File No. 1130-21
Form C26-5927-2

IBM 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 F22-5517), is suggested as preliminary reading.

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.

The reader should also be familiar with the following: IBM 1130 Functional Characteristics (Form A26-5918) and IBM 1130 Computing System, Input/Output Systems (Form A26-5890).

This publication is oriented towards the 1130 Card/Paper Tape Programming System which includes the basic assembler programs. The assembler language is also valid for the 1130 Monitor Programming system and a section on special Monitor statements is included. The Monitor System is described in the publication, IBM 1130 Monitor System Specifications (Form C26-5940).

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).

MACHINE REQUIREMENTS

The minimum machine configuration for assembling programs with the IBM 1130 Card/Paper Tape Assembler 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.

Second Edition

This edition is a reprint of C26-5927-1 and incorporates changes released in Technical Newsletter N26-0554, dated March 10, 1966.

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 Corporation, Programming Publications, Department 452, San Jose, California 95114.

CONTENTS

INTRODUCTION

The IBM 1130 Assembler language, while similar in structure to machine 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 of the flexibility and versatility of machine language, plus facilities that greatly reduce machine language programming effort.

COMPONENTS OF THE IBM 1130 ASSEMBLER

The IBM 1130 Assembler has two parts: the symbolic language used in writing a program and the assembler program that converts the symbolic language into machine language. An additional component is a library of relocatable input/output, arithmetic, and functional subroutines.

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, address, and data, which symbolically describe the problem to be solved by the computer.

The use of symbolic language:

1. Makes a program independent of actual machine locations, thus allowing programs and routines to be relocated and combined as desired.
2. Allows routines within a program that can be written independently and that cause no loss of efficiency in the final program.
3. 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 (processor), supplied to the user by IBM in either paper tape or punched card form, converts (assembles) a symbolic-language source program into a machine-language program. The conversion is one for one — that is, the assembler produces one machine-language instruction for each symbolic-language instruction.

The IBM 1130 Assembler is a two-pass program. The processor 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 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 publication IBM 1130 Subroutine Library (Form C26-5929).

FEATURES OF THE ASSEMBLER

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

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.

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.

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 fixed-point or floating-point arithmetic operations.

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.

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.

Assembler Instructions: A set of special instructions to the assembler is included in the language. Some of the features described in this section are implemented by these instructions.

Program Listings: For every assembly, the user can obtain a program listing. This listing can be produced either off-line on an IBM 407 or similar device, or on-line by the use of an IBM-supplied utility routine.

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:

Description	Mnemonic
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 punched into cards or paper tape. 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. The position numbers shown on the form correspond to the card columns. If paper tape is used, the first character of each tape record corresponds to column 21. 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.

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. A label that is not further referred to is not an error.

A label can consist of up to five alphameric 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.

