

Systems Reference Library

IBM 1130 Assembler Language

This publication contains the information necessary to write programs in the IBM 1130 Assembler language. Included are rules for statement writing, mnemonic codes and descriptions of operands, and descriptions of the instructions used to control the Assembler program.

PREFACE

This manual describes the IBM 1130 Assembler language and defines the programming rules. It is intended as reference material for the writing of an assembler source program and the accomplishment of the steps required to produce the resulting object program. For those without programming experience or a knowledge of the principles involved, the IBM publication, Introduction to IBM Data Processing Systems (Form F22-6517), is suggested as preliminary reading.

Within this publication, all references to the "Monitor System" apply to Version 1 and Version 2. Where the reference only applies to Version 1, the abbreviation DM1 is used. Where the reference only applies to Version 2, DM2 is used.

The term "loader" as it applies to the 1130 programming systems have the following meanings:

Card/Paper Tape	- Relocating Loader
Disk Monitor 1	- Loader
Disk Monitor 2	- Core Load Builder

For those without experience involving different number systems, i. e., binary and hexadecimal, the publication IBM Student Text: Number Systems (Form C20-1618) is recommended.

Fifth Edition

This edition is a revision of the previous edition (C26-5927-3) which is now obsolete. Information has been added to distinguish between Version 1 and Version 2 of the 1130 Disk Monitor System.

Significant changes or additions to the specifications contained in this publication will be reported in subsequent revisions or Technical Newsletters.

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

A form is provided at the back of this publication for reader's comments. If the form has been removed, comments may be addressed to IBM Nordic Laboratory, Technical Communications Department, Vesslevägen 3, Lidingö, Sweden.

The reader should also be familiar with the following: IBM 1130 Functional Characteristics (Form A26-5881) and IBM 1130 Computing System, Input/Output Units (Form A26-5890).

The assembler language is valid for the 1130 Disk Monitor Programming Systems and the 1130 Card/Paper Tape Programming System. The operating procedures for the Monitor Assembler are described in the publications IBM 1130 Disk Monitor System Reference Manual (Form C26-3750), and IBM 1130 Disk Monitor System, Version 2, Programming and Operator's Guide (Form C26-3717).

The operating procedures for the 1130 Card/Paper Tape Assembler are described in the publication IBM 1130 Card/Paper Tape Programming System Operator's Guide (Form C26-3629).

The library subroutines for the 1130 systems are described in the IBM 1130 Subroutines Library manual, (Form C26-5929).

MACHINE REQUIREMENTS

The minimum machine configuration for assembling programs is as follows:

IBM 1131 Central Processing Unit, Model 1,
with 4096 words of core storage
IBM 1442 Card Read Punch, or IBM 1134 Paper
Tape Reader and IBM 1055 Paper Tape Punch.

CONTENTS

GENERAL DESCRIPTION	1	Data Definition Statements	19
Introduction	1	DC - Define Constant	19
Symbolic Language	1	DEC - Decimal Data	19
Assembler Program	1	XFLC - Extended Real Constant	21
Subroutines	1	EBC - Extended Binary Coded Information	21
Features of the Assembler	1	Storage Allocation Statements	22
SYMBOLIC LANGUAGE	3	BSS - Block Started by Symbol	22
Mnemonic Concept	3	BES - Block Ended by Symbol	22
Format of Statements	3	Symbol Definition Statement	22
Statement Fields	3	EQU - Equate Symbol	22
Statement Writing	7	Linking Statements	23
Character Set	7	ENT - Define Subroutine Entry Point	23
Symbols	7	ISS - Define Interrupt Service Entry Point	23
Location Assignment Counter	8	ILS - Define Interrupt Level Subroutine	24
Relative Addressing	8	CALL - Call Direct Reference Subroutine	24
Self-Defining Values	8	LIBF - Call TV (Transfer Vector) Reference Subroutine	25
Decimal Values	9	MONITOR ASSEMBLER STATEMENTS	26
Hexadecimal Values	9	Disk Data Organization Statements	26
Character Values	9	DSA - Define Sector Address	26
Expressions	9	FILE - Define Disk File (DM2 only)	27
Elements	9	Data Definition Statements	28
Terms	10	DMES - Define Message (DM2 only)	28
Operators	10	DN - Define Name (DM2 only)	29
Evaluation of Expressions	10	Linking Statements	30
Types of Expressions	10	LINK - Load Link Program	30
MACHINE-INSTRUCTION STATEMENTS	13	EXIT - Return to Supervisor	30
Mnemonics	13	DUMP - Dump and Terminate Execution (DM2 only)	30
Condition-Testing Instructions (BSC, BOSC, BSI)	13	PDMP - Dump and Continue Execution (DM2 only)	31
Additional Monitor System Mnemonics (DM2 only)	15	List Control Statements	31
ASSEMBLER INSTRUCTIONS	17	HDNG - Heading	31
Program Control Statements	17	LIST - List Segments of Program (DM2 only)	31
ABS - Assemble Absolute	17	SPAC - Space Listing (DM2 only)	32
LIBR - Transfer Vector Subroutine	18	EJCT - Start New Page (DM2 only)	32
SPR - Standard Precision, EPR - Extended Precision	18	APPENDIX A. CHARACTER CODE SUMMARY	33
ORG - Define Origin	18	APPENDIX B. HEXADECIMAL-DECIMAL CONVERSION	
END - End of Source Program	19	CHART	38
		APPENDIX C. ASSEMBLER MNEMONICS AND ERROR	
		CODES FOR 2250 ORDERS.	39, 1
		INDEX	40

INTRODUCTION

The IBM 1130 Assembler language replaces binary instruction codes with mnemonic symbols and uses labels for other fields of an instruction. Other features, such as pseudo-operations, expand the programming facilities of machine language. Thus, the programmer has available, through an assembler language, all the flexibility and versatility of machine language, plus facilities that greatly reduce machine language programming effort.

Symbolic Language

Symbolic language is the notation used by the programmer to write (code) the program. A program written in symbolic language is called a source program. It consists of systematically arranged mnemonic operation codes, special characters, addresses, and data, which symbolically describe the problem to be solved by the computer.

The use of symbolic language:

- Makes a program independent of absolute core locations, thus allowing programs and subroutines to be relocated and combined as desired.
- Allows subroutines that can be written independently and that cause no loss of efficiency in the final program.
- Permits instructions to be added to or deleted from a source program without the user having to reassign storage addresses.

Assembler Program

The assembler program converts (assembles) a source program into a machine-language program. The conversion usually is one for one — that is, the assembler produces one machine-language instruction for each symbolic-language instruction.

The 1130 Disk Monitor Assembler is a two-pass assembler. The source program is read into core from the principal input device and written on the disk for use in pass 2. During the first pass the symbol table is generated. During the second pass the object

program is created in the system Working Storage and the listing, if requested, is produced.

The IBM 1130 Card/Paper Tape Assembler is a two-pass program. It is loaded into the computer and is followed by the first pass of the source program. During the first pass, the source statements are read and a symbol table is generated. During the second pass, the source program is read again and the object program and/or error indications are punched into the first 20 columns of each source card. If paper tape is used, the second pass results in the punching of a new tape that contains both source statements and corresponding object information. Both card and tape object programs must be compressed (via a Compressor Program supplied with the assembler) into a relocatable binary deck (or tape) before they can be loaded into core storage for execution. The output from the second pass is called the list deck (or tape) and can be used to obtain a program listing of source statements and corresponding object statements.

Subroutines

A library of input/output, arithmetic, and functional subroutines is available for use with the IBM 1130 Assembler.

The user can incorporate any subroutine into his program by simply writing a call statement (CALL or LIBF, whichever is required), referring to the subroutine name. The assembler generates the linkage necessary to provide a path to the subroutine and a return path to the user's program. The ability to use subroutines simplifies programming and reduces the time required to write a program.

A description of available subroutines is contained in the system subroutine library manual.

FEATURES OF THE ASSEMBLER

The significant features of the IBM 1130 Assembler are summarized below. More detailed explanations are given later in this manual.

Mnemonic Operation Codes. Mnemonic operation codes are used for all machine instructions instead

of the more cumbersome internal binary operation codes of the machine. For example, the Subtract instruction can be represented by the mnemonic, S, instead of the machine operation code, 10010.

Symbolic References to Storage Addresses. Instructions, data areas, and other program elements can be referred to by symbolic names or actual machine addresses and designations.

Renaming Symbols. A symbolic name can be equated to another symbol, so that both refer to the same storage location. This makes it possible for the same program item to be referred to by different names in different parts of the program.

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

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

Convenient Data Representation. Constants can be specified as decimal digits, alphabetic characters, hexadecimal digits, and storage addresses. Conversion of the data into the appropriate machine format of the 1130 System is performed by the Assembler. Data can be in a form suitable for use in decimal integer, fixed-point, or real arithmetic operations.

Program Listings. For every assembly, the user can obtain a program listing. This listing can be produced either off-line (Card/Paper Tape Assembler) or on-line during the assembly process (Disk Monitor Assembler).

Error Checking. Source programs are examined by the Assembler for errors arising from incorrect use of the language. Where an error is detected, a coded warning message appears in the program listing.

MNEMONIC CONCEPT

Symbolic programming may be defined as a method whereby names and symbols are used to write a program. The symbolic language includes a standard set of mnemonic operation codes. Mnemonic operation codes are easier to remember than machine language codes because they are usually abbreviations for actual instruction descriptions. For example:

<u>Description</u>	<u>Mnemonic</u>
Add	A
Execute I/O	XIO

Each IBM 1130 machine instruction has a corresponding mnemonic operation code. In addition, there are some mnemonic codes that assign storage and others that allow the user to exercise control over the assembly process.

FORMAT OF STATEMENTS

A source program consists of a sequence of statements. These statements can be written on a standard coding form (X26-5994) provided by IBM. The information on each line of the form (Figure 1) is punched into one card or paper tape record or entered from the keyboard. The first position on the form (21) corresponds to card column 21 or to the first character of the paper tape/keyboard record. Space is provided at the top of the coding form to identify the program; however, none of this information is punched into the statement cards. The first 20 columns of an assembler source card must be blank.

NOTE: Keyboard input is acceptable only with the Monitor 2 Programming System.

Statement Fields

An assembler statement is composed of one to seven fields: label field, operation field, format field, tag field, operand field, comments field, and identification sequence field.

Label Field (Columns 21-25)

The label field represents the machine location of either data or instructions. The field may be left blank, may contain an asterisk in column 21, or may be filled with a symbolic address, left-justified in the field. Only data or instructions that are referred to elsewhere in the program need a label, although a label that is not further referred to is not an error.

A label can consist of up to five alphanumeric characters, beginning at the leftmost position of the label field. A label is always a symbol and must therefore conform to the rules for symbols (see Symbols). The example below shows the symbol ALPHA used as a label.

Label					Operation				F	T	Operands & Remarks				
21	25	27	30	32	33	35	40	45	50						
ALPHA		STO								An Expression					

If the label field is left blank, it is ignored by the Assembler and has no effect on the assembled program. If column 21 contains an asterisk (*), the entire statement is treated as comments and appears only in the listing. If the field contains a symbolic name (label), and the statement represents a standard machine language operation (Add, Store, etc.), the value assigned to the label is the address of the assembled instruction, which is equal to the value of the Location Assignment Counter (see Location Assignment Counter) at the time the statement is encountered by the Assembler. Values assigned to labels of the various assembler instructions are specified in the section entitled Assembler Instructions.

Operation Field (Columns 27-30)

Each machine instruction and assembler instruction has a unique mnemonic operation code associated with it. When a particular operation is to be represented, its mnemonic code must be punched, left-justified, in columns 27-30 of the source statement record.

Program _____

Date _____

Programmed by _____

Page No. _____ of _____

Label		Operation		F		T		Operands & Remarks							Identification	
21	25	27	30	32	33	35	40	45	50	55	60	65	70	75	80	

Figure 1. Coding Form

Format Field (Column 32)

The format field specifies the type of machine instruction being represented and, in the use of short (one-word) instructions, how the displacement field is to be handled. Any one of four entries is permitted: two for short instructions, one for a direct long (two-word) instruction, and one for an indirectly-addressed long instruction. For convenience, these formats are referred to by the character used to specify them, namely blank format, X format, L format, and I format.

Blank Format. A blank in the format field (column 32) signifies a short instruction except with some of the extended mnemonics provided with the Disk Monitor Assembler, in which case a blank format

field specifies a long instruction. Bit 5 of the assembled instruction is set to zero. A blank also indicates that any expression in the operand field be interpreted as the desired effective address for the statement.

During execution of certain short instructions, the effective address is the sum of the displacement (last 8 bits of the instruction word) and the contents of the Instruction Address Register (IAR). A blank format for such instructions causes the assembler to subtract the current value of the Location Assignment Counter from the expression in the operand field. Thus, when this result is added to the IAR during execution of the instruction, the correct effective address is obtained.

The effective address of short Store Index (STX) instructions is always obtained by adding the displacement to the IAR. The displacement of the Load

Index (LDX), Load Status (LDS), WAIT, all shift instructions, and all condition testing instructions is never added to the IAR. The effective address of all other short instructions is obtained by adding the displacement to the IAR, if the instructions are not indexed; that is, if column 33 is blank or zero.

The X format suppresses the automatic subtraction of the address counter from the displacement operand value when the instruction is moved. Therefore, the X format should be used for a short instruction which will have an effective address obtained by adding the displacement to the IAR. This requirement is not in conflict with the relocation process, because the process shifts the whole program, including instructions and reference data, to a core storage area different from that for which it was assembled. The relative distances between instructions and data remain the same, and the displacements remain correct.

In a relocatable assembly, the expression specifying an operand modified by the IAR must be relocatable so that the actual displacement is an absolute quantity (see Expressions). If this rule is not followed, a relocation error will be indicated. Also, since displacements must lie in the range -128_{10} to $+127_{10}$, the value of the displacement-specifying expression must not be more than 127_{10} greater, nor more than 128_{10} less than the address of the next location after the instruction in which it appears; otherwise, an addressing error will be indicated. An example illustrating the blank format is shown below:

Assume A = location 1000_{10}
 B = location 1050_{10}

The value of the IAR will be 1001_{10} when instruction A is executed. Therefore, the value computed by the assembler for the displacement will be 49_{10} .

Label		Operation		F	T			
21	25	27	30	32	33	35	40	45
A		L, D,				B,		
		.						
		.						
		.						
		.						
		.						
B		D, C,				C, O, N, S, T,		

In the case of an instruction whose address is not modified by the IAR, the Assembler interprets the expression in the operand field as the desired contents of the displacement field, without modification. In this case, the operand specifying the displacement must be absolute and must be in the range -128_{10} to $+127_{10}$, or relocation and addressing errors result.

X Format. An X in the format field indicates to the Assembler that the related statement is to be assembled as a short instruction. It further indicates that any expression in the operand field is to be interpreted as the desired displacement value.

Consider the example illustrated in Figure 2; the purpose of this instruction sequence is to change the flow of a program by inserting a branch instruction in a location that previously contained a "no operation." If the branch instruction at BRCON were specified as MDX GO (i. e., blank format), the assembler would compute the displacement on the basis of the IAR value of 1101. (The IAR would have a value of 1101 if the BRCON instruction were executed where it was assembled.) However, the programmer, knowing the instruction will be executed at location SWTCH, computes the displacement himself and specifies the X format.

L Format. If column 32 contains the character L, it signifies a long (two-word) instruction with direct addressing. Bit 5 (F) of the assembled instruction is set to 1. The operand-field expression, which may be relocatable or absolute, is used to fill the second word (bits 16-31) of the assembled instruction. A second operand may be present, separated from the first operand by a comma (,). This operand may be used in one of two ways:

1. To specify symbolic condition codes for use with BSC, BSI and BOSC instructions.
2. To specify an expression that has a value in the range of -128 to $+127$ and is not relocatable.

This second operand yields bits to fill bit positions 8-15 of the assembled instruction.

I-Format. If column 32 contains the character I, it signifies an indirectly addressed long instruction. Bit 5 and bit 8 are set to 1. In all other respects an indirect instruction is treated exactly as a long direct instruction. If a displacement operand is specified, its high-order bit (bit 8) will always be a one, causing the displacement to be negative, because this bit is also the indirect flag bit.

