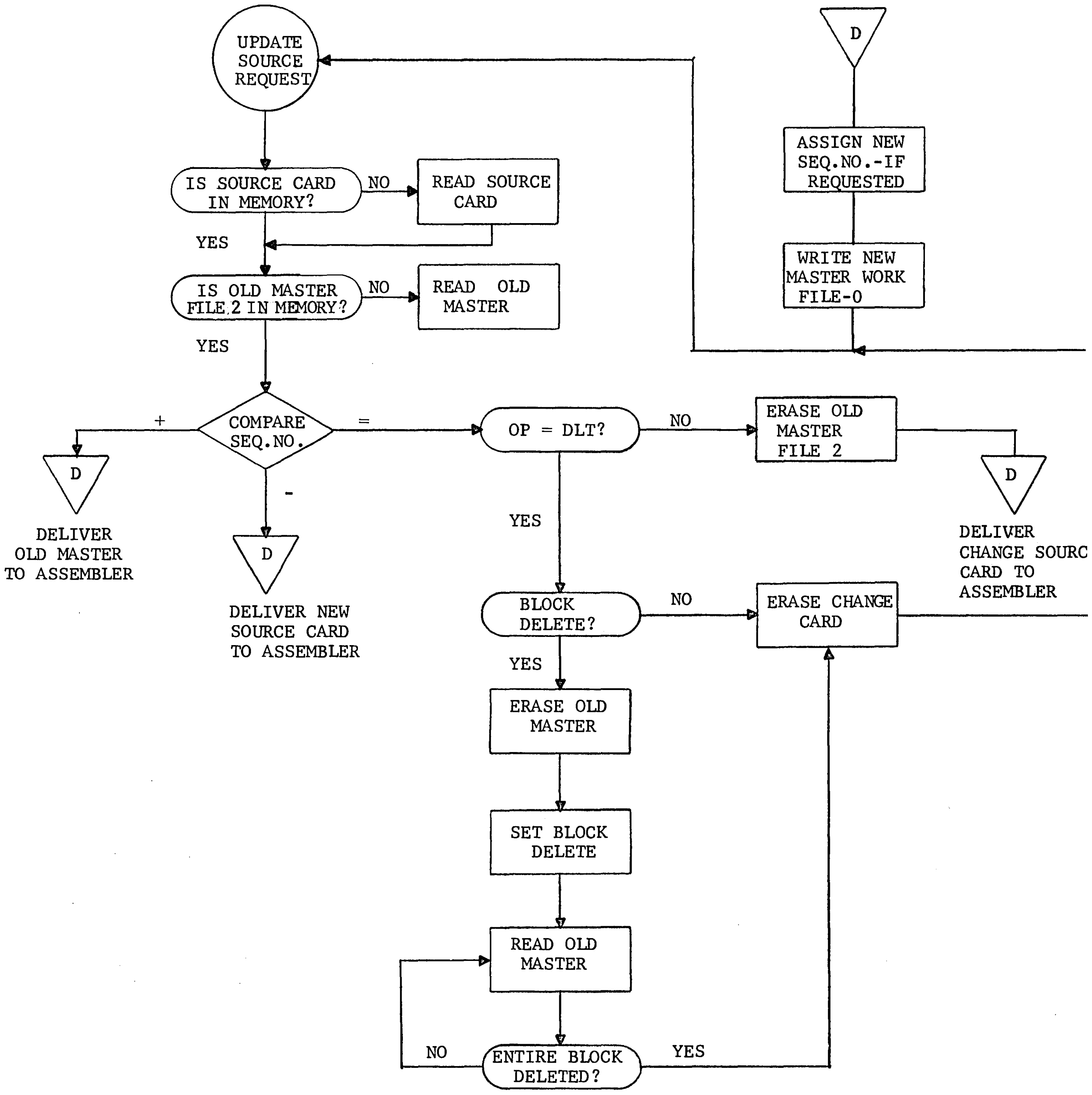


Figure 5. Update Procedure Flow Diagram

If more than one or an entire block of records is to be deleted, it is only necessary to make one DLT and show the ending sequence number in the Operand field. For example, the coding to delete a block of instructions beginning at 1115 and ending at 1139 is as follows:

Sequence	Opr	Operand
1115	DLT	1139

Subroutines cannot be deleted by deleting only the SBR instruction. An SBR pseudo-instruction is converted to "remarks" by the assembler and appears in the work file as a message for the object program listing. To delete a subroutine or group of subroutines, a DLT pseudo-instruction must be used to delete the block of sequence numbers.