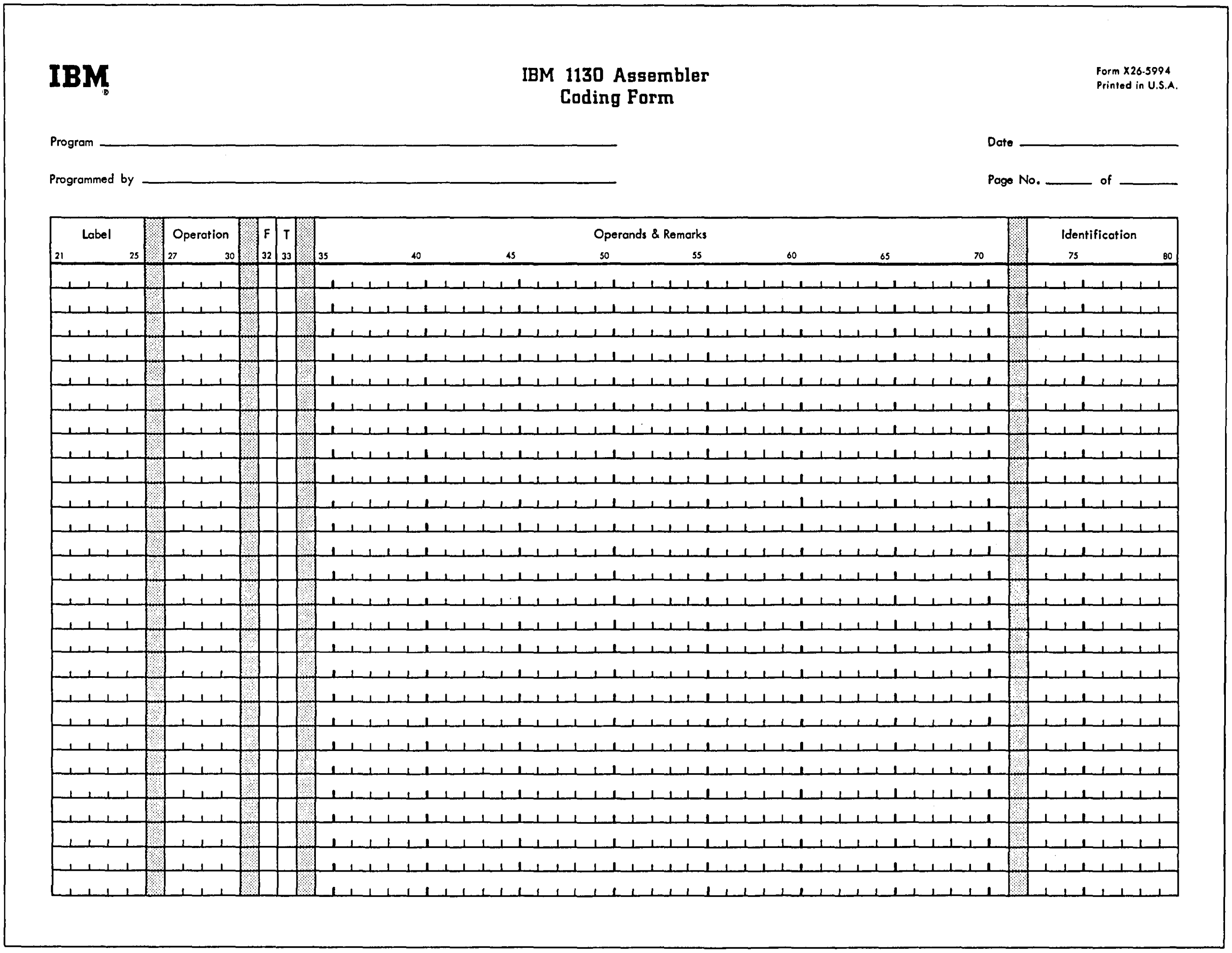
Label 21-25		Operation 27-30		F 32	T 33		35 40 45
ALPHA		STO					AN EXPRESSION

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.

The best labels to select are those that are mnemonically descriptive of the area or instruction to which they are assigned. Labels that have an obvious meaning provide easily remembered references for the original programmer and assist others who may assume responsibility for the program.

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 card.


IBM

IBM 1130 Assembler Coding Form

Form X26-5994
Printed in U.S.A.

Program
Date
Programmed by
Page No. of

Label 21–25	Operation 27–30	F 32	T 33	Operands & Remarks 35 40 45 50 55 60 65 70	Identification 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 standard 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 one-word 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 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 one-word Store Index (STX) instructions is always obtained by adding the displacement to the IAR.

The displacement of 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 one-word 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.

A short instruction assembled with a blank format field must not be relocated at execution time by an amount differing from the relocation increment of the data referred to by that instruction.

If a short instruction of the kind relatively addressed to the IAR is to be moved, under program control, to a location different from the location at which the instruction was assembled, the automatic subtraction of the address counter from the displacement operand value should be suppressed by use of the X format.

This requirement is not in conflict with the load-time relocation process because the process shifts the whole program, including instructions and reference data, to a memory 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 relatively addressed to 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 machine instruction counter will be 1001_{10} when instruction A is executed. Therefore, the value computed by the assembler for the displacement will be 49_{10}.