Label	Operation	F	T	Operands & Remarks						Identification					
21*	25	27	30	32	33	35	40	45	50	55	60	65	70	75	80
<i>S.W.T.C.H</i>	<i>N.O.P.</i>														
	<i>L.D.</i>								<i>B.R.C.O.N. C.H.A.N.G.E. P.R.O.G.R.A.M. F.L.O.W. A.T. S.W.T.C.H.</i>						
	<i>S.T.O.</i>								<i>S.W.T.C.H.</i>						
<i>B.R.C.O.N</i>	<i>M.D.X.</i>			<i>X</i>					<i>G.O. - S.W.T.C.H. - 1</i>						

Figure 2. Use of X Format

Tag Field (Column 33)

Column 33 is used to specify an index register if one is required. The code in column 33 is the index register number; i.e., 1=Index Register 1, 2=Index Register 2, and 3=Index Register 3. A zero or a blank indicates that no index register is to be used.

If no tag is specified in an LD_X, MD_X, or ST_X instruction, the IAR is used. The example below shows an add instruction that addresses the core location whose address is zero plus the contents of Index Register 2.

Label	Operation	F	T	Operands & Remarks					
21	25	27	30	32	33	35	40	45	50
<i>S.U.M</i>	<i>A.</i>			<i>L2</i>	<i>0</i>				

Operands and Remarks Field (Columns 35-71)

The operand field is used to specify subfields in instructions and constants. The content of the operand

field for the various instruction formats are described under Format Field. Blanks must not appear within the operand(s) except as character values or in the EBC statements.

Some examples of one- and two-operand statements are shown in Figure 3.

Remarks Field

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

The Remarks field must appear to the right of the operand field and must be separated from it by at least one blank.

Comments Field

By placing an asterisk in column 21, the combined

<i>S.H.O.R.T</i>	<i>S.T.O.</i>					<i>A.C.C.U.1</i>									
<i>L.O.N.G.</i>	<i>M.D.X.</i>			<i>L</i>		<i>A.C.C.U.1, I.O.O. T.W.O. - O.P.E.R.A.N.D. S.T.A.T.E.M.E.N.T.</i>									
<i>L.O.N.G.</i>	<i>S.T.O.</i>			<i>L</i>		<i>A.C.C.U.1, O.N.E. - O.P.E.R.A.N.D. L.O.N.G. S.T.A.T.E.M.E.N.T.</i>									

Figure 3. One- and Two-Operand Statements

statement fields from columns 22-72 may be used for comments. The identification-sequence field (columns 73-80) should not be used for comments.

If it is necessary to continue comments on additional lines, each line must have an asterisk in column 21, as illustrated in Figure 4.

Identification-Sequence Field (Columns 73-80)

The identification-sequence field may be used for program identification and statement-sequence numbers. It is limited to columns 73-80. The information in this field normally is punched in every statement card. The Assembler, however, does not check this field.

STATEMENT WRITING

Symbolic language statements are accepted by the Assembler only if they conform to the rules of syntax presented in this section. Subsequent sections of this publication deal with the format and content of the specific types of assembler statements (machine instructions and assembler instructions). Instructions of both types are formed by using the basic elements described here. Many of the points introduced in this section are covered more extensively in subsequent sections.

Character Set

The following characters may be used in statements:

Monocase Alphabetics	A through Z, \$, #, @
Numerics	0 through 9
Special Characters	/*+=& ' <> ' . , ; ; () % - ? (blank)

The codes that the assembler accepts for these characters are listed in Appendix A. Appendix A also lists additional codes which may be used in comments statements, as character values, and as alphameric constants. The + and & special characters may be used interchangeably as operators.

Symbols

Storage areas, instructions, and other elements may be given symbolic names for the purpose of referring to them in the program. The symbolic name is called a symbol. It can contain up to five characters.

While the first character of a symbol must be alphabetic, the remainder may be alphabetic, numeric, or any combination of the two. No embedded blanks or special characters may be used. Any violation of these rules is detected by the Assembler and indicated as an error in the program listing.

The following are valid symbols:

PUNCH	START	N
A2345	LOOP2	BC\$#@

\$, # and @ are monocase alphabets, not special characters (see Character Set), and as such can be used in the label field.

The following symbols are invalid, for the reasons noted:

256B	First character is not alphabetic
RECORDAREA2	More than 5 characters
END 1	Contains a blank

If a symbol is to be used as an operand, it must be defined in the program by using it as the label of a statement. Two types of label assignments are allowed. In machine-instruction statements and certain assembler statements, the label is assigned an address equal to the current value of the Location Assignment Counter. In the Equate Symbol statement (see Symbol Definition Statement), the label is assigned the value specified in the operand of the statement.

Symbol Table. For every program assembled, a table of the symbols in that program is created. This is the symbol table; each entry in the table records the value and relocation property of a symbol.

All symbols defined in the program are entered in the symbol table. Symbols that appear in the label field of assembler instructions that do not use labels (for example, ABS, END, ENT) are not placed in the symbol table.

General Restrictions on the Use of Symbols. The following restrictions are imposed on the use of symbols:

- A symbol may appear only once in a program as the label of a statement. If a symbol is used as a label more than once, only the first usage is recognized. Each subsequent usage of the symbol as a label is ignored and, in the card/paper tape system, is noted as an error in the program listing. In addition, any reference to

Label		Operation		F	T	Operands & Remarks							
21	25	27	30	32	33	35	40	45	50	55	60	65	70
*T,H,E		S,T,E,R		S,K		I,N,C,O,L,2,I,M,A,K,E,S,T,H,I,S,A,C,O,M,M,E,N,T,S,L,I,N,E							
*A,N,A		T,E,R,I		K		S,R,E,Q,U,I,R,E,D,F,O,R,E,A,C,H,L,I,N,E,O,F,C,O,M,M,E,N,T,S							

Figure 4. Example of Comments Statement

such a symbol is noted as an error.

- The number of symbols that can be defined in a program is restricted by the amount of core storage available to the assembler. The number of symbols allowed is defined in the system operator's manual.

For example, in the sequence

Label		Operation		F	T	Operands & Remarks							
21	25	27	30	32	33	35	40	45	50	55	60	65	70
S,T,A,R,T		A,				B,E,T,A							
		S,				S,T,O,R,E							
		S,T,O,		L		A,D,D,R,I							
A,L,I,S,T		A,		L		L,I,S,T							
		D,				L,O,C,2							

LOCATION ASSIGNMENT COUNTER

The Assembler maintains a counter to assign sequential storage addresses to program statements. This counter is called the Location Assignment Counter. It always indicates the next available address. As each machine instruction is processed, the counter is incremented by the number of words assigned to that instruction. Certain assembler instructions also cause the Location Assignment Counter to be set or incremented, whereas others do not affect it (see Assembler Instructions).

Location Assignment Counter Overflow. The maximum value of the Location Assignment Counter is 65535, a 16-bit value. If a program being assembled causes the counter to be incremented beyond 65535, the Assembler retains only the rightmost 16 bits in the counter and continues the assembly, checking for any other source program errors. No usable object program is produced. The user can, however, still obtain a listing of the entire source program.

RELATIVE ADDRESSING

Once an instruction has been named by a symbol in the label field, it is possible for other instructions to refer to that instruction by using the same symbol. Moreover, it is possible to refer to instructions preceding or following the instruction named by indicating their positions relative to that instruction. This procedure is referred to as relative addressing. A relative address is, effectively, a type of expression (see Expressions).

control can be transferred to the second instruction by either of the following instructions:

Label		Operation		F	T	Operands & Remarks							
21	25	27	30	32	33	35	40	45	50	55	60	65	70
		B,S,C,		L		S,T,A,R,T,+1							
		B,S,C,		L		A,L,I,S,T,-3							

By using relative addressing, it is also possible to refer to a particular word within a block of reserved storage. For example, the instruction

Label		Operation		F	T	Operands & Remarks							
21	25	27	30	32	33	35	40	45	50	55	60	65	70
B,E,T,A		B,S,S,				50							

reserves a block of 50 words, in which BETA is the address assigned to the first word in the block. The address BETA+1 then refers to the second word, BETA+2 to the third word, and BETA+n to the (nth+1) word.

Relative addressing can also be effected by using the current value of the Location Assignment Counter in an operand. In symbolic language this value is denoted by an asterisk (*). (See The Asterisk Used as an Element.)

SELF-DEFINING VALUES

A self-defining value is a machine value or a bit configuration.

Self-defining values can be used to specify such program elements as data, masks, addresses, and address increments. The type of representation selected (decimal, hexadecimal, or character) depends on what is being specified.

Decimal Values

A machine decimal value is an absolute number from 0 to 65535. It is assembled as its binary equivalent. Some examples of decimal, self-defining values are

500	003
17	52324
7230	1

If a number larger than 65535 is specified in address arithmetic, the value is truncated modulo 65536; that is, only the low order 16 bits of the binary value are retained.

Hexadecimal Values

A hexadecimal value is an unsigned hexadecimal number written as a sequence of digits. The digits must be preceded by a slash (/). The hexadecimal digits represent the 16 possible combinations of four bits.

Each hexadecimal digit is assembled as its four bit value. The hexadecimal digits and their bit patterns are as follows:

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

The following are examples of hexadecimal, self-defining values:

/FFFF	} equivalent
/AB12	
/379B	
/F2	
/00F2	

If more than four hexadecimal digits are specified in one sequence, only the four low-order digits are retained by the assembler. If less than four hexadecimal digits are specified, they are entered, right-justified.

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

Character Values

A character value is a single character, preceded by a period. A character value may be a blank, any

combination of punches in a single card column, or a paper tape character that translates into the eight-bit IBM Extended BCD Interchange Code. Appendix A is a table of these combinations, their interchange codes and, where applicable, their printer graphics. A period used as a character value is represented as two periods in sequence, (i. e., ..).

Examples of character values are:

- . A
- . 1
- . 2
- . D
- . (blank)

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

EXPRESSIONS

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

An expression has three components: elements, terms, and operators.

Elements

The smallest component of an expression is an element. An element is either a single symbol or a single self-defining value. The following are valid elements:

- TMP
- /1A6
- . B
- A
- *
- 4

The Asterisk Used As an Element

When used as an element the asterisk is relocatable and stands for the current value of the Location Assignment Counter for the instruction in which it appears (i. e., the rightmost word of the current instruction + 1). Thus, the asterisk as an element can have different values for different instructions.

Label		Operation				F	T			
21	25	27	30	32	33	35	40	45		
		L,D				A,B,C				
SUM		A				D,E,F				
		S				D,A,T,A				
		B,S,C	L			SUM, +				

The last instruction is a conditional branch to location SUM and can be written

Label		Operation				F	T			
21	25	27	30	32	33	35	40	45		
		B,S,C	L			*-4,+				

Be sure the asterisk refers to the proper word when it is used with a long instruction or in an area where long instructions are present. In the previous example, the BSC instruction will become two machine language words after assembly. Therefore, during assembly of the BSC instruction, the Location Assignment Counter contains a value one greater than if the BSC were a short instruction.

Terms

A term can consist of a single element, two elements separated by an asterisk (which denotes multiplication), or three elements each separated by an asterisk, etc. A term must begin with an element and end with an element, but is not permissible to write two elements in succession. The following are valid terms:

```
TMP * FUNC * TAXY
A * 4
X * Y * 5
6 * 4096
3
```

Operators

An operator is a character that denotes an arithmetic function. The recognized operators are + or & (plus or ampersand), - (minus), and * (asterisk), denoting addition, subtraction, and multiplication, respectively; An operator must be used between two terms. Two operators may not be used in succession.

There is no ambiguity between the use of the asterisk as an element and the use of the asterisk as an operator to denote multiplication, because the

position of the asterisk always makes clear what is meant. Thus, **10 means "the value of the Location Assignment Counter multiplied by 10."

Evaluation of Expressions

From a symbolically written operand, the evaluation procedure derives an integer value that can be used as (1) a displacement value in a short instruction, (2) an address in a long instruction, or (3) an absolute numeric quantity.

An expression is evaluated as follows:

1. Each element is replaced by its numeric value.
2. Each term is evaluated by performing the indicated multiplications from left to right, in the order in which they occur. In multiplication, the low-order 16 bits are retained.
3. The terms are combined from left to right, in the order in which they occur. If the result is negative, it is replaced by its 2's complement.

Grouping of terms, by parentheses or otherwise, is not permitted; however, this restriction can often be circumvented. For example, the product of 25 times the quantity B-C can be expressed as

$$25 * B - 25 * C$$

Types of Expressions

In addition to evaluating expressions, the Assembler must decide whether the expression is absolute or relocatable. Without this information the Assembler would be unable to assign the proper relocation indicator bits for use during loading.

Rules for Determining the Type of Expression

The rules by which the expression type is determined are:

- A symbol that is defined by means of the Location Assignment Counter is a relocatable element.
- Decimal and hexadecimal integers and character values are absolute elements.
- A relocatable element alone is a relocatable expression.
- A relocatable element, plus or minus an absolute element, is a relocatable expression.

- The difference of two relocatable elements is an absolute expression.
- A symbol that has been equated to an expression (by means of the EQU assembler instruction) assumes the same relocation property as that expression.

These rules are clarified by the following example:

Assume that a programmer wishes to incorporate a table into a relocatable program, and he knows that he may later wish to add or delete items without changing program references to the table. The first step is to assign symbols to the first (lowest-addressed) word in the table and to the location immediately after the last (highest-addressed) word of the table. These symbols could be BGTBL and ENTBL, respectively. Regardless of the number of items in the table or of the number of later additions or deletions, the number of words in the table is always equivalent to the value of the expression ENTBL-BGTBL. This illustrates the rule that the difference of two relocatable elements is an absolute expression.

Expanding this example, assume the programmer wishes to use a second table the same length as the first. The first (lowest addressed) word of the second table can be indicated by the symbol STBL. Then, the location following the last (highest-addressed) word of the second table can be indicated by the expression

$$STBL + ENTBL - BGTBL$$

This address is subject to relocation; hence, the expression is relocatable, following the rule that a relocatable element plus or minus an absolute element is a relocatable expression.

Procedure for Determining the Type of Expression

The following paragraphs describe the procedure for determining expression type (absolute or relocatable):

- Discard any term that contains only absolute elements.
- Examine each term of the expression. If any term contains more than one relocatable element, the expression will yield a relocation error.

- Replace each relocatable element by the symbol r, and replace each absolute element by its value. This yields a new expression which involves only numbers and the symbol r.
- Rewrite the expression in simplest form by evaluating it according to the address arithmetic rules given above in the section, Evaluation of Expressions.

If the result is an integer, the operand is absolute. If the result is r, the expression is relocatable. If the result contains r to any power other than one, or contains r with a coefficient other than one, the operand does not have a well-defined relocation property and will yield a relocation error. The following examples illustrate this procedure.

NOTE: When the terms absolute symbol and relocatable symbol are used in text, they mean symbols that refer to addresses.

Example 1: Consider the expression,

$$4+3*TRANS-2*FUNC+COUNT$$

where TRANS and FUNC are relocatable symbols, and COUNT is an absolute symbol. Discarding the terms involving only absolute elements leaves

$$3*TRANS-2*FUNC$$

This does not contain any illegal terms. Replacing each symbol by the letter r results in

$$3*r-2*r$$

Evaluating this produces r; therefore, the expression is relocatable.

Example 2: Consider the expression,

$$2*3*TRANS-FUNC$$

This reduces to

$$2*3*r-r$$

or

$$5r$$

This is neither r nor a number; therefore, the expression will cause a relocation error.

Example 3: Consider the expression,

$$A*2*R-A*A*R+5$$

where A is an absolute symbol, and R is a relocatable symbol. The expression is absolute if the value of A is zero or two and relocatable if the value of A is 1. If the value of A is anything else, a relocation error will result.

In the following examples, A , B , C , and D are relocatable symbols, and J , K , L , M , and N are absolute symbols.

Relocatable expressions:

A	$1*A$
$A+J$	$250*A-249*B$
$A+B+C-D-*$	$100*A+50*B-75*C-74*D$

Absolute expressions:

12345	$0*A$
$A-B+C-D+5$	$500*A-400*B-100*C$

Relocation Errors

If a source program contains an expression having in it one or more of the following, that expression is flagged as a relocation error.

- The negative (complement) of a relocatable element
- An absolute element minus a relocatable element
- The sum of two relocatable elements

In the following examples, A , B , C , and D are relocatable symbols, and J , K , L , M , and N are absolute symbols.

$A+B$	$(+2r)$	$A*B$	(r^2)
$-A$	$(-1r)$	$2*A$	$(2r)$
$15-*$	$(-1r)$	$5*A-6*A$	$(-1r)$
$A+J+M+N+B-C+D+L(+2r)$			

NOTE: In an absolute assembly headed by an ABS statement (described later), all symbols and asterisk values are defined as being absolute; therefore, no relocation errors are possible.