Adding

When new records are to be inserted in the work file, the new record must contain a sequence number greater than the last record before the insertion and less than the first record after the insertion. The assembler, when it assigns sequence numbers to the work file, assigns only the first four digits divisible by ten, so there is always an opportunity to assign sequence numbers for insertion.

There is no limit to the number of records that can be inserted between two records, because any number of new records can have the same sequence number. For instance, if fifty records are required, all fifty records can have the same sequence number. The assembler compares sequence numbers during the translator phase; therefore, as long as every new record sequence number is less than that of the next old record, the new records are inserted.

CONSOLE INSERT SWITCHES

The console insert switches provide the operator with the ability to select manually the optional operations to be performed by COMMA and its related COIL subprogram. Each switch represents a bit of a DATANET-30 word affecting COMMA as follows:

Control Switch	Position	Use
18	center	Sign switch. A general-purpose switch used by COIL to recover from a peripheral error. Toggling this switch during an assembly results in program continuation. The console buzzer will buzz, notifying the operator that further assembly operations may result in errors.
17	center	COMMA will ignore D in column 6 (Control field) in all source program cards.
	down	COMMA will delete all source program cards with D in column 6 (Control field).
16	down	COMMA will pause before executing phase 1 or before loading phase 2 of the assembly program. As soon as the assembler is loaded into memory, it will interrogate switcl 16; if it is on, the assembler will wait for it to be turned off before proceeding. When phase 1 is finished, the assembler will once again interrogate switch 16 before calling for the loading of the next phase.
15		This switch is not used by the assembler.

14	down	COMMA pauses before executing phase 2, or before loading phase 3.
13	down	COMMA repeats the execution of phase 2 before loading phase 3, or repeats phase 3.
12	down	COMMA pauses before reading or writing the next record. This enables the operator to stop the assembler without losing data when the assembler is working in a real-time environment.
11	down	COMMA suppresses generation of the object program listing file during phase 2.
10	down	COMMA suppresses generation of the object file during phase 2.
9-1		These switches are assigned specific operations by COIL and are set according to the operations to be performed.

CONSOLE MESSAGES

Messages conveyed by the register contents displayed by lights on the console are as follows.

Display	Contents	Interpretation
A-register	777000_8	Assembly stopped because of error condition shown in the B-register.
	000777_8	Assembly stopped because of condition shown in the B-register.

Display	Contents	Interpretation
B-register	000077_8	Control card present but not first card in source deck.
	007700_8	No DSA record was provided for this file; or A file assigned to a DSU requires more space on the DSA record (card).
	777777_8	End of job.
	any other number	Assembler pausing as requested by setting of insert control switch.
C-register	177_8	Assembler caused the stop.
	any other number	COIL caused the stop according to the displayed file number.

PART II

4. CONSTRUCTING A COIL

The technique of using subprograms for all input and output operations permits a wide variety of peripheral configurations and easy adaptations to various operating systems associated with the DATANET-30. When designing records without knowing the nature of the storage medium, full utilization of the storage capacity of the device probably cannot be achieved; however, this is offset by the flexibility and control made available through the concept of subprograms such as COIL.

A user can write his own COIL, or he can modify an existing COIL program. (The General Electric Computer Department provides specialized COIL programs with the COMMA assembler.)