Label 21–25	Operation 27–30	F 32	T 33	35 40 45
A	LD			B
	.			
	.			
	.			
	.			
	.			
	.			
B	DC			CONST

In the case of an instruction which is not relatively addressed to 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 an effective "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. 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 that 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 two-word 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 because this bit is also the indirect flag bit.

Label (21–25)	Operation (27–30)	F (32)	T (33)	Operands & Remarks (35–70)
	•			
	•			
	•			
	•			
SWTCH	SLT			0 EFFECTIVE NO OPERATION
	•			
	•			
	•			
	LD			BRCON CHANGE PROGRAM FLOW AT SWTCH
	STO			SWTCH
	•			
	•			
	•			
BRCON	MDX	X		GO-SWTCH-1

Assume SWTCH = location 1000_{10}
GO = location 1050_{10}
BRCON = location 1100_{10}

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 LDX, MDX, or STX instruction, the IAR is used. The example below shows an add instruction in its long form addressing the core location whose address is zero plus the contents of Index Register 2.

Label (21–25)	Operation (27–30)	F (32)	T (33)	(35–40)
SUM	A	L	2	

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

The entire statement field can be used for comments by placing an asterisk in column 21; the entire

SHORT	STO			ACCU1
LONG	MDX	L		ACCU1,100 TWO-OPERAND STATEMENT
LONG	STO	L		ACCU1 ONE-OPERAND LONG STATEMENT

Figure 3. One- and Two-Operand Statements

statement is then treated as 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 can 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 which also contains 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 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$#@

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 which 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:

1. 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 is noted as an error in the program listing. In addition, any reference to such a symbol is noted as an error.
2. The number of symbols that can be defined in a program is restricted by the amount of core storage available to the assembler (see 1130 Card/Paper Tape Programming System Operator's Guide (C26-3629).

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

Label 21–25		Operation 27–30		F 32	T 33		Operands & Remarks 35–70
*THE	A	STER	I	S	K		IN COL 21 MAKES THIS A COMMENTS LINE
*AN A	S	TERI	S	K		I	S REQUIRED FOR EACH LINE OF COMMENTS

Figure 4. Example of Comments Statement

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).

For example, in the sequence

Label 21–25	Operation 27–30	F 32	T 33	35–45
START	A			BETA
	S			STORE
	STO	L		ADDR1
ALIST	A	L		LIST
	D			LOC2

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

Label 21–25	Operation 27–30	F 32	T 33	35–45
	BSC	L		START+1
	BSC	L		ALIST-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 21–25	Operation 27–30	F 32	T 33	35–45
BETA	BSS			50

reserves a block of 50 words, in which BETA is 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 (n+1)th 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 an actual 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 (highest value)
/AB12
/F2
/379B

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 21–25	Operation 27–30	F 32	T 33	35 40 45
	LD			A,B,C
SUM	A			DEF
	S			DATA
	BSC	L		SUM,+

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

Label 21–25	Operation 27–30	F 32	T 33	35 40 45
	BSC	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 separated by two asterisks, etc. A term must begin with an element and end with an element. It 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 which 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 program 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 by the loading routine.

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):

1. Discard any term that contains only absolute elements.
2. Examine each term of the expression. If any term contains more than one relocatable element, the expression will yield a relocation error.
3. 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.
4. 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.

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 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 off	15
C	Carry	Skip or do not branch if Carry indicator off	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:

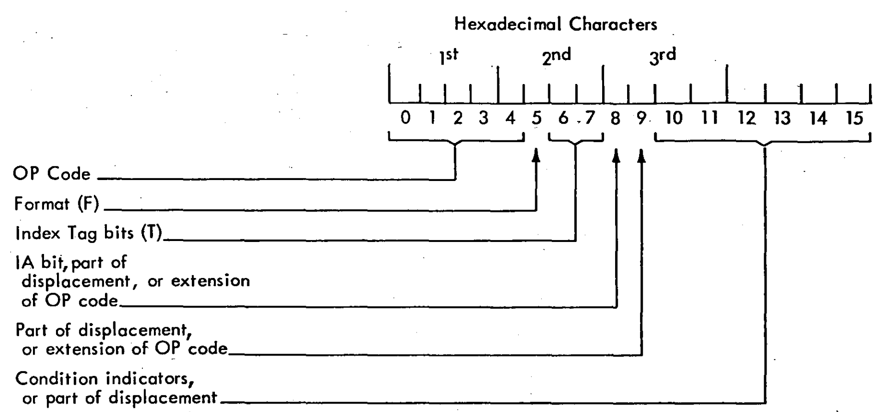
BSC	+	Skip on plus condition
BSC	+-	Skip on non-zero (plus or minus)
BSC	Z-	Skip on non-plus (zero or minus)
BSC	C	Skip if Carry indicator off
BSC L	EXIT, +	Branch to EXIT if not plus (zero or minus)
BSC L	EXIT, + -	Branch to EXIT if zero (not plus or minus)
BSC L	EXIT	Unconditional Branch to EXIT
BSC L1	0, Z+	Branch to the contents of XR1 on minus (not zero or plus)
BSI L	SUBR, O	Branch and Store instruction counter to SUBR if Overflow is on

Table 1. Machine Instruction Mnemonics

Mnemonic	OP Code (Hexadecimal Representation)[1]	Instruction
Load and Store		
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
Arithmetic		
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
Branch		
BSI	400	Branch and Store Instruction Counter
BSC	480	Branch or Skip Conditionally
BOSC[2]	484	Branch Out or Skip Conditionally
MDX	700	Modify Index and Skip
Shift		
SLA*	100	Shift Left Accumulator
SLT*	108	Shift Left Accumulator and Q Reg.
SLC*	10C	Shift Left and Count Accumulator and Q Reg.
SLCA*	104	Shift Left and Count Accumulator
SRA*	180	Shift Right Accumulator
SRT*	188	Shift Right Accumulator and Q Reg.
RTE*	18C	Rotate Right
Input/Output		
XIO	080	Execute I/O
Miscellaneous[3]		
NOP*	100	No Operation
WAIT*	300	Wait

*Valid in short format only.

1. The hexadecimal representation of the machine operation code is derived from the instruction format in the manner shown below. Bits 5, 6, 7, 10, and 11 are assumed to be zeros because they do not enter into the makeup of any operation codes.
2. Same as BSC with Bit 9 set to one.
3. An operand should not be specified.



Just as machine instructions are requests to the computer to perform a sequence of operations during program execution time, 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 at assembly time; 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 instructions permitted by the IBM 1130 Assembler.

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 21 25		Operation 27 30		F 32	T 33		35 40 45
		ABS					

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 at assembly time 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 and certain main programs should 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, as indicated on the end-of-program card. 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 that are inserted into the binary object program cards when the program is compressed following assembly.

Absolute Assembly

The programmer uses the ORG assembler statement in his source program to specify the locations into which the binary object program resulting from an absolute assembly is loaded. The loader makes no adjustment to any field during loading. Subroutines are loaded into successively higher even-core locations following the end of the main program.

Remember that 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 and 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. A typical Define Origin statement is shown below.

Label 21–25	Operation 27–30	F 32	T 33	35 40 45
	ORG			3000

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 21–25	Operation 27–30	F 32	T 33	Ope[rand] 35 40 45 50
	ORG			XYZ
START	ORG			XYZ+50
START	ORG			*+50 LOC CTR+50

If the label XYZ has been previously defined as 1000_{10} 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 cause execution of a main program to begin immediately after loading. 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 (21–25)	Operation (27–30)	F (32)	T (33)	Operands (35–50)
	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 (21–25)	Operation (27–30)	F (32)	T (33)	Operands (35–45)
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 (21–25)	Operation (27–30)	F (32)	T (33)	Operands (35–50)
HEX	DC			/FFFF HEX CONST
DEC	DC			-385 DEC INTEGER
ALPHA	DC			.B CHAR CONST
ADDRS	DC			ALPHA+5 ADDR CON