MACHINE-INSTRUCTION STATEMENTS

All machine instructions can be represented symbolically as assembler language statements. There are two basic formats: short and long. However, within each basic format, further variations are possible.

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

Any machine-instruction statement can be named by a symbol, which other assembler statements can use as an operand. The value of the symbol is the address of the leftmost word assigned to the assembled instruction.

MNEMONICS

A list of all IBM 1130 machine language instructions and their associated mnemonics, including those mnemonics available for the monitor system only, is given in Table 1.

Condition-Testing Instructions (BSC, BOSC, BSI)

The machine instructions Branch or Skip on Condition (BSC), Branch Out or Skip on Condition (BOSC), and the long form of Branch and Store Instruction counter (BSI) use bits 10-15 of the displacement to test any combination of six conditions associated with the accumulator. When coding these instructions, the user does not use an expression to specify the displacement field, but, instead, writes a series of unique characters, each of which represents one bit of the condition-testing mask. These character symbols may be written in any combination; the bits they represent are combined by the assembler in a logical OR fashion. The symbols and their representations are:

Unique Character	Condition	Description	Bit Position Set to 1
O (Alpha)	Overflow	Skip or do not branch if Overflow indicator <u>off</u>	15
C	Carry	Skip or do not branch if Carry indicator <u>off</u>	14
E	Even	Skip or do not branch if bit 15 of Acc =0	13
+ or &	Plus	Skip or do not branch if bit 0 of the Acc =0, but not all bits of Acc =0	12
-	Minus	Skip or do not branch if bit 0 of Acc =1	11
Z	Zero	Skip or do not branch if all bits of Acc =0	10

Examples:

Operation	F	T			
27	30	32	33	35	
B,S,C,				+	Skip on plus condition
B,S,C,				+,-	Skip on non-zero (plus or minus)
B,S,C,				Z,-	Skip on non-plus (zero or minus)
B,S,C,				C	Skip if Carry indicator off
B,S,C,	L			EX,I,T,+,-	Branch to EXIT if not plus (zero or minus)
B,S,C,	L			EX,I,T,+,-	Branch to EXIT if zero (not plus or minus)
B,S,C,	L			EX,I,T	Unconditional Branch to EXIT
B,S,C,	L	I		0,Z,+	Branch to the contents of XR1 if minus (not zero or plus)
B,S,I,	L			S,U,B,R,+O	Branch and Store instruction counter to SUBR if Overflow is on

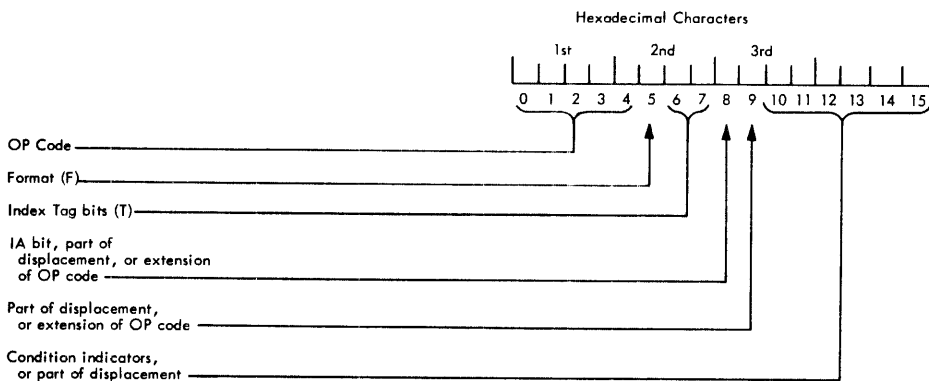
Table 1. Machine Instruction Mnemonics

Mnemonic	OP Code (Hexadecimal Representation) ¹	Instruction
<u>Load and Store</u>		
LD	C00	Load Accumulator
LDD	C80	Load Double
LDX	600	Load Index
LDS*	200	Load Status
STO	D00	Store Accumulator
STD	D80	Store Double
STX	680	Store Index
STS	280	Store Status
<u>Arithmetic</u>		
A	800	Add
AD	880	Add Double
S	900	Subtract
SD	980	Subtract Double
M	A00	Multiply
D	A80	Divide
AND	E00	And
OR	E80	Or
EOR	F00	Exclusive Or
MDM	+ 5 740	Modify Memory
<u>Branch</u>		
B	+4 700 or 4C0	Branch
BSI	400	Branch and Store Instruction Counter
BSC	480	Branch or Skip Conditionally
BP	t6 4C30	Branch Accumulator Positive
BNP	t6 4C03	Branch Accumulator Not Positive
BN	t6 4C28	Branch Accumulator Negative
BNN	t6 4C10	Branch Accumulator Not Negative
BZ	t6 4C18	Branch Accumulator Zero
BNZ	t6 4C20	Branch Accumulator Not Zero
BC	t6 4C02	Branch on Carry
BO	t6 4C01	Branch on Overflow
BOD	t6 4C04	Branch Accumulator Odd
SKP*	+ 480	Skip on Condition(s)
BOSC	2 484	Branch Out or Skip Conditionally
MDX	700	Modify Index and Skip
<u>Shift</u>		
SLA*	100	Shift Left Accumulator
SLT*	108	Shift Left Accumulator and Extension
SLC*	10C	Shift Left and Count Accumulator and Extension
SLCA*	104	Shift Left and Count Accumulator
SRA*	180	Shift Right Accumulator
SRT*	188	Shift Right Accumulator and Extension
RTE*	18C	Rotate Right
XCH*	+3 18D	Exchange Accumulator and Extension
<u>Input/Output</u>		
XIO	080	Execute I/O
<u>Miscellaneous³</u>		
NOP*	100	No Operation
WAIT*	300	Wait

*Valid in short format only

+Not included in card/paper tape Assembler or DM1 Assembler.

- The hexadecimal representation of the machine operation code is derived from the instruction format in the manner shown below.
- Bits 6 and 7 are assumed to be zeros because they do not enter into the makeup of any operation codes.
- Same as BSC with Bit 9 set to one.
- An operand should not be specified.
- When branch is short (Blank or X format), this operation code is assembled as an MDX (700). If the branch is long (L or I format), this operation code is assembled as a BSC with Bit 5 set to one (4C0).
- This instruction is automatically assembled as a long instruction (L is not required in the format field). Note that an attempt to use indirect addressing will result in a syntax error. Indexing is not permitted with this extended operation code.
- Extended conditional branch operation codes are assembled automatically as long instructions. (L is not required in the format field). Note that the proper condition code bits are preset, and further condition bits may not be specified following the operand.



ADDITIONAL MONITOR SYSTEM MNEMONICS (DM2)

Several new mnemonic operation codes which are equivalent to a Branch or Skip on Condition (BSC) may be used with the DM2 System. The operation code to be used for a specific job depends on the format and condition code required.

A new mnemonic MDM has been introduced that may be used in place of an unindexed MDX long. XCH may be used in place of RTE 16.

Examples of the additional DM2 System mnemonics are shown in Table 2. The mnemonics are listed below.

Skip on Condition (SKP). The condition codes (+, -, Z, E, O, and C) are specified as with a short BSC instruction. This instruction must not be indexed.

Branch Unconditionally (B). If the Format field contains an L or I, the BSC operation code is used with bit 5 set to one. Condition codes are not allowed after the address expression in the Operand field. If the Format field is left blank or contains an X, the MDX operation code is used, and the expression in the Operand field is used to form the displacement.

Branch Accumulator Positive (BP). Condition codes for accumulator zero (Z) and accumulator negative (-) are set to one.

Branch Accumulator Not Positive (BNP). Condition code for accumulator positive (+) is set to one.

Branch Accumulator Negative (BN). Condition codes for accumulator zero (Z) and accumulator positive (+) are set to one.

Branch Accumulator Not Negative (BNN). Condition code for accumulator negative (-) is set to one.

Branch Accumulator Zero (BZ). Condition codes for accumulator positive (+) and accumulator negative (-) are set to one.

Branch Accumulator Not Zero (BNZ). Condition code for accumulator zero (Z) is set to one.

Branch on Carry (BC). Condition code for Carry indicator off (C) is set to one.

Branch on Overflow (BO). Condition code for Overflow indicator off (O) is set to one.

Branch Accumulator Odd (BOD). Condition code for accumulator even (E) is set to one.

NOTE: Condition codes may not be used with any of the above instructions, except SKP, since the condition code is implicit in the extended mnemonic. The conditional branch instructions (all except SKP and B) are always assembled as long instructions; thus, the Format field need not contain an L, although the instruction is not classed as an error if L is specified. Indirect addressing may be specified.

Modify Memory (MDM). Contents of the location specified by the first operand is incremented or decremented by the value of the second operand. The second operand must be in the range -128 to +127.

NOTE: This instruction is always assembled as a long instruction; thus, the Format field need not contain an L, although the instruction is not classed as an error if L is specified. Indexing and indirect addressing must not be specified. If the operand becomes zero or changes sign, the next word in the program will be skipped.

Exchange Accumulator and Extension (XCH). Exchange is identical to a RTE of 16. No operand is specified with this instruction.

Table 2. Examples of New (Extended) Machine Instruction Mnemonics (DM2 only)

New Instruction Statements					Equivalent Statements					Operations Performed		
27	30	32	33	35	40	27	30	32	33	35	40	
SKP				+		BSC				+		Skip if accumulator is positive
SKP				+ -		BSC				+ -		Skip if accumulator is non-zero
SKP				Z		BSC				Z		Skip if accumulator is zero
SKP				O		BSC				O		Skip if Overflow indicator is off
SKP				C		BSC				C		Skip if Carry indicator is off
SKP				+ - C		BSC				+ - C		Skip if accumulator is non-zero or if Carry indicator is off
B				EXIT		HDX				EXIT		Branch unconditionally to EXIT, where EXIT must be within normal displacement range.
B	L			ALPH		BSC	L			ALPH		Branch unconditionally to ALPH
BZ				BETA		BSC	L			BETA, + -		Branch to BETA if accumulator is zero
BN				BETA		BSC	L			BETA, Z +		Branch to BETA if accumulator is negative
BNZ	I			BETA		BSC	I			BETA, Z -		Branch indirectly to BETA (i.e., the address specified by contents of BETA) if accumulator is non-zero
BN				RTNA		BSC	L			RTNA, Z +		Branch to RTNA if accumulator is negative
BNN				RTNB		BSC	L			RTNB, -		Branch to RTNB if accumulator is non-negative (zero or positive)
BP				SUB@		BSC	L			SUB@, Z -		Branch to SUB@ if accumulator is positive
BP	I			SUB\$		BSC	I			SUB\$, Z -		Branch indirectly to SUB\$ (i.e., the address specified by the contents of SUB\$) if accumulator is positive
BNP				SUB#		BSC	L			SUB#, + -		Branch to SUB# if accumulator is non-positive (zero or negative)
BC				ENTR+1		BSC	L			ENTR+1, C		Branch to ENTR+1 if Carry indicator is on
BC	I1			Ø		BSC	I1			Ø, C		Branch indirectly to address specified by contents of index register 1 if Carry indicator is on
BO	Z			5		BSC	L2			5, O		Branch to address specified by contents of index register 2 plus 5 if Overflow indicator is on
BOD				\$AFE		BSC	L			\$AFE, E		Branch to \$AFE if accumulator is odd
MDM				SAVA, +5		HDX	L			SAVA, +5		Increment contents of core location SAVA by 5
MDM				/1D6A, 100		HDX	L			/1D6A, 100		Increment contents of core location /1D6A by 100 decimal
MDM				A, -12		HDX	L			A, -12		Decrement contents of core location A by 12
XCH						RTE						Exchange the accumulator and extension (rotate right 16)

Just as machine instructions are requests to the computer to perform a sequence of operations during program execution, assembler instructions are requests to the Assembler to perform certain operations during the assembly. In contrast to machine-instruction statements, assembler-instruction statements do not always cause machine instructions to be included in the assembled program. Some, such as BSS and BES, generate no instructions but do cause storage areas to be set aside for constants and other data. Others (e. g., EQU) are effective only during the assembly; they may or may not generate something in the assembled program. If nothing is generated, the Location Assignment Counter is not affected.

The following is a list of all assembler statements permitted by the IBM 1130 Card/Paper Tape Assembler. These statements are also valid for the Monitor Assembler. Additional statements are provided for the Monitor Assembler and are listed in the section Monitor Assembler Statements.

Program Control

- ABS - Absolute Assembly
- LIBR - Transfer Vector Subroutine
- SPR - Standard Precision
- EPR - Extended Precision
- ORG - Define Origin
- END - End of Source Program

Data Definition

- DC - Define Constant
- DEC - Decimal Data
- XFLC - Extended Floating Constant
- EBC - Extended Binary Coded Information

Storage Allocation

- BSS - Block Started by Symbol
- BES - Block Ended by Symbol

Symbol Definition

- EQU - Equate Symbol

Program Linking

- ENT - Define Subroutine Entry Point
- ISS - Define Interrupt Service Entry Point
- ILS - Define Interrupt Level Subroutine
- CALL - Call Subroutine (2-word call)
- LIBF - Call Subroutine (1-word call)

PROGRAM CONTROL STATEMENTS

Program control statements are used to set the Location Assignment Counter to a specific value, to define the end of a source program, or to specify whether a particular program is to be assembled as absolute or relocatable. None of these assembler statements generate machine-language instructions or constants in the object program.

ABS — Assemble Absolute

An ABS statement is used to specify that a main program is to be assembled as an absolute program. An absolute program is one in which the core locations used at execute time are the same as those specified by the programmer in the source program. The ABS statement is punched as shown below and is then used as the first statement of a source program.

Label		Operation			F	T			
21	25	27	30	32	33	35	40	45	
		A	B	S					

If the first (non-comment) statement of a source program is not an ABS statement, the program will be assembled as relocatable. In an absolute assembly headed by an ABS statement, all symbols and asterisk values are defined as absolute quantities; therefore, no relocation errors are possible. The significance of relocatable and absolute assemblies is explained in the following paragraphs.

Relocatable Assembly

Some programs assembled by the IBM 1130 Assembler are absolute; that is, the locations of assembled instructions are known during the assembly and the location on the listing is the actual location where a particular word is loaded. However, subroutines used by an absolute program must be in such a form that they may be loaded at various locations; otherwise, it would be necessary for the user to reassemble the subroutines each time he assembled a main program that required them. Therefore, all subroutines must be and main programs may be assembled relocatable.

Every relocatable program or subroutine produced by the IBM 1130 Assembler is assembled as though it begins at location zero. Since a job to be executed may contain several subroutines, it is obvious that they cannot all be loaded into locations starting with location zero. In fact, no relocatable program is ever loaded at location zero; instead, each program is relocated. The relocatable main program is loaded into the first available location. Subroutines are then loaded into successively higher locations of core storage, each beginning with the

next even location after the last core storage location used by the preceding subroutine. When a particular program has been loaded, the address of the first word is called the load address for that program.

Thus, the address in core storage actually occupied by an instruction of the program is the address assigned to that instruction during assembly, plus the load address of that program. To keep the program self-consistent, the load address must be added to the address of many (but not all) 2-word instructions, and those constants whose values are relocatable.

This process of conditionally adding the load address is performed by the loading program before execution and is called relocation. In relocating instructions, the loading program is guided by relocation indicator bits which are a part of the object program.

Absolute Assembly

The programmer uses the ORG assembler statement in his source program to specify the locations into which the object program resulting from an absolute assembly is loaded. Subroutines are loaded into successively higher even-core locations following the end of the main program.

Only main programs may be assembled absolute; subroutines must be assembled relocatable.

LIBR — Transfer Vector Subroutine

An LIBR statement is used as the first statement of a subroutine to specify that the subroutine is to be called by LIBF statements only (see Program-Linking Statements). The absence of an LIBR statement specifies that the subroutine is to be called by CALL statements only. LIBR statements are for subroutines only, as ABS statements are for main programs only. An LIBR statement needs no operands.

SPR — Standard Precision, EPR — Extended Precision

The SPR or EPR statement specifies that the program (main or subroutine) in which it appears uses standard precision or extended precision, respectively, for arithmetic operations. If these statements are included in the user's programs, the loader ensures that main programs and subroutines always match with regard to precision. Their use is optional, however.

If used, the SPR or EPR statement must follow the ABS or LIBR statement. If no ABS or LIBR statement is used, the SPR or EPR statement is the first statement in the program.