FUNCTION OF COIL

COIL replaces the routines and subroutines normally required by the main program to read data in from or write data out to peripherals. The work of handling data is divided between COIL and COMMA. All work dependent upon the input/output device is assigned to COIL. These functions include:

Reading records
Writing records
Error-checking and associated recovery procedures
End-of-tape and end-of-file checking
Labeling (record ID)
Fencing
Blocking and deblocking

All work independent of the input/output devices is assigned to the assembler. These functions include the following:

Formatting records
Packing records
Unpacking records
Modifying and analyzing records, except in those cases where code conversion (such as from Teletype) is required or when space-suppression or other condensing is desired.

LINKAGE BETWEEN COMMA AND COIL

COMMA turns control over to COIL by branching to predetermined memory locations linking to subroutines in COIL. These connections are made through a linkage table which must be provided in COIL at specific locations in the common data bank.

Area Available to COIL

Areas in the common data bank and the channel tables are shared by both COMMA and COIL, and some areas are reserved exclusively for COIL. The locations reserved exclusively for COIL are shown in the shaded areas in Figure 6. Octal locations 2000 to 3777 are also reserved for COIL.

COMMA provides parameters for the read or write operation to COIL in the A, B, and/or C registers. COIL, in turn, accesses one of its input files, delivering a logical record to the assembler at the location specified by one of the parameters; or COIL takes a logical record from a specified location and writes it on one of its own output files.

COIL Files

COMMA assigns specific numbers to each of the files used by COIL. This number serves as a key to address or close a file, log errors, and keep track of chaining or linkage addresses (as when using a DSU). The numbers assigned by COMMA, their symbolic name, linkage address, and number of words are shown in Figure 7.

When constructing a COIL program, the programmer must equate (EQO) the linkage address. A description of each file follows.