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 (21–25)	Operation (27–30)	F (32)	T (33)	Operands (35–45)
LABEL	DEC			DEC 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 (21–25)	Operation (27–30)	F (32)	T (33)	Operands (35–45)
DATA	DEC			+19

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, floating-point numbers, and fixed-point numbers. A floating-point 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

Decimal Integer	Stored As
50	00000032_{16}
1535	$000005FF_{16}$
-3729	$FFFFF16F_{16}$ (2's complement)

Floating-Point Numbers. A floating-point 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 floating-point number is converted to a normalized, floating-point, 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 floating-point numbers are equivalent and will be converted to the same floating-point binary number.

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

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

Word 1	Word 2
4800	0083

The DEC assembler instruction stores floating-point numbers in the standard precision floating-point format described in the manual, IBM 1130 Subroutine Library (Form C26-5929).

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 floating-point numbers.

- Exponent — The exponent is the same as described for floating-point 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 floating-point 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	Word 2
5A00	0000

XFLC — Extended Floating Constant

The XFLC assembler instruction is used to introduce into a program an extended precision floating-point constant, expressed in three consecutive data words. When assembled, this instruction produces a format identical to the extended range floating-point format described in the manual, IBM 1130 Subroutine Library (Form C26-5929).

The format of the XFLC instruction is shown below:

Label (21–25)	Operation (27–30)	F (32)	T (33)	Operand (35–50)
LABEL	XFLC			FLOAT POINT NUM

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:

Label (21–25)	Operation (27–30)	F (32)	T (33)	Operand (35–45)
	XFLC			0.53125
FLOPT	XFLC			-0.53125
	XFLC			5.12E2

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

	Word 1	Word 2	Word 3
1.	0080	4400	0000
2.	0080	BC00	0000
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:

Label (21–25)	Operation (27–30)	F (32)	T (33)	Operand (35–45)
LAB1	EBC			.ALPHA DATA.

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

Label (21–25)	Operation (27–30)	F (32)	T (33)	Operand (35–45)
CONST	EBC			.EROR.
ALPHA	EBC			.CONSTANT.

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.

	Word 1	Word 2	Word 3
	C5D9	D9D6	D940
CONST →	↑		

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:

Word 1	Word 2	Word 3	Word 4
C3D6	D5E2	E3C1	D5E3

NOTE: A period may not appear in the comments 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 21–25	Operation 27–30	F 32	T 33	35 40 45
LABEL	BSS			ABSOLUTE EXPR.

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 skipped, 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 21–25	Operation 27–30	F 32	T 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 21–25	Operation 27–30	F 32	T 33	35 40 45
SYMBL	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 21–25	Operation 27–30	F 32	T 33	35 40 45
NAME	EQU			26
LOOP	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.

PROGRAM-LINKING STATEMENTS

Program-linking statements are used to establish communication between a main program and its subroutines. Five statements are provided: three for entry point definition and two for subroutine calls.

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 ten entry points may be defined for each subroutine (this would require ten ENT statements). The format of the ENT statement is shown below.

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

NAME is a symbol which 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.

ISS — Define Interrupt Service Entry Point

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

Label (21–25)		Operation (27–30)		F (32)	T (33)		(35–45)
		ISS		N	N		NAME L

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.

Label	ISS Header Word 30
blank	blank
1130	1
1800	2

NAME is as described for the ENT statement and NN (the ISS number) is a number from 01 to 20 used by the Relocating or core image loader to establish the linkage from the appropriate point in the corresponding ILS. The numbers and associated devices used in the routines provided by IBM are listed below:

Number*	Device or Function
01	1442 Card Read Punch
02	Input Keyboard; Console Printer
03	1134 Paper Tape Reader; 1055 Paper Tape Punch
04	2310 Disk
06	1132 Printer
07	1627 Plotter

* Numbers 08 through 20 are assignable by the user.

L is a one-digit number used 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 and they must be left-justified in these columns (first in 45, second in 50 etc.).