ORG — Define Origin

This assembler instruction is used to set the Location Assignment Counter (i.e., the next location to be assigned) to any desired value. In this way the programmer is able to control the assignment of storage to instructions, constants, and data. If a Define Origin statement is not the first entry in an absolute source program, the processor begins the assignment of storage at a location compatible with the size of the applicable loader (Card/Paper Tape Assembler) or the size of the resident monitor plus DISKN (Disk Monitor Assembler). A typical Define Origin statement is shown below.

Label	Operation	F	T	
21	25	27	30	32 33 35 40 45
	ORG			3,0,0,0

The label, if used, is assigned a value equal to the value of the Location Assignment Counter at the time the statement is encountered in the source program. (This assignment is made before the counter is modified.) If any symbols are used in the expression, they must have been previously defined. In a relocatable assembly, an absolute expression in the operand field is considered a relocation error and the statement is ignored.

Some examples of Define Origin statements are given below:

Label	Operation	F	T	Oper
21	25	27	30	32 33 35 40 45 50
	ORG			X,Y,Z
S,T,A,R,T	ORG			X,Y,Z+5,0
S,T,A,R,T	ORG			*+5,0, L.O.C, C.T.R.+5,0

If the label XYZ has been previously defined as 1000₁₀ the first entry directs the assembler to begin the assignment of succeeding entries at location 1000. The second entry directs the Assembler to begin the assignment of succeeding entries 50 core locations beyond the location that has been assigned to the symbol XYZ. The third entry directs the Assembler to begin the assignment of succeeding entries at the

address specified by the current address of the Location Assignment Counter plus 50.

END — End of Source Program

An END statement is the last statement of a source program; it indicates to the assembler that all statements of the source program have been processed. An END statement is also used to define the execution address of the main program. To do this, the END statement requires an operand that represents the starting address of the program. At the completion of loading, execution begins at the address specified by the operand. For subroutines, all entry points are specified by ENT statements (described later); therefore, the operand of the END statement for a subroutine is blank.

The following statements illustrate both types of END statements.

Label	Operation	F	T	Operands
21	25	27	30	32 33 35 40 45 50
	END			END OF PROGRAM
	END			GO BRANCH TO GO

DATA DEFINITION STATEMENTS

Data Definition statements are used to enter data constants into storage. The statements can be named by symbols so that other program statements can refer to the fields generated. Any type of data definition statement can be used in standard or extended precision program.

DC — Define Constant

The Define Constant statement is for generating constant data in main storage. Data can be specified as characters, hexadecimal numbers, decimal numbers, storage addresses, or any valid expression. One 16-bit word is generated for each DC statement. The format of this statement is shown below:

Label	Operation	F	T	Operands
21	25	27	30	32 33 35 40 45
LABEL	DC			AN EXPRESSION

If a label is used, the address assigned to it is the location of the generated data word and is equal to the current value of the Location Assignment Counter. Some examples of DC statements follow:

Label	Operation	F	T	Operands
21	25	27	30	32 33 35 40 45 50
H, E, X	DC			/, F, F, F, H, E, X, C, O, N, S, T
D, E, C	DC			- 3, 8, 5, D, E, C, I, N, T, G, E, R
A, L, P, H, A	DC			, B, C, H, A, R, C, O, N, S, T
A, D, D, R, S	DC			A, L, P, H, A, + 5, A, D, D, R, C, O, N

DEC — Decimal Data

The Decimal Data statement is used to enter binary data, expressed in decimal form, into a program. One DEC statement generates two 16-bit words of binary information. The format of the DEC statement is as follows:

Label	Operation	F	T	Operands & Res
21	25	27	30	32 33 35 40 45 50
LABEL	DEC			Decimal Data Item

If a label is used, its value is equal to the current value of the Location Assignment Counter if the current value is even; if the current value is odd, the label will be equal to the current value plus one. The label is assigned to the leftmost word of the generated constant. The types of data permitted in the operand field are described in the paragraphs entitled Decimal Data Items. An example of a DEC statement follows:

Label	Operation	F	T	Operands
21	25	27	30	32 33 35 40 45
D, A, T, A	DEC			7, 1, 9

If the value of the Location Assignment Counter is 1000 when the DEC statement is encountered, the two words in storage look like this:

Location	Contents in Hexadecimal Form
01000	0000
01001	0013

Decimal Data Items

A decimal data item is used to specify, in decimal form, two or three words of data to be converted into binary form. Decimal data items are used in the

operand field of DEC assembler statements. Three types of decimal-data items are permitted: decimal integers, real numbers, and fixed-point numbers. A real decimal-data item can also be used as the operand of an XFLC statement that generates a 3-word constant.

Decimal Integers. A decimal integer is composed of a series of numeric digits with or without a preceding plus or minus sign. The allowable range of decimal integers is $-(2^{31}-1)$ to $2^{31}-1$.

Examples

<u>Decimal Integer</u>	<u>Stored As</u>
50	00000032 ₁₆
1535	000005FF ₁₆
-3729	FFFFF16F ₁₆ (2's complement)

Real Numbers. A real number has two components: a mantissa and an exponent.

- Mantissa — The mantissa is a signed or unsigned decimal number, which can be written with or without a decimal point. The decimal point can appear at the beginning, at the end, or within the decimal number. If the exponent (see below) is present, the decimal point can be omitted, in which case it is assumed to be located at the right-hand end of the decimal number.
- Exponent — The exponent consists of the letter E, followed by a signed or unsigned decimal integer. The exponent part can be omitted if the mantissa contains a decimal point. If used, it must follow the mantissa.

A real number is converted to a normalized, real binary number. The exponent part, if present, specifies a power of ten by which the mantissa is multiplied during conversion. For example, all of the following real numbers are equivalent and will be converted to the same real binary number.

4.500
45.00E-1
4500E-3
.4500E1

In standard precision, the above real numbers are converted and stored in two consecutive storage locations as follows:

<u>Word 1</u>	<u>Word 2</u>
4800	0083

The DEC assembler instruction stores real numbers in the standard precision real number format described in the system subroutine library manual.

Fixed Point Numbers. A fixed-point number can have up to three components: a mantissa, an exponent, and a binary-point identifier.

- Mantissa — The mantissa is the same as described for real numbers.
- Exponent — The exponent is the same as described for real numbers.
- Binary-Point Identifier — This identifier consists of the letter B, followed by a signed or unsigned decimal integer. The binary-point identifier must be present in a fixed-point number and must come after the mantissa. If the number has an exponent, the binary point identifier may precede or follow the exponent.

A fixed-point number is converted to a fixed-point binary number that contains an understood binary point. The purpose of the binary-point identifier of the number is to specify the location of this understood binary point within the word. The number that follows the letter B specifies the number of binary places in the word to the left of the binary point (that is, the number of integral places in the word). The sign bit is not counted. Thus, a binary-point identifier of zero specifies a 31-bit binary fraction. B2 specifies two integral places and 29 fractional places. B31 specifies a binary integer. B-2 specifies a binary point located two places to the left of the leftmost bit of the word; that is, the word would contain the low-order 31 bits of binary fraction. As with real numbers, the exponent, if present, specifies a power of ten by which the mantissa is multiplied during conversion.

A fixed-point number preceded by a minus sign is stored in 2's complement form.

The following fixed-point numbers all specify the same configuration of bits, but not all of them specify the same location for the understood binary point:

22.5B5
11.25B4
1125B4E-2

1125E-2B4
9B7E1

All of the above fixed-point numbers are converted to the same binary configuration, whose hexadecimal representation is:

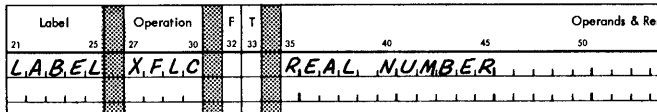
Word 1
5A00

Word 2
0000

XFLC — Extended Real Constant

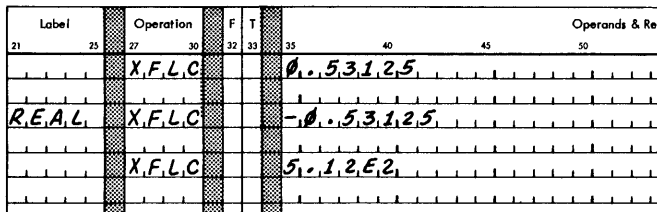
The XFLC assembler instruction is used to introduce into a program an extended precision real constant, expressed in three consecutive data words. When assembled, this instruction produces a format identical to the extended range real format described in the system subroutine library manual.

The format of the XFLC instruction is shown below:



The label is optional; if it is used, it is assigned to the location of the leftmost word generated.

Some examples of the XFLC instruction are shown below:

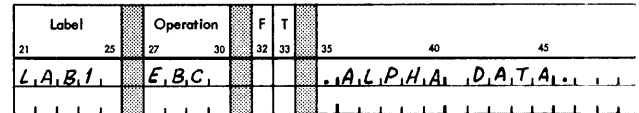


The data (in hexadecimal form) generated by each of these examples is

1. Word 1 Word 2 Word 3
 0080 4400 0000
2. Word 1 Word 2 Word 3
 0080 BC00 0000
3. Word 1 Word 2 Word 3
 008A 4000 0000

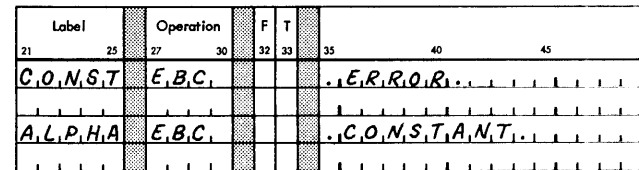
EBC — Extended Binary Coded Information

The EBC statement is used to generate data words, each consisting of two 8-bit characters in the Extended BCD Interchange Code (see Appendix A). Up to 18 sixteen-bit words can be generated with one EBC statement. The format of the statement is shown below:

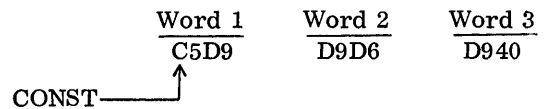


If a label is present, it is assigned to the location of the leftmost word generated. The operand field contains the alphameric data to be represented in storage. This data must begin and end with a period. The data can be any valid character in the Extended BCD Interchange Code, including the period.

Examples

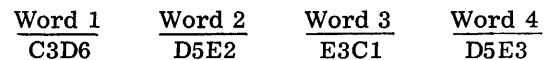


The first example generates three words of data, with the location of the label CONST assigned to the leftmost location of the first word generated.



Note that if the constant has an odd number of characters, as in the above example, the last word of data ends with the 8-bit equivalent of blank.

The second example generates four words of data:



NOTE: A period may not appear in the remarks field of an EBC instruction.

STORAGE ALLOCATION STATEMENTS

Storage allocation statements are used to reserve blocks of storage for data or work areas. Two such statements are available with the IBM 1130 Assembler: Block Started by Symbol and Block Ended by Symbol.

BSS — Block Started by Symbol

The BSS assembler instruction is used to reserve an area of core storage, within a program, for data storage or for working space. The format of the BSS instruction follows:

Label		Operation		F	T	Operands & Re			
21	25	27	30	32	33	35	40	45	50
LABEL		BSS					Absolute Expression		

The expression specifies the number of words to be reserved; the label, if specified, refers to the leftmost word reserved. The location of the block of storage within the object program is determined by the location of the BSS statement within the source program.

If the character E is punched in column 32, the assembler assigns the leftmost word of the reserved location to the next available even location. If a blank or any character other than E appears in column 32, the assembler assigns the leftmost word of the reserved area to the next available location regardless of whether that location is even or odd. This feature is useful when defining areas for use with double precision instructions.

A BSS statement with an E format and an operand value of zero causes the Location Assignment Counter to be made even (if necessary) before the next instruction is assembled.

A BSS instruction causes an area to be reserved, not cleared; therefore, it should not be assumed that an area reserved by a BSS instruction contains zeros.

Any symbols in the operand field of a BSS assembler instruction must have been previously defined. The expression in the operand field must be an absolute expression.

In the following example, the symbol AREA is equivalent to 3000; the next location assigned is 3028.

Label		Operation		F	T	Operands & Re			
21	25	27	30	32	33	35	40	45	
		ORG					3000		
AREA		BSS					28		

BES — Block Ended by Symbol

The BES instruction is identical to the BSS instruction except that the address assigned to the label is the rightmost word in the area plus 1, i.e., the next location available for assignment.

In the previous example, the symbol AREA is equivalent to 3028.

SYMBOL DEFINITION STATEMENT

One symbol definition statement (EQU) is available in the IBM 1130 Assembler language.

EQU — Equate Symbol

The EQU statement is used to assign to a symbol a value other than the value of the Location Assignment Counter at the time the symbol is encountered. The format of the EQU statement is

Label		Operation		F	T	Operands & Re			
21	25	27	30	32	33	35	40	45	50
SYMBOL		EQU					An Expression		

The symbol in the label field is made equivalent to the value of the expression. The expression may be absolute or relocatable. All symbols appearing in this expression must have appeared as a label in a previous statement. If an asterisk (*) is used as the expression, the value assigned to it is the next location to be assigned by the assembler.

Examples

Label		Operation		F	T	Operands & Re			
21	25	27	30	32	33	35	40	45	
NAME		EQU					26		
LOOPI		EQU					NAME+1		

In the first example, the symbol NAME is assigned a value of 26. In the second example, the symbol LOOP is assigned a value of 27.

Word 30 of the header record can be set for identification purposes as shown below. Word 30 is not used by any of the 1130 programs.

LINKING STATEMENTS

Linking statements are used to establish communication between a main program and its subroutines or between a program and the 'Monitor system.

<u>Label</u>	<u>ISS Header Word 30</u>
blank	blank
1130	1
1800	2

ENT - Define Subroutine Entry Point

The ENT statement should be used to define the entry point(s) in all subroutines except ISS and ILS. Up to fourteen entry points (ten with the Card/Paper Tape Assembler) may be defined for each subroutine (this would require an equal amount of ENT statements). The format of the ENT statement is shown below.

NAME is as described for the ENT statement and NN (the ISS number) is a decimal number from 01 to 20 used during loading to establish the linkage from the appropriate point in the corresponding ILS. The numbers and associated devices used in the subroutines provided by IBM are listed below.

Card/Paper Tape System and DM1 System

Label	Operation	F	T	
21	25	27	30	32 33 35 40
	ENT			NAME

Number* Device or Function

01	1442 Card Read Punch
02	Input Keyboard/Console Printer
03	1134 Paper Tape Reader; 1055 Paper Tape Punch
05	Single Disk Storage
06	1132 Printer
07	1627 Plotter
08	Synchronous Communications Adapter

NAME is a symbol that identifies an entry point for the associated subroutine. This symbol must be relocatable. All ENT statements for a given subroutine must be together and must precede all statements except LIBR, SPR, EPR, and comments statements. ENT, ISS, or ILS statements (see below) may not be used in the same subroutine.

*Numbers 09 through 20 are assignable by the user.

DM2 System

Number* Device or Function

01	1442 Card Read Punch; 1442 Card Punch
02	Input Keyboard/Console Printer
03	1134 Paper Tape Reader; 1055 Paper Tape Punch
04	2501 Card Reader
05	Single Disk Storage; 2310 Disk Storage
06	1132 Printer
07	1627 Plotter
08	Synchronous Communications Adaptor
09	1403 Printer
10	1231 Optical Mark Page Reader
11	2250 Display Unit, Model 4

* Numbers 12 through 20 are assignable by the user.

ISS - Define Interrupt Service Entry Point

IBM provides interrupt service subroutines (ISS) for all devices; however, the user is given the option of replacing or adding to these subroutines with his own. The ISS statement is used to define an entry point in an interrupt service subroutine and to establish interrupt linkages to the subroutine during loading. Only one entry point may be defined for each subroutine. The format of the ISS statement is shown below.

NOTE: User-assigned ISS numbers should start at twenty and proceed backwards in order to avoid conflict with IBM-assigned ISS numbers.

Label	Operation	F	T	
21	25	27	30	32 33 35 40 45
	ISS	NN		NAME L

L is a one-digit number required by the Card/Paper Tape Assembler to indicate the interrupt level(s) associated with the subroutine. The level numbers (0-5) can be listed in any order in columns 45, 50, 55, 60, 65, and 70 with the first appearing in 45, the second in 50, etc.

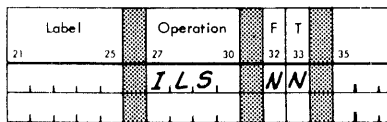
L is not used with the monitor system. Instead, LEVEL control cards are used with the subroutine being assembled, one card per interrupt level required (see the monitor system operator's manual).