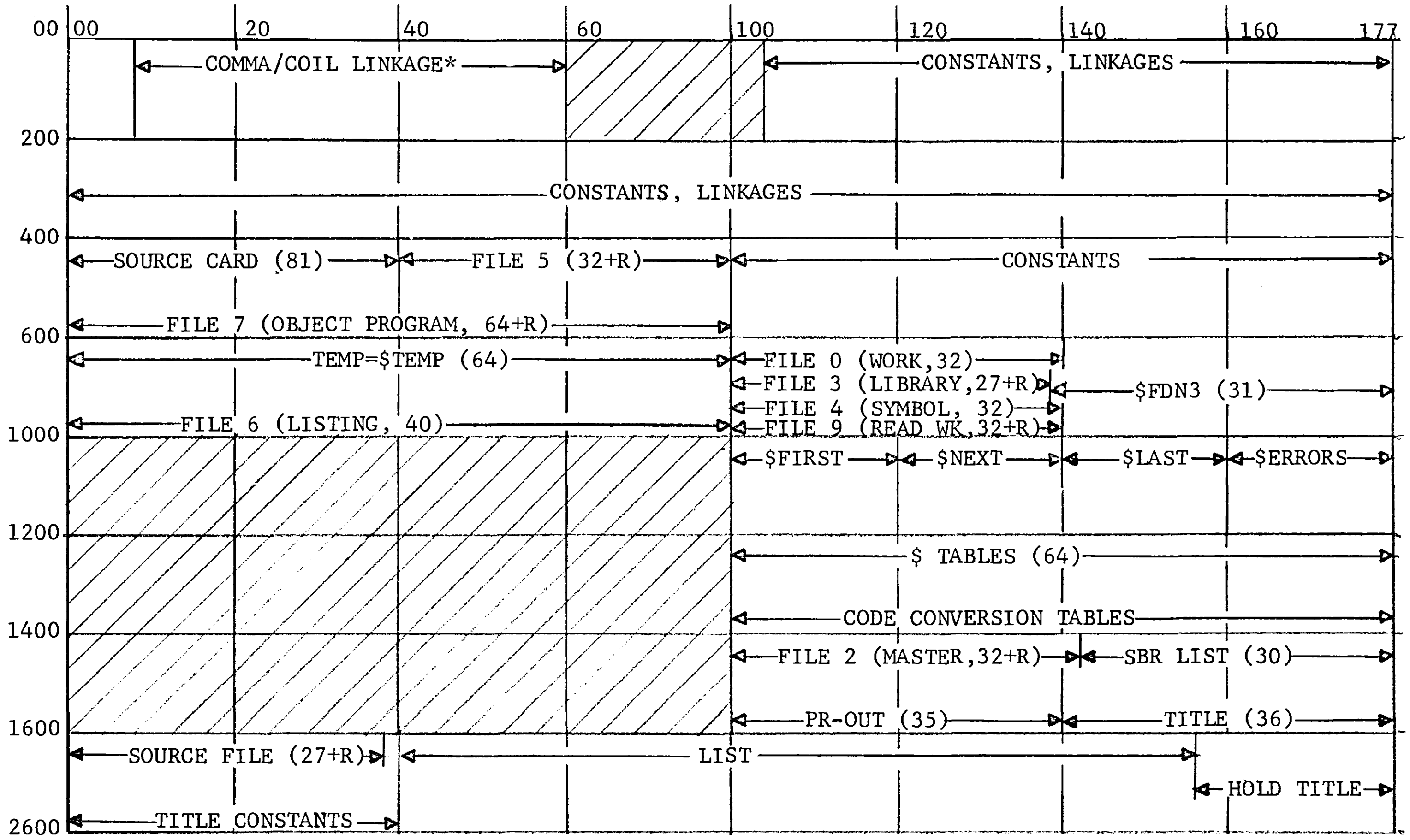


Figure 6. COIL Memory Map

* COMMA/COIL LINKAGE

LOCATION		LOCATION		LOCATION	
10	FILL	30	SUBRTN	50	ENDJOB
12	PAUSE	32	DUMPSYM	52	CUTOFF
14	MOVE	34	READSYM	54	
16		36	PRINT	56	
20	CLOSE	40	OBJECT	134	PPAUSE
22	SOURCE	42	LOAD		
24	WRITE	44	READ		
26	MASTER	46			

NO.	SYMBOL	LINK OCTAL LOC.	TITLE	TYPE	MODE	NO.OF WORDS	I/O AREA
0	WRITE	24	MASTER WORK	OUTPUT	BIN	32	0700
1	SOURCE	22	SOURCE PROG	INPUT	BCD	27	1600
2	MASTER	26	OLD MASTER	INPUT	BIN	32	1500
3	SUBRTN	30	SUBR.LIBRARY	INPUT	BCD	27	0700
4	DUMPSYM	32	DUMP SYMBOL TABLE	OUTPUT	BIN	32	0700
5	READSYM	34	LOAD SYMBOL TABLE	INPUT	BIN	32	0440
6	PRINT	36	OBJECT PROGRAM LISTING	OUTPUT	BCD	V**	0600
7	OBJECT	40	OBJECT PROGRAM	OUTPUT	BIN	V**	0400
8	LOAD	42	LOAD ASSEMBLER	INPUT	BIN	V**	0400
9	READ	44	READ WORK FILE	INPUT	BIN	32	0700

** V = VARIABLE

Figure 7. COIL Files

- <u>File 0 - Master Work.</u> The master work file (WRITE) is created during the translator phase and serves as an input to the assembly phase. It becomes the old master work file (file 2) during updating and reassembly procedures. This is the most frequently used file, since it contains at least one logical record for every card in the source program and can contain one or more overflow records for each instruction generating one or more instructions. It provides an image of each assembled instruction and is used instead of a complete new source program for reassembly, changing or altering an existing assembly, and creating new files. Because it is used frequently, it should be put on one of the high-speed peripherals, such as a DSU or magnetic tape subsystem.

  Each logical record of file 0 contains 32 words. The records are of two types, as follows:

  1. <u>The main record.</u> An image of the source card is in the first 27 words. The last five words contain binary control information.

  2. <u>The overflow record.</u> This record contains 32 words of binary control information.

  The last word of each logical record must contain the following information:

Bit	Set	Means
18	1	Main record.
	0	Overflow record.
17	1	Last record generated by the source card.
	0	One or more overflow records follow this record.
16	1	Old format source card.
	0	New format source card.
15-1	Five-digit sequence number assigned to each record by the assembler.	