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

Procedures for writing interrupt-service subroutines are provided in the publication IBM 1130 Subroutine Library (C26-5929).

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 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:

Label (21–25)	Operation (27–30)	F (32)	T (33)	35
	ILS	N	N	

NN is the interrupt level number (00-05) associated with the interrupt level subroutine and is used by the relocating loader or core image converter. The devices associated with each interrupt level are shown below:

Interrupt Level	Device(s)
00	1442 Card Read Punch
01	1132 Printer
02	Disk Storage
03	1627 Plotter
04	Console Keyboard, Console Printer, 1442 Card Read Punch, 1134 Paper Tape Reader, 1055 Paper Tape Punch
05	PROGRAM STOP Key or Interrupt Run Mode.

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

Procedures for writing interrupt level subroutines are provided in the publication, IBM 1130 Subroutine Library (Form C26-5929).

CALL — Call Direct Reference Subroutine

A CALL statement is used to call some of the subroutines in the IBM Subroutine Library or any user-written subroutine written for the CALL statement. At execution time, this type of call takes the form of a long (two-word) BSI, to the entry point named in the CALL and the corresponding ENT or ISS statement.

When BSI is executed, the location of the first word following it is placed in the entry point location, and control is transferred to the first word following the entry point. The format of the CALL statement is:

Label (21–25)	Operation (27–30)	F (32)	T (33)	35–40
LABEL	CALL			NAME

If used, the label is assigned to the current value of the Location Assignment Counter, which is the same as the leftmost word of the generated BSI instruction. The name of the called subroutine is assembled into the object deck, together with a unique code identifying the CALL. This code is recognized by the loader, which generates the BSI to the called subroutine.

LIBF — Call TV (Transfer Vector) Reference Subroutine

An LIBF statement is used to call any of the subroutines in the Subroutine Library (or any user-written subroutine) written to utilize the transfer vector (see the following section, Subroutine Transfer Vector). The format of the LIBF statement is:

Label (21–25)	Operation (27–30)	F (32)	T (33)	35–40
LABEL	LIBF			NAME

If used, the label is assigned to the current value of the Location Assignment Counter when the LIBF statement is encountered. The name of the called subroutine is assembled into the object deck, together with a unique code identifying the call as an LIBF call. This code is recognized by the loader, which generates a TV linkage to the called subroutine. During execution, the TV subroutine calls Index Register 3. Therefore, if Index Register 3 is used by any other instruction in the user's program, it must be saved and restored before it is needed by any TV subroutine calls. An LIBF Call is sometimes referred to as a One Word Call.

LIBF Subroutine Transfer Vector

To understand fully the use of the LIBF statement, the user should be familiar with the makeup of the subroutine transfer vector. A transfer vector allows main programs 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 16-bit words for each subroutine entry point referred to by an LIBF statement. The contents of the three words vary as the subroutine goes through the three phases of being called, loaded, and executed. The following paragraphs describe these three phases, and illustrate the contents of the transfer vector for each phase.

Recognizing the Subroutine Call. All subroutines that utilize the transfer vector are called via LIBF statements. These statements take the following general form:

```
LIBF        NAME
DC          Parameter
DC          Parameter
etc.
```

When an LIBF call is recognized during the loading of an object program, the loader begins to build the transfer vector by setting aside a three-word record and placing in it the name of the called subroutine.

Name of Subroutine	Zeros

Subsequent LIBF statements produce additional records in the transfer vector, each containing a unique subroutine name. Calls to a subroutine previously listed in the transfer vector do not produce a new record. Each time an LIBF call is recognized, the loader replaces the associated LIBF NAME statement in the user's program with a one-word, indexed BSI instruction pointing to the first word of the associated transfer vector record. This instruction, generated by the loader, uses Index Register 3 and a computed displacement to refer to the proper transfer vector record.