An ISS statement must precede all statements except LIBR, SPR, EPR and comments statements.

Procedures for writing ISSs are provided in the subroutine library manual for the Card/Paper Tape and DM2 systems and in the operator's manual for the DM2 system.

ILS - Define Interrupt Level Subroutine

IBM provides interrupt level subroutines for the various I/O devices and their associated interrupt levels; however, the user may replace or add to these subroutines with his own. The ILS statement is used to define an interrupt level subroutine and to associate the subroutine with a specific interrupt level. The format of the ILS statement is shown below.



NN is the interrupt level number (00-05) associated with the interrupt level subroutine and is used during loading. The devices associated with each interrupt level are shown below:

<u>Interrupt Level</u>	<u>Device(s)</u>
00	1442 Card Read Punch (1442 Card Punch)
01	1132 Printer, Synchronous Communications Adaptor
02	Single Disk Storage (2310 Disk Storage)

<u>Interrupt Level</u>	<u>Device(s)</u>
03	1627 Plotter
04	Keyboard/Console Printer, 1442 Card Read Punch, 1134 Paper Tape Reader, 1055 Paper Tape Punch (2501 Card Reader, 1403 Printer, 1231 Optical Mark Page Reader)
05	PROGRAM STOP Key or Interrupt Run Mode.

NOTES: 1. The devices listed in parentheses are used with the DM2 system only.
 2. An ILS statement must precede all statements except SPR, EPR, and comments statements.

Procedures for writing interrupt level subroutines are provided in the subroutine library manual for the Card/Paper Tape and DM1 systems and in the operator's manual for the DM2 system.

CALL or LIBF Subroutines

The user may reasonably ask when to write a LIBF subroutine and when a CALL subroutine. Two factors influence the decision. First, the maximum limitation on the size of the LIBF Transfer Vector limits the number of unique LIBF subroutine entry points referenced by any core load. There is no such limitation on the CALL subroutine. However, the CALL subroutine references may require more core storage. The number of words of core required by CALL or LIBF references varies with the type of call as follows:

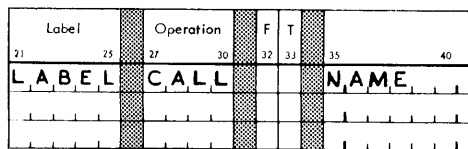
$$\text{LIBF linkage} = (\text{number of references to a LIBF name}) + 3$$

$$\text{CALL linkage} = \begin{cases} (\text{number of references to a CALL name} * 3) & \text{(Monitor)} \\ (\text{number of references to a CALL name} * 2) & \text{(Card/Paper Tape System)} \end{cases}$$

Thus, in the Monitor System any subroutine entry name referenced four or more times in a given core load will be more economically written as a LIBF subroutine.

CALL - Call Direct Reference Subroutine

A CALL statement is used to call any of the subroutines in the IBM Subroutine Library that are written not to utilize the LIBF Transfer Vector, whether user- or IBM-supplied. The format of the CALL statement is:



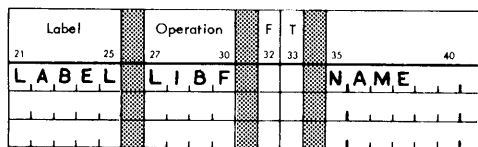
The name of the called subroutine is assembled into the object program, together with a unique code identifying the CALL. This name occupies two words.

As a Monitor core load is being built, the first word of the called subroutine name is replaced with an indirect BSI instruction and the second word with the address of the one-word entry for that subroutine entry point in the CALL Transfer Vector, which will contain the address of the subroutine entry point.

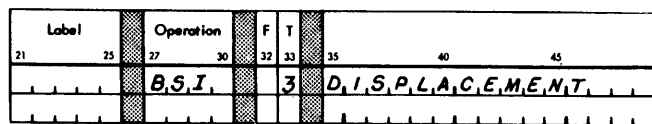
As a Card/Paper Tape System core load is being built, the first word of the called subroutine name is replaced with a long BSI instruction and the second word of the called subroutine name with the address of the subroutine entry point. Thus, a Transfer Vector is unnecessary in this case.

LIBF - Call TV (Transfer Vector) Reference Subroutine

A LIBF statement is used to call any of the subroutines in the IBM Subroutine Library that are written to utilize the LIBF Transfer Vector, whether user- or IBM-supplied. The format of the LIBF statement is:



The name of the called subroutine is assembled into the object program, together with a unique code identifying the call as a LIBF call. This is replaced during loading as follows:



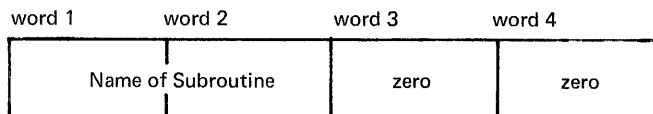
As indicated, Index Register 3 is required for the Transfer Vector and may be used by the user's program only if the following requirements are met:

- Index Register 3 must be saved and restored before the execution of the next LIBF statement.
- There must be no overlapping of I/O.
- There must be no possibility of interrupts from the SCA.

LIBF Subroutine Transfer Vector

To fully understand the use of the LIBF statement, the user should be familiar with the makeup of the Transfer Vector, which allows mainlines to communicate with relocatable subroutines (and relocatable subroutines to communicate with each other) without knowing where in core storage the subroutines are loaded. The Transfer Vector consists of three words for each entry point referred to by a LIBF statement.

To build the Transfer Vector the Monitor constructs a Load Table of four words for each unique subroutine entry point referred to by a LIBF statement. The name of the subroutine entry point is stored in the first two words. The last two words of the entry are initially zero.



As shown above, Index Register 3 will be used during execution to point to the first word of the LIBF Transfer Vector entry. The displacement of the indexed BSI instruction replacing the LIBF statement in the object code is the displacement from the middle of a maximum size Transfer Vector (255 words) to the first word of the referenced Transfer Vector entry.

After the mainline has been processed, each subroutine named in the Load Table is processed, and the address of the entry point is stored in the third word of the Load Table entry. The fourth word of the entry is used for several purposes during processing.

word 1	word 2	word 3	word 4
Name of Subroutine		Entry address of Subroutine	undefined

Any subroutines referenced by LIBF statements as the subroutines are being loaded are added to the Load Table (if they are not already there) and processed in the same manner. After the core load has been built, the Transfer Vector is created. The first word of each entry is set to zero, a long BSC instruction is placed in the second word, and the subroutine entry point address (from the corresponding Load Table entry) is stored in the third word.

word 1	word 2	word 3
zero	BSC L	Entry address of Subroutine

In the Card/Paper Tape System, the Load Table, which consists of three-word entries, becomes the Transfer Vector. As the LIBF statements are encountered during loading, the subroutine entry point name is stored to the first two words of the Transfer Vector entry. The third word of the Transfer Vector entry is initially zero.

word 1	word 2	word 3
Name of Subroutine		zero

The displacement of the indexed BSI instruction replacing the LIBF statement in the object code is the displacement from the middle of a maximum size Transfer Vector to the first word of the Transfer Vector entry.

As the subroutines are loaded, the third word of each Transfer Vector entry is replaced with the address of the subroutine entry point. The first word is cleared to zero, and a long BSC instruction is stored in the second word.

word 1	word 2	word 3
zero	BSC L	Entry address of Subroutine

LIBF Subroutine Execution

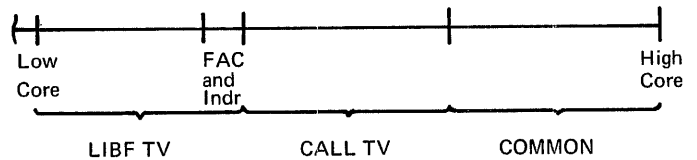
As the mainline is executed, the subroutine calls are encountered one by one. Each LIBF statement is now in the form of a BSI instruction. The BSI instruction stores the contents of the IAR into the first word of the Transfer Vector entry associated with the subroutine being called and then branches to the second word of the same Transfer Vector entry. As a result, the first word of the Transfer Vector entry contains the address of LIBF + 1.

word 1	word 2	word 3
LIBF + 1 Address	BSC L	Entry Address of Subroutine

Following execution of the BSI instruction, the BSC instruction in word 2 of the Transfer Vector entry is executed, transferring control to the subroutine. In order to access parameters and to return from a subroutine to the calling program, the subroutine must know the address of the Transfer Vector entry associated with the subroutine. This address was placed into the subroutine entry point + 2 as the core load was built. With this information the return address and the parameters associated with the LIBF statement may be determined by the subroutine itself. See Figure 5.

Size and Location of the Transfer Vector

The maximum size of the Monitor Transfer Vector is 85 three-word entries, including an entry for the Floating Accumulator (FAC) and the function indicators.



The Card/Paper Tape System Transfer Vector consists of a maximum of 256 words.

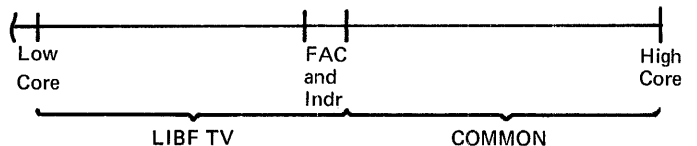


Figure 5. Sample LIBF Coding Sequence

Label	Operation	F	T	Operands & Remarks								
21	25	27	30	32 33	35	40	45	50	55	60	65	70
	LIBF				SUBR	MAIN PROGRAM CALL						
	DC				ARG	ADDRESS OF ARGUMENT						
	.											
	.											
	END				START	END OF MAIN PROGRAM						
	LIBR											
	ENT				SUBR	SUBROUTINE HEADER STATEMENT						
SUBR	STX	2			SAVE+1	SAVE INDEX REGISTER 2						
*	LDX	I2			*~*	THIS LOCATION WILL BE BLANK						
*						WHEN SUBR IS ASSEMBLED,						
*						BUT WILL BE FILLED WITH TV						
	LD	I2	0			ENTRY ADDRESS BY THE LOADER						
						FETCH ARGUMENT INTO ACC						
	.											
	.											
	.											
	MDX	2	+1			INCREMENT INDEX REGISTER 2						
	STX	2	LEAVE+1			*FOR AND STORE RETURN ADDR						
SAVE	LDX	I2	*~*			RESTORE INDEX REGISTER 2						
LEAVE	BSC	I	*~*			RETURN TO MAIN PROGRAM						
	END											

MONITOR ASSEMBLER STATEMENTS

In addition to the basic assembler statements, the IBM 1130 Monitor Assembler is provided with the following capabilities.

Disk Data Organization

- DSA - Define Sector Address
- FILE - Define Disk File (DM2 only)

Data Definition

- DMES - Define Message (DM2 only)
- DN - Define Name (DM2 only)

Linking

- LINK - Load and Execute Another Program
- EXIT - Return Control to Supervisor
- DUMP - Dump and Terminate (DM2 only)
- PDMP - Dump and Continue (DM2 only)

List Control

- HDNG - Print Heading on Each Page
- LIST - List Segments of Programs (DM2 only)
- SPAC - Space Listing (DM2 only)
- EJCT - Start New Page (DM2 only)

The label is defined as the current value of the Location Assignment Counter when the DSA statement is encountered. The symbol in the operand field must be the name of a data file or DCI program that is on disk both when the assembly is made and during execution.

The following statements illustrate the use of the DSA statement to read one sector of data. For a description of the disk calling sequences, see the system subroutines library manual.

Label	Operation	F	T	Operands & Res
21	25	27	30	32 33 35 40 45 50
	.			
	.			
	L.I.B.F			D.I.S.K.I
	D.C.			/I.O.O.O
	D.C.			I.O.A.R
	D.C.			E.R.R.O.R
	.			
I.O.A.R	D.S.A.			D.A.T.A.
	B.S.S.			3.1.9
	.			
	.			

The Assembler reserves three words in the object program for each DSA statement. These words are filled in by the Core Load Builder. For a data file they will contain:

- Word 1 - Length (in words)
- Word 2 - Sector Address, including the drive code
- Word 3 - Sector count of the file

For a program they will contain: \varnothing

- Word 1 - Length (in words)
- Word 2 - Sector Address, including the drive code
- Word 3 - Execution Address of the Program

If the area corresponding to the DSA statement is used as the I/O area for a disk read operation, the execution address of the program must be saved prior to the disk call to bring in the program. (The contents of the third word are destroyed by the incoming data).

DISK DATA ORGANIZATION STATEMENTS

DSA - Define Sector Address

The DSA statement allows the programmer to refer symbolically to a disk-stored data file or program stored in Disk Core Image format (DCI) without knowing the specific disk location of the data or program. The disk location of data files and programs can vary on disk because of deletions, but the DSA statement allows easy reference through the use of the symbolic name of the data file or program.

The format of the DSA statement is:

Label	Operation	F	T	Operands & Res
21	25	27	30	32 33 35 40 45 50
L.A.B.E.L	D.S.A.			N.A.M.E

The following statements illustrate the use of the DSA statement to supply the disk address of a one-sector program.

Label	Operation	F	T	Operands & Remarks
	.			
	L.D.			I,O.A.R.+2,
	S.T.O.			B,R,N,C,H,+1,
READ	L.I.B.F.			D,I,S,K,1
	D.C.			/1.0000
	D.C.			I,O.A.R.
	D.C.			E,R,R,O,R
C.A.L.L.	L.I.B.F.			D,I,S,K,1
	D.C.			/0.0000
	D.C.			I,O.A.R.
	M.D.X.			C,A,L,L
B,R,N,C,H	B.S.C.	L		0
	.			
	.			
	.			
I,O.A.R.	D.S.A.			P,R,G,R,M
	B.S.S.			3,1,9
	.			
	.			

The following statements can be added to the previously shown program call to call a second program and have it loaded to the same area as the first.

Label	Operation	F	T	Operands & Remarks
	L.D.			A,D,R,2,
	S.T.O.			I,O,A,R,
	L.D.			A,D,R,2,+1,
	S.T.O.			I,O,A,R,+1,
	L.D.			A,D,R,2,+2,
	S.T.O.			B,R,N,C,H,+1,
	M.D.X.			R,E,A,D,
A,D,R,2	D.S.A.			P,G,R,M,2,
	.			
	.			

The execution address of the second program can be different from the first, but the programs must be executable from the same locations. This requires a certain amount of planning before assembling the "overlay" programs.

Programming Considerations

The following considerations must be observed by the user who wishes to use the DSA statement to supply the disk address for programs.

- The called programs must be in DCI format.
- If the calling program is converted to DCI format, the data for the DSA statement is filled in during the core image conversion and will be fixed for all subsequent executions. Thus, if the referenced program or data files are subsequently moved, incorrect results will occur. Data files referenced by a Core Image program should be stored in the Fixed area.
- Any loading functions, such as the setting of Index Register 3, will have to be supplied by the calling program.

FILE - Define Disk File (DM2)

The FILE statement specifies to the Assembler the file identification, the number of file records in a file, and the size of each record in a disk data file that will be used with a particular mainline and its associated subprograms. The Assembler FILE statement allows the Assembler language user to define files so that they are similar to FORTRAN defined files.

As a core load is constructed by the Core Load Builder, the defined files are equated to data files already assigned in the User/Fixed Area or to files in Working Storage.

The FILE statement must not appear in a sub-program; it is permitted only in a relocatable mainline program. Therefore, all subprograms used by the mainline must use the defined files of the mainline. The format of the FILE statement is as follows:

Label	Operation	F	T	Operands & Remarks
l	FILE			a,m,n,u,v

where

l is any valid label (optional),

a is the file identification number, a decimal integer in the range 1-32767,

m is a decimal integer that defines the number of records in the file,

n is a decimal integer in the range 1-320 that defines the length (in words) of the longest record in the file,

U is a required constant, specifying that the file must be read/written with no data conversion,

v is the associated variable, the label of a core location (variable) defined elsewhere in the program.

FILE statements must precede all other statements except HDNG, EPR, SPR, EJCT, SPAC, and LIST in the source program. The label, if used, is assigned the location of the first word of the seven words generated (see list below). The Format and Tag fields are not used and should be left blank.