  Since each logical record contains 32 words, two logical records will completely fill a DSU record, leaving no space for a chaining address. COIL can use some of the last 15 bits of the last word if chaining is necessary, providing the sequence number is replaced before the work file is reread. The sequence number is used each time the file is read, serving as a protection against lost records.

- <u>File 1 - Source Program.</u> The source program file (SOURCE) is the source program written by the user. The file can come from any peripheral as long as each instruction is in 80-column BCD format. The file is used as an input to the translator phase.

  The source program file is not one of the larger files; however, it is of fairly high volume. If buffering and/or blocking is feasible with the peripheral device used or if adequate memory is available, the programmer may consider having COIL include buffering or blocking. The file consists only of changes when file 0 needs to be updated; therefore, the speed of the input peripheral need not be given much consideration by the programmer.

  As a safety factor, COIL operations should detect an end-of-file condition to protect the program if the source program is not followed by an END instruction. In such cases, COIL flags the end-of-file condition to the assembler, and the assembler terminates the first pass and continues with subsequent passes. If COIL does not detect an end-of-file condition, the program halts in a "not-ready" condition.

- <u>File 2 - Old Master.</u> The old master work file (MASTER) is simply the master work file (file 0) from a previous assembly. It is used as one of the inputs for a reassembly or an update routine.

- <u>File 3 - Subroutine Library.</u> The subroutine library file (SUBRTN) is a multifile group of routines and subroutines, each recorded as physical records on either magnetic tape or the DSU. It can be a high-volume and high-frequency file, depending upon the programs being processed by the DATANET-30 system.

  The subroutine library file can be recorded in any mode (binary or BCD) and can be blocked and/or buffered according to the amount of memory available. Each physical record (routine or subroutine) must be identified with an information data (ID) label as its first logical record and must terminate with an END record.

The ID label must contain the following information:

Col 1-3	+-+
Col 4-6	Blank if the subroutine is in new format. If the record is in old format, it must read OLD.
Col 7-12	Any six-character code for the ID configuration.

The last logical record must contain the following:

Col 1-3	+-+
Col 4-6	END

Subroutines are called from file 3 with the Subroutine Call (SBR) instruction from the assembler's request list table. The subroutines can be called intermittently (with I in the Modifier field, exercising the immediate-call option) or at the end of the source program.

When subroutines are called, the assembler counts the number of subroutines found. If not all subroutines have been found upon reaching the end-of-file condition on the subroutine library, the assembler prints a warning on the listing but does not indicate which subroutines are missing. The subroutines are called from the assembler's request list table in the sequence in which they appear in the subroutine library, regardless of the requested sequence. If the subroutines are to be assembled in a different sequence, the immediate-call option must be used to force calling the later subroutines first.

Once a subroutine is called, it becomes a part of the work file (file 0). Since the work file becomes an input to later assemblies, the assembler converts the SBR and immediate-call option request to a remarks line and does not execute the immediate-call option.

The logical records may be 80-column card images, or they can be truncated to reduce file space. If they are truncated, COIL must blank-fill the unused part of a 32-word record before delivering it to the assembler.

- File 4 - Symbol Table Dump. The symbol table dump file (DUMPSYM) is created as the result of a Dump Symbol Table (DMP) pseudo-instruction. The file is low volume; therefore, little is gained from buffering or blocking.

  The first two words of each logical record contain the identification (symbol) from the DMP instruction. The third word contains the assembly sequence number (columns 1-6) assigned by the assembler. The fourth word contains a word count of the record. The remaining 28 words are divided into seven groups of four words each. Each group of four words contains a symbol, its associated memory address, and its control bits.

  This file is called by other assemblies and is addressed as file 5 (load symbol table).

- File 5 - Load Symbol Table. The load symbol table file (READSYM is used when file 4 is to be reread into memory. A Load Symbol Table (LDS) pseudo-instruction causes the assembler to read into memory the symbol record as shown in the Operand field of the instruction.