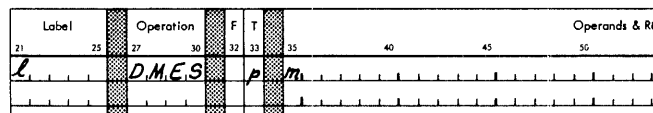
Each FILE statement causes the Location Assignment Counter to be incremented by seven. The data stored in these seven words, which constitute a DEFINE FILE Table entry in the object program is as follows:

<u>Word</u>	<u>Contents</u>
1	a, the file identification number
2	m, the number of records per file
3	n, the record length (in words)
4	The address of the associated variable, v.
5	Zero. This word is filled by the Core Load Builder with the sector address of the data file. This address is relative to the address of Working Storage (with bit zero set to one) for Working Storage files and is absolute, including the drive code, for User/Fixed area files.
6	r, the number of records per sector. The number, computed by the Assembler, is the quotient of $\frac{320}{n}$ (remainder ignored)
7	b, the number of disk blocks per file. This number, computed by the Assembler, is the quotient of $\frac{16(m)}{r}$

DATA DEFINITION STATEMENTS

DMES - Define Message (DM2)

The DMES statement is used to store a message within a program in a form that is acceptable to the printer output subroutines. The format of the DMES statement follows:



where

l is any valid label (optional),

p is the printer type code,

m is any string of valid message and control characters.

If a label is present, it is assigned to the location of the first word generated. The Tag field (column 33) is used to specify the printer type code:

<u>Tag</u>	<u>Printer</u>
b or 0	Console Printer
1	1403 Printer
2	1132 Printer

If the Tag field (printer type code) contains a character other than blank, zero, one, or two, an error results and the message is stored two EBCDIC characters per word.

The Operand field contains the control and message characters. Remarks are permitted only after an 'E' or 'b' control character.

The output generated by one DMES statement cannot exceed 60 words (120 characters). If an odd number of characters is generated, the last word is filled in with a blank, except when the statement ends with 'b'. In this case, the first character of the next DMES statement is used to fill out the word.

Control characters are used to specify certain printer operations and to define message parameters. Each control character is actually two characters, the first of which is always an apostrophe. The apostrophe (5-8 punch in IBM Card Code) is a control

delimiter and therefore is not included in the character count. The control characters and their functions or meanings are as follows:

Control Character	Function or Meaning
'X	Blank (or space)
'T	Tabulate
'D	Backspace
'B	Print black
'A	Print red
'S	Space (or blank)
'R	Carriage return
'L	Line feed
'F	Repeat following character
'E	End of message
'b	(b=blank) continues text with next DMES statement

All the above characters can be used when the printer is the Console Printer. Only 'E', 'F', 'S', 'X and 'b are valid control characters when the 1132 or 1403 Printer is specified; any other control characters are considered as errors.

The characters 'X and 'S are interchangeable. A blank character is generated for either 'X or 'S if the 1132 or 1403 Printer is specified; a space is generated for either 'X or 'S if the Console Printer is specified.

The character 'F (repeat following character) refers only to message characters. The control characters themselves, except 'A, 'B, 'E, and 'b, can be repeated up to 99 times by inserting a number (1-99) between the apostrophe and unique control definition character. For example, '32S results in 32 space characters being inserted in the generated message.

The character 'E is used to designate the end of the message line. The character 'b is used to designate that the message is continued on the following DMES statement. If neither 'E nor 'b is included, 'E is assumed to follow column 71. DMES statements that end with 'b must be followed by another DMES statement.

Text apostrophes are generated by writing two successive apostrophes.

The message characters can be any valid character for the printer being used. Invalid characters are replaced with blanks.

The following example illustrates the DMES statement.

Assembler input:

Label	Operation	F	T	Operands & Remarks
	DMES			'RSAMPLE PROGRAM'S'
	DMES			OUTPUT
	DMES			'2R'9S'1'9S'2'9S'3'9S'4'E
	DMES			'R12345678901234567890'
	DMES			012345678901234567890'E
	DMES			'2R'7X'7F-'4DF(X)'
	DMES			'7X'8F-'5DF'(X)'

Printed output:

```

SAMPLE PROGRAM'S OUTPUT
1           2           3           4
1234567890123456789012345678901234567890
          F(X)           F'(X)
  
```

Note that the device code specified in the preceding example is blank in order to generate a message for the Console Printer.

DN — Define Name

The Define Name statement is used to convert a name specified in the Operand field of the statement to a name in Name Code in the object program. The format of this statement is shown below:

Label	Operation	F	T	Operands & Remarks
I	DN			n

where

I is any valid label (optional),

n is any valid label or name.

Name Code is truncated packed EBCDIC. The two high order bits of each character in the name are removed and the five characters are packed into the right thirty bits of two words.

```

00  C      H      A      R      S
XX|XX XXXX|XXXX XX|XX XX|XXXX XX|XX XXXX|
  
```

If a label is used, the address assigned to it is the location of the first word of the two words generated and is equal to the current value of the Location Assignment Counter. Columns 32 and 33 must be blank. The operand can have up to five characters that comply with the rules for writing symbols. The name to be converted must be left-justified in the Operand field. If remarks are used, one blank must be left between the operand and the remarks. The Location Assignment Counter is incremented by two for this statement.

LINKING STATEMENTS

LINK — Load Link Program

In the assembler language, the LINK statement is used to cause another core load to be loaded and executed. Only COMMON of the current core load is saved. The program loaded and executed must be specified by name. The format of the LINK statement is:

1. A symbol or blanks in the label field
2. The mnemonic, LINK, in columns 27-30
3. A valid program name in the operand field

The label of the LINK pseudo-operation is defined as the current value of the Location Assignment Counter when the LINK statement is encountered; this value is the address of the first word generated by the LINK statement.

The operand field contains a valid program name (one to five alphameric characters), left-justified in the field. The name must be present in LET/FLET at execution time. The LINK statement causes four words to be generated in the object program. The first two words contain a long BSI instruction, which branches to a specified location within the Skeleton Supervisor. The next two words contain the program name in Name Code (see DN - Define Name). The Core Image Loader uses the core load name and begins the process required to load the new core load.

EXIT — Return to Supervisor

In the assembler language, the EXIT statement is used to return control to the Supervisor. The format of the EXIT statement is:

1. A symbol or blanks in the label field
2. The mnemonic, EXIT, in columns 27-30

The label of the EXIT statement is defined as the current value of the Location Assignment Counter when the EXIT statement is encountered; this value is the address of the instruction generated by an EXIT statement. The operand field is ignored and can therefore be used for remarks.

The EXIT statement causes a short branch instruction to be generated in the object program. The instruction branches to a fixed location in the Skeleton Supervisor. During execution, the branch is executed and control is returned to the Supervisor. The EXIT statement should be the last logical statement in a program.

DUMP — Dump and Terminate Execution

The DUMP statement provides an entry to the System DUMP program, which prints the contents of core storage on the principal print device in hexadecimal format.

The DUMP statement allows for flexible specification of the upper and lower limits to be dumped without altering core storage. After core has been dumped between the limits specified, the System Dump returns control to the calling program, at which point a CALL EXIT is executed. The DUMP statement is written as follows:

Label	Operation	F	T	Operands & Rem
21	25	27	30	32 33 35 40 45 50
<i>l</i>	<i>DUMP</i>			<i>a₁, b₁, f</i>

where

l is any valid label (optional),

a is any valid expression specifying the lowest-addressed core location to be dumped,

b is any valid expression specifying the highest-addressed core location to be dumped,

f is the dump format code (either blank or zero). The dump is always in hexadecimal format.

The label, if used, is assigned the location of the first of the six words generated (see list below). The Tag and Format fields must be left blank.

A DUMP statement causes the Location Assignment Counter to be incremented by six. The data stored in these six words is as follows:

Word	Contents
1 } 2 }	A long (two-word) BSI to the DUMP entry point in the Skeleton Supervisor
3	The format indicator (always zero)
4	The starting address of the core dump
5	The ending location of the core dump
6	A short branch to the EXIT entry point in the Skeleton Supervisor

If no address is specified for word 3, the dump starts in location zero. If no address is specified for word 4, the dump continues to the end of core.

A DUMP statement can be used at any point in a program; however, the user is reminded that DUMP causes a terminal DUMP to be printed. At the completion of the dump printout, the branch to EXIT is executed, thus transferring control to the Skeleton Supervisor for processing of the next job or subjob.

The format of the DUMP program output is as follows:

```
AAAA xxxx xxxx xxxx || xxxx xxxx xxxx
```

The contents (xxxx) of 16 core storage locations are printed per line. At the left is the address (AAAA) of the first location printed on that line.

PDMP — Dump and Continue Execution

The PDMP statement provides the ability to dump core storage between specified limits and to continue execution. The core dump is printed on the principal print device without altering core. The PDMP statement is specified in the same way as DUMP, except that PDMP appears in columns 27-30 instead of DUMP.

The PDMP statement is translated by the Assembler into a long BSI to the DUMP entry point in the Skeleton Supervisor. The parameters (operands) are converted as described in the DUMP statement (see above) except that the exit to the Supervisor is not generated for PDMP.

Upon completion of the printout of the core dump, control is returned to the next instruction following the PDMP statement to continue execution.

LIST CONTROL STATEMENTS

The list control statements — HDNG, LIST, SPAC, and EJCT — provide the user with the means to control and identify the assembler output listing.

HDNG — Heading

The HDNG statement is used to specify a one line page heading for a printed listing. The heading line consists of the data in the Operand-Remarks field.

The format of the HDNG statement is as follows.

Label	Operation	F	T	Operands & Remarks	
21	25	27	30	32 33 35	40 45 50
		HDNG			PAGE HEADING

Multiple HDNG statements may be used thus allowing different sections of a listing to have different page headings.

When the 1132 or 1403 is the principal printer, the HDNG statement causes the listing to be ejected to a new page and the heading is printed. The same heading is repeated at the top of each succeeding page until a new HDNG statement is encountered.

When the Console Printer is the principal printer, the heading line is preceded by five line feeds and followed by a single line feed, and otherwise functions as a comments statement.

LIST — List Segments of Program

The LIST statement allows the user to list certain segments of a program on the principal printer and avoid listing other segments. The three variations of the LIST statement are shown below:

Label	Operation	F	T	Operands & Remarks	
21	25	27	30	32 33 35	40 45 50
		LIST			
		LIST			ON
		LIST			OFF

The Label, Tag, and Format fields are not used with the LIST statement and should be left blank. The Operand field may be left blank or may contain the operand ON or OFF.

The LIST statement does not cause the Location Assignment Counter to be incremented.

If a LIST statement with the operand ON is encountered, the following statements, up to the next LIST statement, are listed by the Assembler.

If a LIST statement with no operand is encountered, the Assembler assumes an operand depending on the use of the LIST control record. If the LIST control record preceded the assembly, the ON operand is assumed and the Assembler acts accordingly. If the LIST control record did not precede the assembly, the OFF operand is assumed and the Assembler acts accordingly.

SPAC — Space Listing

The SPAC statement is used to insert one or more blank lines in the listing immediately following the SPAC statement. The format of the SPAC statement is as follows:

Label	Operation	F	T	Operands & Re
21	27	30	32 33	35 40 45 50
	SPAC			e

where e is any valid positive expression.

The Label, Format, and Tag fields are not used and should be left blank.

The number of blank lines inserted in the listing is determined by the operand in the statement. The

operand can be any valid expression. The operand (expression) value must be positive; otherwise, the Assembler ignores the statement.

When the number of blank lines specified exceeds the number of lines left on the page, the page is spaced to the bottom, a restore occurs, a new heading is printed, and spacing is resumed until the number of blank lines specified has been exhausted.

The SPAC statement does not cause the Location Assignment Counter to be incremented.

EJCT — Start New Page

The EJCT statement causes the next line of the listing to appear at the top of a new page following the page heading. The format of the EJCT statement is as follows:

Label	Operation	F	T	Operands & Re
21	27	30	32 33	35 40 45 50
	EJCT			

The Label, Tag, Format, and Operand fields are not used and should be left blank.

A page overflow occurs immediately following the EJCT statement. EJCT statements may be used in succession to obtain blank pages (except for the headings printed).

The EJCT statement does not cause the Location Assignment Counter to be incremented.

Hexadecimal Notation

In hexadecimal notation, each digit represents a four-bit binary value. This means that a 16-bit word in the Processor-Controller can be expressed as four hexadecimal digits. The binary — hexadecimal — decimal correspondence is defined as follows:

<u>Binary</u>	<u>Hexadecimal</u>	<u>Decimal</u>
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Extended Binary Coded Decimal Interchange Code (EBCDIC)

In the EBCDIC code, each character is represented by a unique configuration of eight binary bits. In

the table that follows, each EBCDIC character is expressed as two hexadecimal digits.

IBM Card Code

In the IBM Card Code, each character represents a 12-bit card-column image. In the table that follows, each card code character is expressed as four hexadecimal digits and as the card-column image.

Paper Tape Transmission Code, 8 Channel (PTTC/8)

In the PTTC/8 code, each character is represented by a unique configuration of a case shift, plus an eight-bit code. The case shift can be common to more than one character and need be inserted only when a case shift change is necessary. In the table that follows, each character is expressed as two hexadecimal digits, followed by the case shift in parentheses.

1132 Printer EBCDIC Subset Hex Code

In the 1132 Printer EBCDIC subset hex code, each character is represented by a unique configuration of eight bits. In the table that follows, each 1132 Printer character is expressed as two hexadecimal digits.

Console Printer Hex Code

In the Console Printer hexadecimal code each character is represented as two hexadecimal digits.

1403 Printer Hex Code

In the 1403 Printer hexadecimal code each character is represented as two hexadecimal digits.

Ref No.	EBCDIC		IBM Card Code					Graphics and Control Names	1132 Printer EBCDIC Subset Hex	PTTC/8 Hex U-Upper Case L-Lower Case	Console Printer Hex Notes	1403 Printer Hex	
	Binary 0123 4567	Hex	12	11	0	9	8						7-1
0	0000	0000	00	12	0	9	8	1	8030	NUL			
1	0001	0001	01	12		9		1	9010				
2	0010	0010	02	12		9		2	8810				
3	0011	0011	03	12		9		3	8410				
4	0100	0100	04	12		9		4	8210				
5*	0101	0101	05	12		9		5	8110				
6*	0110	0110	06	12		9		6	8090				
7*	0111	0111	07	12		9		7	8050				
8	1000	1000	08	12		9	8		8030				
9	1001	1001	09	12		9	8	1	9030				
10	1010	1010	0A	12		9	8	2	8830				
11	1011	1011	0B	12		9	8	3	8430				
12	1100	1100	0C	12		9	8	4	8230				
13	1101	1101	0D	12		9	8	5	8130				
14	1110	1110	0E	12		9	8	6	8080				
15	1111	1111	0F	12		9	8	7	8070				
16	0001	0000	10	12	11	9	8	1	D030	RES Restore NL New Line BS Backspace IDL Idle	4C (U/L) DD (U/L) 5E (U/L)	05 ② 81 ③ 11	
17	0001	0001	11		11	9		1	5010				
18	0010	0010	12		11	9		2	4810				
19	0011	0011	13		11	9		3	4410				
20*	0100	0100	14		11	9		4	4210				
21*	0101	0101	15		11	9		5	4110				
22*	0110	0110	16		11	9		6	4090				
23	0111	0111	17		11	9		7	4050				
24	1000	1000	18		11	9	8		4030				
25	1001	1001	19		11	9	8	1	5030				
26	1010	1010	1A		11	9	8	2	4830				
27	1011	1011	1B		11	9	8	3	4430				
28	1100	1100	1C		11	9	8	4	4230				
29	1101	1101	1D		11	9	8	5	4130				
30	1110	1110	1E		11	9	8	6	4080				
31	1111	1111	1F		11	9	8	7	4070				
32	0010	0000	20		11	0	9	8	1	7030	BYP Bypass LF Line Feed EOB End of Block PRE Prefix	3D (U/L) 3E (U/L)	03
33	0001	0001	21			0	9	1	3010				
34	0010	0010	22			0	9	2	2810				
35	0011	0011	23			0	9	3	2410				
36	0100	0100	24			0	9	4	2210				
37*	0101	0101	25			0	9	5	2110				
38*	0110	0110	26			0	9	6	2090				
39	0111	0111	27			0	9	7	2050				
40	1000	1000	28			0	9	8	2030				
41	1001	1001	29			0	9	8	1	3030			
42	1010	1010	2A			0	9	8	2	2830			
43	1011	1011	2B			0	9	8	3	2430			
44	1100	1100	2C			0	9	8	4	2230			
45	1101	1101	2D			0	9	8	5	2130			
46	1110	1110	2E			0	9	8	6	2080			
47	1111	1111	2F			0	9	8	7	2070			
48	0011	0000	30	12	11	0	9	8	1	F030	PN Punch On RS Reader Stop UC Upper Case EOT End of Trans.	0D (U/L) 0E (U/L)	09 ④
49	0001	0001	31			9		1	1010				
50	0010	0010	32			9		2	0810				
51	0011	0011	33			9		3	0410				
52	0100	0100	34			9		4	0210				
53*	0101	0101	35			9		5	0110				
54*	0110	0110	36			9		6	0090				
55	0111	0111	37			9		7	0050				
56	1000	1000	38			9	8		0030				
57	1001	1001	39			9	8	1	1030				
58	1010	1010	3A			9	8	2	0830				
59	1011	1011	3B			9	8	3	0430				
60	1100	1100	3C			9	8	4	0230				
61	1101	1101	3D			9	8	5	0130				
62	1110	1110	3E			9	8	6	0080				
63	1111	1111	3F			9	8	7	0070				

NOTES: Typewriter Output

- ① Tabulate
- ② Shift to black

- ③ Carrier Return
- ④ Shift to red

* Recognized by all Conversion subroutines
Codes that are not asterisked are recognized only by the SPEED subroutine

Ref No.	EBCDIC			IBM Card Code				Graphics and Control Names	1132 Printer EBCDIC Subset Hex	PTTC/8 Hex U-Upper Case L-Lower Case	Console Printer Hex	1403 Printer Hex		
	Binary	Hex	Rows	Hex										
	0123	4567	12	11	0	9	8	7-1						
64*	0100	0000	40	no punches				0000	blank	40	10 (U/L)	21	7F	
65		0001	41	12	0	9	1	8010						
66		0010	42	12	0	9	2	A810						
67		0011	43	12	0	9	3	A410						
68		0100	44	12	0	9	4	A210						
69		0101	45	12	0	9	5	A110						
70		0110	46	12	0	9	6	A090						
71		0111	47	12	0	9	7	A050						
72		1000	48	12	0	9	8	A030						
73		1001	49	12			8	1	9020					
74*		1010	4A	12			8	2	8820	‡ (period)	20 (U)	02		
75*		1011	4B	12			8	3	8420		6B (L)	00	6E	
76*		1100	4C	12			8	4	8220	<	02 (U)	DE		
77*		1101	4D	12			8	5	8120	(19 (U)	FE	57	
78*		1110	4E	12			8	6	80A0	+	70 (U)	DA	6D	
79*		1111	4F	12			8	7	8060	(logical OR)	3B (U)	C6		
80*	0101	0000	50	12				8000	&	50	70 (L)	44	15	
81		0001	51	12	11		9	1	D010					
82		0010	52	12	11		9	2	C810					
83		0011	53	12	11		9	3	C410					
84		0100	54	12	11		9	4	C210					
85		0101	55	12	11		9	5	C110					
86		0110	56	12	11		9	6	C090					
87		0111	57	12	11		9	7	C050					
88		1000	58	12	11		9	8	C030					
89		1001	59	11			8	1	5020	!		42		
90*		1010	5A	11			8	2	4820	!	5B (U)			
91*		1011	5B	11			8	3	4420	\$	5B (L)	40	62	
92*		1100	5C	11			8	4	4220	*	5C	D6	23	
93*		1101	5D	11			8	5	4120)	5D	F6	2F	
94*		1110	5E	11			8	6	40A0	;		D2		
95*		1111	5F	11			8	7	4060	† (logical NOT)		F2		
96*	0110	0000	60	11				4000	- (dash)	60	40 (L)	84	61	
97*		0001	61		0			1	3000	/	61	31 (L)	BC	4C
98		0010	62	11	0	9		2	6810					
99		0011	63	11	0	9		3	6410					
100		0100	64	11	0	9		4	6210					
101		0101	65	11	0	9		5	6110					
102		0110	66	11	0	9		6	6090					
103		0111	67	11	0	9		7	6050					
104		1000	68	11	0	9	8		6030					
105		1001	69		0		8	1	3020					
106		1010	6A	12	11				C000					
107*		1011	6B		0		8	3	2420	, (comma)	6B	3B (L)	80	16
108*		1100	6C		0		8	4	2220	%		15 (U)	06	
109*		1101	6D		0		8	5	2120	_ (underscore)		40 (U)	BE	
110*		1110	6E		0		8	6	20A0	>		07 (U)	46	
111*		1111	6F		0		8	7	2060	?		31 (U)	86	
112	0111	0000	70	12	11	0		E000						
113		0001	71	12	11	0	9	1	F010					
114		0010	72	12	11	0	9	2	E810					
115		0011	73	12	11	0	9	3	E410					
116		0100	74	12	11	0	9	4	E210					
117		0101	75	12	11	0	9	5	E110					
118		0110	76	12	11	0	9	6	E090					
119		0111	77	12	11	0	9	7	E050					
120		1000	78	12	11	0	9	8	E030					
121		1001	79				8	1	1020					
122*		1010	7A				8	2	0820	:		04 (U)	82	
123*		1011	7B				8	3	0420	#		0B (L)	C0	
124*		1100	7C				8	4	0220	@		20 (L)	04	
125*		1101	7D				8	5	0120	' (apostrophe)	7D	16 (U)	E6	0B
126*		1110	7E				8	6	00A0	=	7E	01 (U)	C2	4A
127*		1111	7F				8	7	0060	"		0B (U)	E2	

Ref No.	EBCDIC		Hex	IBM Card Code					Hex	Graphics and Control Names	1132 Printer EBCDIC Subset Hex	PTTC/8 Hex U-Upper Case L-Lower Case	Console Printer Hex	1403 Printer Hex
	Binary 0123	4567		12	11	0	9	8						
128	1000	0000	80	12		0		8	1	B020				
129		0001	81	12		0			1	B000				
130		0010	82	12		0			2	A800	a			
131		0011	83	12		0			3	A400	b			
132		0100	84	12		0			4	A200	c			
133		0101	85	12		0			5	A100	d			
134		0110	86	12		0			6	A080	e			
135		0111	87	12		0			7	A040	f			
136		1000	88	12		0		8		A020	g			
137		1001	89	12		0	9			A010	h			
138		1010	8A	12		0		8	2	A820	i			
139		1011	8B	12		0		8	3	A420				
140		1100	8C	12		0		8	4	A220				
141		1101	8D	12		0		8	5	A120				
142		1110	8E	12		0		8	6	A0A0				
143		1111	8F	12		0		8	7	A060				
144	1001	0000	90	12	11			8	1	D020				
145		0001	91	12	11				1	D000	i			
146		0010	92	12	11				2	C800	k			
147		0011	93	12	11				3	C400	l			
148		0100	94	12	11				4	C200	m			
149		0101	95	12	11				5	C100	n			
150		0110	96	12	11				6	C080	o			
151		0111	97	12	11				7	C040	p			
152		1000	98	12	11			8		C020	q			
153		1001	99	12	11		9			C010	r			
154		1010	9A	12	11			8	2	C820				
155		1011	9B	12	11			8	3	C420				
156		1100	9C	12	11			8	4	C220				
157		1101	9D	12	11			8	5	C120				
158		1110	9E	12	11			8	6	C0A0				
159		1111	9F	12	11			8	7	C060				
160	1010	0000	A0		11	0		8	1	7020				
161		0001	A1		11	0			1	7000	s			
162		0010	A2		11	0			2	6800	t			
163		0011	A3		11	0			3	6400	u			
164		0100	A4		11	0			4	6200	v			
165		0101	A5		11	0			5	6100	w			
166		0110	A6		11	0			6	6080	x			
167		0111	A7		11	0			7	6040	y			
168		1000	A8		11	0		8		6020	z			
169		1001	A9		11	0	9			6010				
170		1010	AA		11	0		8	2	6820				
171		1011	AB		11	0		8	3	6420				
172		1100	AC		11	0		8	4	6220				
173		1101	AD		11	0		8	5	6120				
174		1110	AE		11	0		8	6	60A0				
175		1111	AF		11	0		8	7	6060				
176	1011	0000	B0	12	11	0		8	1	F020				
177		0001	B1	12	11	0			1	F000				
178		0010	B2	12	11	0			2	E800				
179		0011	B3	12	11	0			3	E400				
180		0100	B4	12	11	0			4	E200				
181		0101	B5	12	11	0			5	E100				
182		0110	B6	12	11	0			6	E080				
183		0111	B7	12	11	0			7	E040				
184		1000	B8	12	11	0		8		E020				
185		1001	B9	12	11	0	9			E010				
186		1010	BA	12	11	0		8	2	E820				
187		1011	BB	12	11	0		8	3	E420				
188		1100	BC	12	11	0		8	4	E220				
189		1101	BD	12	11	0		8	5	E120				
190		1110	BE	12	11	0		8	6	E0A0				
191		1111	BF	12	11	0		8	7	E060				

Ref No.	EBCDIC		IBM Card Code				Graphics and Control Names	1132 Printer EBCDIC Subset Hex	PTTC/8 Hex U-Upper Case L-Lower Case	Console Printer Hex	1403 Printer Hex
	Binary	Hex	Rows			Hex					
	0123 4567		12 11 0 9 8 7-1								
192	1100	0000	C0	12	0		A000	(+ zero)			
193*		0001	C1	12		1	9000	A			64
194*		0010	C2	12		2	8800	B	C1	61 (U)	3C or 3E
195*		0011	C3	12		3	8400	C	C2	62 (U)	18 or 1A
196*		0100	C4	12		4	8200	D	C3	73 (U)	1C or 1E
197*		0101	C5	12		5	8100	E	C4	64 (U)	30 or 32
198*		0110	C6	12		6	8080	F	C5	75 (U)	34 or 36
199*		0111	C7	12		7	8040	G	C6	76 (U)	10 or 12
200*		1000	C8	12		8	8020	H	C7	67 (U)	14 or 16
201*		1001	C9	12		9	8010	I	C8	68 (U)	24 or 26
202		1010	CA	12	0	9 8 2	A830		C9	79 (U)	20 or 22
203		1011	CB	12	0	9 8 3	A430				
204		1100	CC	12	0	9 8 4	A230				
205		1101	CD	12	0	9 8 5	A130				
206		1110	CE	12	0	9 8 6	A080				
207		1111	CF	12	0	9 8 7	A070				
208	1101	0000	D0	11	0		6000	(- zero)			
209*		0001	D1	11		1	5000	J	D1	51 (U)	7C or 7 E
210*		0010	D2	11		2	4800	K	D2	52 (U)	58 or 5A
211*		0011	D3	11		3	4400	L	D3	43 (U)	5C or 5E
212*		0100	D4	11		4	4200	M	D4	54 (U)	70 or 72
213*		0101	D5	11		5	4100	N	D5	45 (U)	74 or 76
214*		0110	D6	11		6	4080	O	D6	46 (U)	50 or 52
215*		0111	D7	11		7	4040	P	D7	57 (U)	54 or 56
216*		1000	D8	11		8	4020	Q	D8	58 (U)	64 or 66
217*		1001	D9	11		9	4010	R	D9	49 (U)	60 or 62
218		1010	DA	12	11	9 8 2	C830				
219		1011	DB	12	11	9 8 3	C430				
220		1100	DC	12	11	9 8 4	C230				
221		1101	DD	12	11	9 8 5	C130				
222		1110	DE	12	11	9 8 6	C080				
223		1111	DF	12	11	9 8 7	C070				
224	1110	0000	E0		0	8 2	2820				
225		0001	E1		11	0 9 1	7010				
226*		0010	E2		0	2	2800	S	E2	32 (U)	98 or 9A
227*		0011	E3		0	3	2400	T	E3	23 (U)	9C or 9E
228*		0100	E4		0	4	2200	U	E4	34 (U)	B0 or B2
229*		0101	E5		0	5	2100	V	E5	25 (U)	B4 or B6
230*		0110	E6		0	6	2080	W	E6	26 (U)	90 or 92
231*		0111	E7		0	7	2040	X	E7	37 (U)	94 or 96
232*		1000	E8		0	8	2020	Y	E8	38 (U)	A4 or A6
233*		1001	E9		0	9	2010	Z	E9	29 (U)	A0 or A2
234		1010	EA		11	0 9 8 2	6830				
235		1011	EB		11	0 9 8 3	6430				
236		1100	EC		11	0 9 8 4	6230				
237		1101	ED		11	0 9 8 5	6130				
238		1110	EE		11	0 9 8 6	6080				
239		1111	EF		11	0 9 8 7	6070				
240*	1111	0000	F0		0		2000	0	F0	1A (L)	C4
241*		0001	F1			1	1000	1	F1	01 (L)	FC
242*		0010	F2			2	0800	2	F2	02 (L)	D8
243*		0011	F3			3	0400	3	F3	13 (L)	DC
244*		0100	F4			4	0200	4	F4	04 (L)	F0
245*		0101	F5			5	0100	5	F5	15 (L)	F4
246*		0110	F6			6	0080	6	F6	16 (L)	D0
247*		0111	F7			7	0040	7	F7	07 (L)	D4
248*		1000	F8			8	0020	8	F8	08 (L)	E4
249*		1001	F9			9	0010	9	F9	19 (L)	E0
250		1010	FA	12	11	0 9 8 2	E830				
251		1011	FB	12	11	0 9 8 3	E430				
252		1100	FC	12	11	0 9 8 4	E230				
253		1101	FD	12	11	0 9 8 5	E130				
254		1110	FE	12	11	0 9 8 6	E080				
255		1111	FF	12	11	0 9 8 7	E070				

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
D0	D1	D2	D3	D4	D5	D6	D7	D8	DA	DB	DC	DD	DE	DF	FF
E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
D0	D1	D2	D3	D4	D5	D6	D7	D8	DA	DB	DC	DD	DE	DF	FF
E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF

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

extended chart to the right.

For conversion of decimal values beyond the main table, deduct the largest number in the table at the right that will yield a positive result. The related digit is the high-order hexadecimal digit. Determine the three remaining hexadecimal digits by converting the product of the above subtraction in the main table.

The table to the left gives the decimal, binary, and hexadecimal coding for the full range of four binary bits, from zero through F₁₆ and 15₁₀.

To convert a four-digit hexadecimal number to decimal, determine the decimal value of the three low-order hexadecimal digits in the main table, and add the value for the high-order digit, as shown in the

Dec	Bin	Hex	Bin	Hex
0	0000	0	8	1000
1	0001	1	9	1001
2	0010	2	10	1010
3	0011	3	11	1011
4	0100	4	12	1100
5	0101	5	13	1101
6	0110	6	14	1110
7	0111	7	15	1111

APPENDIX C: ASSEMBLER MNEMONICS AND ERROR CODES FOR 2250 ORDERS

This appendix provides information about the 2250 orders: their names, mnemonics, and the corresponding 2250 model 4 orders; coding formats; and notes pertaining to using the orders. The information about each order is presented as follows:

Assembler Order (2250 Order)

Coding format and Assembler mnemonic

Notes: Operand limitations, restrictions, significance of format and tag fields, etc.

Functional descriptions of the 2250, model 4 orders are contained in the publication IBM 1130 Component Description: IBM 2250 Display Unit Model 4, Form A27-2723. This appendix is to be used in conjunction with that publication.

Table 2 lists the codes used to identify errors encountered during assembly of the orders, the causes of the errors, and the actions taken by the Assembler.

Set Graphic Mode Vector (Set Graphic Mode)

21	27
[label]	SGMV

Notes: The graphic mode vector must be established before generating lines. Vector mode is set automatically if no graphic mode has been previously set (see the 2250 model 4 Component Description publication).

Set Graphic Mode Point (Set Graphic Mode)

21	27
[label]	SGMP

Notes: The graphic mode point must be established before generating points.

Set Character Mode Basic (Set Character Mode)

21	27
[label]	SCMB

Notes: A character mode must be established before executing a character stroke or entering a character stroke subroutine.

Set Character Mode Large (Set Character Mode)

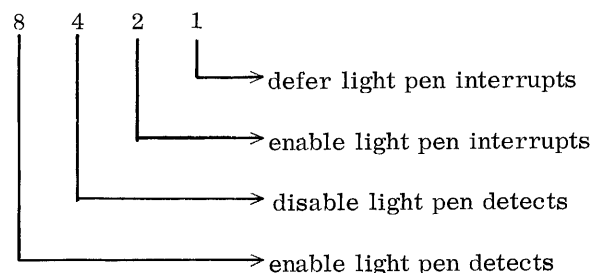
21	27
[label]	SCML

Notes: The notes for SCMB also apply for SCML.

Set Pen Mode (Set Pen Mode)

21	27	35
[label]	SPM	/hex digit or equivalent

Notes: The operand may be any hex digit or any valid absolute Assembler expression in the range 0 to F. The bit pattern of a hex digit and the effect of a 1 in a bit position are as follows:



For example, /9 (1001) defers interrupts and enables detects. Hex values 0, 3, C, and F result in no-operation.