- File 6 - Object Listing. The object listing file (PRINT) contains variable-length logical records to record each line of the listing. If the file is to be stored on magnetic tape, both buffering and blocking are recommended. Blocking is worthwhile for saving time on the DATANET-30 processor. However, if the printer is being used, the media-conversion routine will be printer bound; so blocking is of little value.

  The assembler branches to COIL with the parameters in the B and C registers. The B-register contains the number (maximum 39) of words in the line for the listing. COIL uses this number to set whatever end-of-line indicator is needed by the device doing the printing. The C-register contains the slewing control, coded as follows:

0	Print, single-spaced.
1	Print, double-spaced.
2	Slew to top of page, then print, double-spaced.

- <u>File 7 - Object Program.</u> The object program file (OBJECT) can be stored either in card format (81 columns, 40-word logical records) or in DSU format (64-word logical records), depending upon the storage medium chosen. The assembler branches to COIL with the record size in the B-register.

  The object program file is a low-volume file; therefore, it need not be buffered or blocked.

  When the object program file is to be written on a DSU, a WOD instruction is used. Each record has 64 words, with a starting address for the words that follow.

- <u>File 8 - Load Assembler.</u> The load assembler file (LOAD) is a part of COIL and is the loader for the assembly program. The loader must be assigned to the input area assigned to COIL, and it can use any of the buffers used by other files. The loader must not clear memory nor be in upper memory, since COMMA uses upper memory for the symbol tables and must leave the tables intact during the assembly phases.

- <u>File 9 - Read Work File.</u> The read work file (READ) is the work file 0 which was created as an output file during the translator phase. During the assembly phase, file 0 becomes an input (file 9).

  If magnetic tape is used for the file, COIL must include a mark and rewind for the CLOSE routine of file 0 and a CLOSE routine for file 9 to rewind and set a closed flag. On only the first call for file 9, COIL should wait for rewinding, thus eliminating addressing a rewinding tape, even if the file is called upon more than once.

<u>Status Returns</u>

After a read or write operation COIL must return any status condition to COMMA in the A-register.

- Input Status Returns. After a record is delivered to (read into) the assembler, COIL must return with one of the following status codes in the A-register.

Code	Meaning
000000	Normal record, no unrecoverable error.
000xxx	Unrecoverable error. Before returning to the assembler, COIL must make appropriate attempts to reread the record. If COIL determines that it cannot recover from the error, it should place the file number in the C-register, add 1 to $ERRORS table, and then return with the status code in the A-register.
777777	End of file. This status does not deliver a record. It is given only if the assembler calls for a record after the last record has been delivered. Do not confuse an end-of-file with an end-of-reel condition on multireel tape files. If multireel files are used, COIL must determine for itself whether more reels are needed.

The assembler tests the A-register before proceeding with the next operation in the program. The assembler continues and flags the corresponding output record. At the end of the run, the assembler lists the number of errors on each file.

COIL can be constructed to inform the operator of errors. It can load a register with an error code (including file code), turn on the buzzer, and wait for the operator to acknowledge the error (see PAUSE subroutine, p. 107). A message code may be logged if Teletype, typewriter, or printer is available.

- Output Status Returns. After a record is obtained (written) from the assembler, COIL must return with one of the following status codes in the A-register.

Code	Meaning
000000	Normal record, no unrecoverable error.
000xxx	Unrecoverable error. Before returning to the assembler, COIL should place the file number in the C-register and add 1 to $ERRORS. Whether or not assembly should continue after an output error is up to the author of COIL, and the decision should depend upon the file. For example, an error in the object listing, (file 6) can probably be ignored. An error in the object program (file 7) is more serious and may call for rerun.

As in the case of input files, COIL can be constructed to inform the operator of errors and let him take appropriate action.

NONEXISTENT FILES

Some COIL programs will not provide for all allowable files, because either the application on the DATANET-30 does not require their use or the number of peripherals is limited. The assembler, however, will not know that the file does not exist; and, if the program calls for a nonexistent file, the assembler will request a read (or write) of COIL.

COIL must provide linkage for every file. It should place the file number in the C-register, add 1 to $ERRORS, and return to the assembler. For example, if the request is for an input file, the return should be an end-of-file return. If the request is for an output file, there should be a normal return. $ERRORS must be equated (EQO) to 1160_8.

OPENING A FILE

Since COMMA does not call for opening an assembly file, COIL must include all opening operations of its first call. The opening subroutines must include the following:

- o Rewinding or positioning.
- o Wait for rewinding or positioning.
- o Initialization of buffering, when buffering is required.
- o Setting a switch so that future calls on the assembly file will bypass the opening routine.

If magnetic tape subsystems are being used by the system, it is desirable to have COIL rewind all tapes on the first call for file 1 (even if the source program is not on tape) to minimize waiting time for rewinding tape. Files used frequently will probably not need to be rewound the first time they are opened by the opening procedure; however, they should be rewound on closing. The opening routine must include waiting for rewind each time the files are called upon.

CLOSING A FILE

COMMA calls for the closing of every file, both input and output. It may close an input file before an end-of-file record is reached. For example, it can close file 3 (subroutine library) as soon as it has found all requested subroutines.

When a file is on magnetic tape, the closing procedure for COIL should include only rewinding for input files or tape marking and rewinding for output files. Files not on magnetic tape do not require a closing routine.

When there is only one subroutine linkage for closing all files, the assembler enters the CLOSE routine (location 20_8) with the file number in the C-register.

COMMA SUBROUTINES AVAILABLE TO COIL

The assembler contains several subroutines available to COIL. These subroutines do not save registers, but they do save memory space and can be used by COIL whenever necessary. They are as follows:

<u>MOVE</u> The MOVE subroutine (location 14_8) moves a specified number of words from one area to another. It uses the A-register as an index and the B-register for moving. MOVE must be equated (EQO) to 14_8.

Code as follows:

Opr	Operand
BRS	MOVE
INA	Address from which data is to be moved.
INA	Address to which data is to be moved.
DEC	Number of words to be moved.

Example: Move 12 words from temporary to permanent storage.

```
Opr   Operand

BRS   MOVE
INA   TEMP
INA   OUTPERM
DEC   12
```

FIL**L**

FILL The FILL subroutine (location 10_8) fills a specified number of consecutive words with a specified constant. The A-register is used as an index and the B-register for moving. FILL must be equated (EQO) to 10_8.

Code as follows:

Opr	Operand
BRS	FILL
INA	Address of area to be filled.
ALF	Constant to be used.
DEC	Number of words to be filled.

Example: Fill 30 words at output with zeros.

```
BRS   FILL
INA   OUTPUT
ALF   000
DEC   30
```

PAUSE The PAUSE subroutine (location 12_8) waits for console switch 18 to be toggled; it turns off the buzzer but does not turn it on. PAUSE must be equated (EQO) to 12_8.

Code as follows:

```
Opr   Operand

BRS   PAUSE
```

PPAUSE The PPAUSE subroutine (location 134_8) waits before reading or writing the next record when console switch 12 is set. PPAUSE must be equated (EQO) to 134_8.

Code as follows:

Opr	Operand
BRS	PPAUSE

CALLING SUBROUTINES FROM FILE

When subroutines are called from the subroutine library file by the source program (SBR pseudo-instruction), the subroutine address is set in the request table of the assembler for recall at the end of the object program. Each time the request table is accessed to process an address, the file is passed from the beginning; and the subroutines are transferred in the sequence in which they appear in the file, regardless of the sequence of request. The assembler keeps a count of the number of subroutines requested and located. If a subroutine is requested and not found before the end-of-file record, the assembler prints a warning on the object program listing but does not indicate the missing subroutine.

G L O S S A R Y   O F   D A T A C O M M U N I C A T I O N   T E R M S

For readers not familiar with data communications, assemblers, and symbolic programming, the following definitions will be useful.