Start Regeneration Timer (Start Timer)

21	27
[label]	STMR

Notes: The STMR order should be the first order in an order program. Its use is required for accepting keyboard attentions and setting the status of the light pen switch.

Store Revert Register (Store Revert Register)

21	27
[label]	SRVT

Notes: The SRVT order is used for return linkage when multiple levels of subroutines are used. A graphic branch indirect through the second word of the SRVT order returns control to the calling program. The SRVT order should appear in the subroutine preceding any graphic branch orders within the subroutine.

Revert (Revert)

21	27
[label]	RVT

Notes: If an order subroutine does not contain any graphic branch orders, the RVT order can be used to return control to the main order program at the order following the branch to the subroutine. If the subroutine is two or more levels from the main order program, the RVT order does not pass control to the main order program.

Graphic No-operation (Set Pen Mode)

21	27
[label]	GNOP

Notes: GNOP is assembled as an SPM order with an operand of hexadecimal 00. It can be used to reserve a single word in an order stream for later modification.

Move* Beam Incremental (Incremental XY)

21	27	35
[label]	MBI	X, Y

Notes: The x- and y-coordinates may be any valid absolute Assembler expressions, but must be in the range +63 to -64.

*The word "move" here and in the following orders relates to blanked beam movement.

Draw* Beam Incremental (Incremental XY)

21	27	35
[label]	DBI	X, Y

Notes: In vector mode, beam movement is unblanked; in point mode, only the end point is unblanked. The notes for MBI also apply.

*The word "draw" here and in following orders relates to unblanked beam movement.

Move Beam Absolute (Absolute XY)

21	27	35
[label]	MBA	X, Y

Notes: The x- and y-coordinates may be any valid absolute Assembler expressions, but must be in the range 0 to 1023.

Draw Beam Absolute (Absolute XY)

21	27	35
[label]	DBA	X, Y

Notes: In vector mode, beam movement is unblanked; in point mode, only the end point is unblanked. The notes for MBA also apply.

Move Beam Absolute X (Absolute X/Y)

21	27	35
[label]	MBAX	X

Notes: The operand may be any valid absolute Assembler expression, but must be in the range 0 to 1023.

Move Beam Absolute Y (Absolute X/Y)

21	27	35
[label]	MBAY	Y

Notes: The notes for MBAX also apply here.

Draw Beam Absolute X (Absolute X/Y)

21	27	35
[label]	DBAX	X

Notes: In vector mode, beam movement is unblanked; in point mode, only the end point is unblanked. The notes for MBAX also apply.

Draw Beam Absolute Y (Absolute X/Y)

21	27	35
[label]	DBAY	Y

Notes: In vector mode, beam movement is unblanked; in point mode, only the end point is unblanked. The notes for MBAX also apply.

Move Beam Stroke (Character Stroke Word)

21	27	32	35
[label]	MBS	[R]	X, Y

Notes: The x- and y-coordinates may be any valid absolute Assembler expressions, but X must be in

the range 0 to 6, and Y must be in the range 0 to 7. The x- and y-coordinates occupy a half-word. For consecutive orders, the coordinates for two orders are placed in one word. The revert function is executed if an R is placed in column 32. Character stroke orders must be executed out of line by means of a graphic branch following a set character mode order.

Draw Beam Stroke (Character Stroke Word)

21	27	32	33	35
[label]	DBS	[R]	[D]	X, Y

Notes: A D in column 33 indicates that less than normal (decreased) intensity is desired (recommended for character strokes less than 2 character units long; see the 2250 model 4 Component Description publication for details about character units). Programmed intensity provides a means of generating characters that have nearly uniform intensity for all the strokes of the character regardless of the stroke lengths. The programmer should experiment with this facility to achieve desired results. The notes for MBS also apply.

Control Stroke (Character Stroke Word)

21	27	32	35
[label]	CS	[R]	1, [data]
[label]	CS		2, [data]
[label]	CS	R	2, [data]
[label]	CS	[R]	4, [data]
[label]	CS	R	7, [data]

Notes: The first operand, which may be any valid absolute Assembler expression, has the following meanings:

- 1 = subscripting - the character grid is offset downward 3 vertical character units.
- 2 = no operation - the order performs no operation if R is not specified.
- 2 = null function - the order performs a null function if R is specified.
- 4 = superscripting - the character grid is offset upward 3 vertical character units.
- 7 = new line - the beam is positioned at the next line (R must be specified).

The "data" operand may be any data the programmer wants, but must not exceed 7 bits. Data exceeding the limit is truncated to the 7 low-order bits.

If revert (R) is not specified for the superscript control order, execution continues with the second word after the superscript control order. Placing a subscript control after the superscript control order gives a character stroke subroutine the capability of being executed in superscript, subscript, or normal mode if different entry points to the subroutine are defined.

Graphic Short Branch (Short Branch)

21	27	35
[label]	GSB	address

Notes: The address may be either symbolic or an absolute Assembler expression, but must have a value less than 8192. Use of the symbolic operand is restricted to referring to graphic orders that are within the same Assembler-language program.

Graphic Branch (Long Branch/Interrupt)

21	27	32	33	35
[label]	GB	[I]	[N]	address

Notes: An I in column 32 specifies an indirect branch. An N in column 33 specifies a two-word no-operation. The notes for GSB also apply, except that the address is not restricted to a value of less than 8192.

Graphic Branch Conditional (Long Branch/Interrupt)

21	27	32	33	35
[label]	GBC	[I]	[N]	address, condition

Notes: The condition for the branch may be one of the following:

- D = branch if light pen detect
- S = branch if light pen switch closed
- DS or SD = branch if light pen detect and switch closed

The notes for GB also apply.

Graphic Branch External (Long Branch/Interrupt)

21	27	33	35
[label]	GBE	[N]	name

Notes: Name is the name of an external order program (subroutine). An N in column 33 specifies a two-word no operation.

Graphic Branch Conditional External (Long Branch/Interrupt)

21	27	33	35
[label]	GBCE	[N]	name, condition

Notes: The conditions for the branch are the same as those described for GBC. The notes for GBE also apply.

Graphic Interrupt (Long Branch/Interrupt)

21	27	33	35
[label]	GI	[N]	[data]

Notes: An N in column 33 specifies a two-word no-operation. Data may be a symbolic address, number, or expression. The range of numerical data or an expression, when resolved, must be +32767 to -32768. The data word may be used for any purpose.

Graphic Interrupt Conditional (Long Branch/Interrupt)

21	27	33	35
[label]	GIC	[N]	[data], condition

Notes: The condition for the interrupt may be one of the following:

- D = interrupt if light pen detect
- S = interrupt if light pen switch closed
- DS or SD = interrupt if light pen detect and switch closed

The notes for GI also apply.

Table 2. Assembler Error Codes for 2250 Errors

Error Code	Cause	Action Taken by Assembler
W	x- or y-coordinate, or both, not within the specified range; or invalid operand	Operand set to zero.
X	Character other than R or I in column 32; or character other than D or N in column 33.	Field set to zero.
Y	Unnecessary operand specified; or unnecessary Tag or Format field entry.	Operand ignored; Tag or Format field ignored.
Z	Invalid condition in a conditional branch or or interrupt order.	Condition bits in first word set to zero.

INDEX

- ABS Statement 17
- Absolute Assembly 18
- Absolute Expressions 10
- Arithmetic Instructions 14
- Assemble Absolute Statement 17
- Assembler Features 1
- Assembler Instructions 17, 26
 - ABS - Assemble Absolute 17
 - BES - Block Ended by Symbol 22
 - BSS - Block Started by Symbol 22
 - CALL - Call Subroutine 24
 - DC - Define Constant 19
 - DEC - Decimal Data 19
 - EBC - Extended BCD Information 21
 - END - End of Source Program 19
 - ENT - Define Subroutine Entry Point 23
 - EQU - Equate Symbol 22
 - ORG - Define Origin 18
 - XFLC - Extended Floating Constant 21
- Assembler Program 1
- Asterisk as an Element 9
- Asterisk as an Operator 10
- Asterisk in Column 21 3
- Automatic Storage Assignment 2

- BES Statement 22
- Binary-Point Identifier 20
- Blank Format 4
- Block Ended by Symbol Statement 22
- Block Started by Symbol Statement 22
- Branch Instructions 14, 16
- BSC Equivalents (Monitor) 15
- BSS Statement 22

- CALL Statement 24
- Character Code Chart 34
- Character Codes 33
- Character Set 7, 34
- Character Values 9
- Coding Form 4
- Comments Field 6
- Condition Testing Instructions 13
- Console Printer Hex Code 33

- Data Definition Statements 19, 28
- Data Representation 2
- DC Statement 19
- DEC Statement 19
- Decimal Data Items 19
- Disk Data Organization Statements 26
- Decimal Data Statement 19
- Decimal/Hexadecimal Conversion Chart 38
- Decimal Integers 20
- Define Constant Statement 19
- Define Disk File (Monitor) 27
- Define Interrupt Level Subroutine Statement 24
- Define Interrupt Service Entry Point Statement 23
- Define Message (Monitor) 28
- Define Name Statement (Monitor) 29
- Define Origin Statement 18
- Define Sector Address Statement (Monitor) 26
- Define Subroutine Entry Point Statement 23
- Displacement 4, 5
- DMES Statement (Monitor) 28
- DN Statement (Monitor) 29
- DSA Statement (Monitor) 26
- Dump and Continue Execution (Monitor) 31
- Dump and Terminate Execution (Monitor) 30
- DUMP Statement (Monitor) 30

- EBC Statement 21
- EBCDIC Code 33
- Effective Address 4
- EJCT Statement (Monitor) 32
- Elements 9
- End of Source Program Statement 19
- END Statement 19
- ENT Statement 23
- EPR-Extended Precision 18
- EQU Statement 22
- Equate Symbol Statement 22
- Error Checking 2
- EXIT Statement (Monitor) 30
- Exponent 20
- Expressions 9
- Extended Binary Coded Information Statement 21
- Extended Binary Coded Decimal Interchange Code 33
- Extended Real Constant Statement 21

- Features of the Assembler 1
- Fields
 - Comments 6
 - Format 4
 - Identification-Sequence 7
 - Index Register 6
 - Label 3
 - Operand 6
 - Operation 3
 - Remarks 6
 - Tag 6
- FILE Statement (Monitor) 27
- Fixed Point Numbers 20
- Format Field 4
- Format of Assembler Statements 3
- Graphic Orders 39.1

- HDNG Statement (Monitor) 31
- Heading Statement (Monitor) 31
- Hexadecimal/Decimal Conversion Chart 38

Hexadecimal Notation 33
Hexadecimal Values 9

I Format 5
IAR 4
IBM Card Code 33
Identification Field 7
ILS Statement 24
Index Registers, Specifying 6
Index Registers 6, 25
Index Register Field 6
Indirect Addressing 6
Input/Output Instructions 14
Instruction Address Register 4
ISS Number 23
ISS Statement 23

L Format 5
Label Field 3
LIBF - Call TV Reference Subroutine 25
LIBF Subroutine Transfer Vector 25
LIBR - Transfer Vector Subroutine 18
LINK Statement (Monitor) 30
Linking Statements 23, 30
LIST Statement (Monitor) 31
List Control Statements (Monitor) 31
List Segments of Program (Monitor) 31
Location Assignment Counter 7
Location Assignment Counter Overflow 8
Load Instructions 14
Load Link Program (Monitor) 30

Machine-Instruction Statements 13
Machine-Instruction Mnemonics 13
Mantissa 20
Miscellaneous Instructions 14
Mnemonics 13, 15
Mnemonic Concept 3
Mnemonic Operation Codes 1
Modify Memory (Monitor Mnemonic) 15
Monitor Assembler Statements
 DMES - Define Message 28
 DN - Define Name 29
 DSA - Define Sector Address 26
 DUMP - Dump and Terminate Execution 30
 EJCT - Start New Page 32
 EXIT - Return to Supervisor 30
 FILE - Define Disk File 27
 HDNG - Heading 31
 LINK - Load Link Program 30
 LIST - List Segments of Program 31
 PDMP - Dump and Continue Execution 31
 SPAC - Space Listing 32

Name Code 29

Operand Field 6
Operators 10
Operation Field 3

ORG Statement 18
Overflow, Location Assignment Counter 7

Paper Tape Transmission Code 33
PDMP Statement (Monitor) 31
Program Control Statements 17
Program-Linking Statements 23, 30
Program Listings 2
Programming Considerations for DSA Statement 27
PTTC/8 33

Real Numbers 20
Relative Addressing 8
Relocatable Assembly 17
Relocatable Expressions 10
Relocatable Programs 17
Remarks Field 6
Renaming Symbols 2
Return to Supervisor (Monitor) 30

Self-Defining Values
 Decimal 9
 Hexadecimal 9
 Character 9
Sequence Field 7
Shift Instructions 14, 16
Slash (/), Use of 9
Source Program 3
SPAC Statement (Monitor) 32
Space Listing (Monitor) 32
SPR-Standard Precision 18
Start New Page (Monitor) 32
Statement Field 3
Statement Writing 7
Storage Allocation Statements 22
Store Instructions 14
Subroutine Transfer Vector 25
Subroutines 1
Symbol Definition Statement 22
Symbol Table 7
Symbolic Language 1, 3
Symbolic Reference to Storage Addresses 2
Symbols 7
Symbols, Restrictions 7

Tag Field 6
Terms 10
Transfer Vector (LIBF) 25
Types of Expressions 10

Writing
 Statements 7
 Subroutines 23

X Format 5
XFLC Statement 21

1132 Printer EBCDIC Subset Hex Code 33
1403 Printer Hex Code 33
2250 Display Unit, Model 4 39.1

File Number 1130-21
Re: Form No. C26-5927-4
This Newsletter No. N33-8015
Date June 5, 1968
Previous Newsletter Nos. None

IBM 1130 Assembler Language

This Technical Newsletter provides replacement pages for IBM 1130 Assembler Language, Form No. C26-5927-4. Pages to be inserted and/or removed are listed below.

<u>Remove Pages</u>	<u>Insert Pages</u>
iii, iv	iii, iv
17, 18	17, 18
23-26	23, 24
	25, 25.1
	25.2, 26
29-32	29-32
39-42	39
	39.1, 39.2
	39.3, 39.4
	39.5, 40
	41, 42

A change to the text or a small change to an illustration is indicated by a vertical line to the left of the change; a changed or added illustration is noted by the symbol * to the left of the caption. Completely new or substantially changed pages are indicated by the symbol * to the left of the page number.

Summary of Amendments

This TNL supplies the following amendments to the manual:

- + New information on the 2250 Display Unit, Model 4, ISS (DSPYN).
- + Revision of the discussion on CALL and LIBF statements.
- + Various corrections to previous information in the manual.

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

READER'S COMMENT FORM

IBM 1130 Assembler Language

Form C26-5927-4

- Your comments, accompanied by answers to the following questions, help us produce better publications for your use. If your answer to a question is "No" or requires qualification, please explain in the space provided below. Comments and suggestions become the property of IBM.

- | | Yes | No |
|--|--------------------------|--------------------------|
| • Does this publication meet your needs? | <input type="checkbox"/> | <input type="checkbox"/> |
| • Did you find the material: | | |
| Easy to read and understand? | <input type="checkbox"/> | <input type="checkbox"/> |
| Organized for convenient use? | <input type="checkbox"/> | <input type="checkbox"/> |
| Complete? | <input type="checkbox"/> | <input type="checkbox"/> |
| Well illustrated? | <input type="checkbox"/> | <input type="checkbox"/> |
| Written for your technical level? | <input type="checkbox"/> | <input type="checkbox"/> |
- What is your occupation? _____
 - How do you use this publication?

As an introduction to the subject?	<input type="checkbox"/>	As an instructor in a class?	<input type="checkbox"/>
For advanced knowledge of the subject?	<input type="checkbox"/>	As a student in a class?	<input type="checkbox"/>
For information about operating procedures?	<input type="checkbox"/>	As a reference manual?	<input type="checkbox"/>

Other _____

- Please give specific page and line references with your comments when appropriate.

COMMENTS

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

YOUR COMMENTS PLEASE . . .

This SRL bulletin is one of a series which serves as reference sources for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

Fold

Fold

FIRST CLASS
PERMIT NO. 1359
WHITE PLAINS, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES



POSTAGE WILL BE PAID BY . . .

IBM Corporation
112 East Post Road
White Plains, N. Y. 10601

Attention: Department 813

Fold

Fold



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

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)