ALPHANUMERIC	Capable of representing the alphabet as well as the decimal digits 0-9 and (usually) a group of miscellaneous symbols.
BUFFER	An internal portion of a data processing system serving as intermediary storage between two storage or data handling systems with different access times or formats--usually to connect an input or output device with the main or internal high-speed storage.
CHANNEL	(1) A path along which information, particularly a series of digits or characters, may flow. (2) One or more parallel tracks treated as a unit. (3) A path for electrical communication. (4) A band of frequencies used for communication.
CODE CONVERSION	A process for changing the bit groupings for characters in one code into the corresponding character bit groupings for a second code.
COMMAND	(1) An electronic pulse, signal, or set of signals to start, stop, or continue some operation. (2) The portion of an instruction word which specifies the operation to be performed.
DATA COMMUNICATION	The transmission of information to and from data processing equipment. This includes assembly, sequencing, routing, and selection of such information as is generated at independent remote points of data origination and the distribution of the processed information to remote output terminals or other data processing equipment.
ELEMENT	A specific item of information appearing in a set of data.

EXPRESSION	Any symbol representing a variable or a group of symbols representing a group of variables possibly combined by symbols representing operators in accordance with a set of definitions and rules.
FILES	A group of related records pertaining to a captioned subject, stored on one of the storage mediums for rapid retrieval to the DATANET-30 system.
FILES, SCRATCH	A group of records, stored temporarily on one of the storage mediums.
FLAG	An indicator used to tell some later part of a program that some condition occurred earlier.
INPUT/OUTPUT	A general term for the equipment and the data involved in a communication system.
LITERALS	The quantities or messages which will be present in the machine and available as data for the program and which, usually, are not subject to change with time.
LOADER PROGRAM	Synonymous with "loading routine." A routine which, once it is itself in memory (storage), is able to bring other information into storage.
MACHINE INSTRUCTION	An instruction actually used by the computer system to perform an operation.
MACRO-INSTRUCTION	The assembly program recognizes the mnemonics for macro-instructions and automatically generates and inserts into the object program the necessary series of instructions for performing a specific operation.
MICRO-INSTRUCTION	A small, single, short add, shift, or delete type of command.
MNEMONIC INSTRUCTION	A notation used in writing programs to represent the actual machine instruction.
MULTIPROGRAMMING	A technique for handling numerous routines or programs simultaneously by means of an interweaving process.

OBJECT PROGRAM	The result obtained after the source program has been processed by the assembly program to translate the instructions on the coding sheet into machine instruction form.
PSEUDO-INSTRUCTION	A group of characters having the same general form as a machine instruction but never executed by the computer system as an actual instruction. Pseudo-instructions are used to control the assembly process, generate constants, and annotate the object program listing.
PARITY CHECK	A summation check in which the binary digits in a character or word are added, modulo 2, and the sum checked against a single, previously computed parity digit: for example, a check which tests whether the number of 1's in a word is odd or even.
SATELLITE	Off-line computer system operating independently of or in interdependence with a main system.
SOURCE PROGRAM	A program written in a language designed for ease of expression by humans of a class of problems or procedures -- for example, a symbolic or algebraic language.
SOURCE PROGRAM DECK	The results obtained after punching all the instructions on a coding sheet onto punch cards and retained in the order as written on the coding sheet.
START BIT	Synonymous with "start element." The first element of a character in certain serial transmissions, used to permit synchronization.
SYMBOLIC NOTATION	The designation (name) given a location in memory instead of the actual memory location. An instruction to a computer must specify the location of data as well as the operation it is to perform. In symbolic programming, the actual location of a word in memory is always referred to by a name or symbol. The symbol is usually chosen to have a meaning in relation to the memory location.

SYMBOLS	A substitute or representation of characteristics, relationships, or transformations of ideas or things. In the actual writing of a program, the use of symbols to designate memory location and mnemonics to designate the operation allows the programmer to list the desired operations in a more easily understood language.

I N D E X

This index lists subjects alphabetically by subject name and by significant words in subject matter headings and titles of figures and tables.

To locate a subject, look in the index for the subject itself and for significant words related to that subject. The page number for each listed subject appears at the right of the line. When entries are truncated to the left of the significant word, the truncated words appear to the right of the line, following an asterisk.

Subject

Subject

Subject

Subject

Subject