Original Statement

Label (21–25)	Operation (27–30)	F (32)	T (33)	35 40 45
	LIBF			NAME

Modified Statement

Label (21–25)	Operation (27–30)	F (32)	T (33)	35 40 45
	BSI		3	DISPLACEMENT

When this BSI instruction is encountered during execution of the main program, it causes a branch to the associated transfer vector record and from there to the entry address of the subroutine (see the following section, Loading the Subroutine). A BSI statement is generated for each LIBF statement encountered.

NOTE: Index Register 3 is reserved for LIBF subroutine calls. Therefore, if any instructions are to be tagged for register 3, register 3 should be restored prior to any LIBF subroutine call.

Loading the Subroutine. After the main program has been read and all the called subroutine names are in the transfer vector, the loader loads the subroutines. As the header card of each subroutine in the library deck is read, the subroutine name is compared with the list of names previously placed in the transfer vector. When a match is found, the subroutine is loaded in an available core location, determined by the loader, and the subroutine entry address is placed in the third word of the associated transfer vector record. In addition, the address of the transfer vector record is inserted in the subroutine at entry point +2.

Name of Subroutine	Entry address of Subroutine

After the loader has read all of the subroutines in the subroutine deck and loaded those that were called, it modifies the transfer vector record of each subroutine that was loaded. This modification consists of deleting the subroutine names and placing, in the second word of each record, the first half of a long BSC instruction.

Undefined	BSC L	Entry address of Subroutine

Note that word 3 of the record is not affected and thus becomes the effective address of the long BSC instruction. This instruction is used to branch to a particular subroutine when that subroutine is called during execution of the main program.

NOTE: The preceding description of the subroutine loading process applies only to the card and paper tape systems and not to the Monitor system.

Executing the Subroutine. As the main program is executed, the subroutine calls are encountered one by one. Remember that each call is now in the form of a BSI instruction. The BSI instruction loads the contents of the IAR into the first word of the transfer vector record associated with the subroutine being called and then branches to the second word of the transfer vector record. As a result, the first word of the transfer vector record contains the address LIBF +1.

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 record is executed, transferring control to the subroutine. In order to collect parameters and to return from a subroutine to the calling program, the programmer must know the address of the transfer vector record associated with the subroutine. This address was placed into the subroutine location, entry point +2, by the loader. Having this information, the programmer can find the address, LIBF +1. If parameters follow the statement, the address LIBF +1 (stored in the first word of the transfer vector record) is the address of the first parameter, otherwise it is the return address. A suggested method of computing the return address for a subroutine with one parameter is illustrated in Figure 5.

Size and Location of the Transfer Vector

The transfer vector consists of a maximum of 256 core locations and is located in the uppermost part of core storage (highest-numbered locations). If a COMMON area is used for FORTRAN programs, the transfer vector is located below the COMMON area.

The location of the transfer vector in the Monitor system is different. The size of the transfer vector (256 locations) is sufficient to contain 85 three-word records. All but the first two of these records are available for use as subroutine communication areas. These two reserved records are used by certain arithmetic subroutines as a floating-point pseudo-accumulator, and as program error indicators.

Restrictions on the Use of the TV Reference Subroutines

1. No more than 83 unique entry point names referred to by LIBF statements can be loaded at one time.
2. In the card and paper tape systems, a subroutine called by an LIBF call must not contain any CALL (direct reference call) statements.

Label (21–25)	Operation (27–30)	F (32)	T (33)	Operands & Remarks (35–70)
	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 REG 2
	LDX	I	2	0 THIS LOCATION WILL BE BLANK
				WHEN SUBR IS WRITTEN, BUT WILL
				BE FILLED WITH TV ENTRY ADDRESS
				BY THE LOADER
	LD	I	2	0 LOAD ARGUMENT INTO A REG FOR USE
	.			
	.			
	.			MAIN BODY OF SUBROUTINE
	.			
	.			
	MDX		2	+1 ADD 1 TO IX2 TO OBTAIN RETURN ADDR
	STX		2	*+3 STORE RETURN ADDRESS IN SAVE+3
SAVE	LDX	L	2	0 RESTORE INDEX REG 2
	BSC	I		0 RETURN TO MAIN PROGRAM
	END			

Figure 5. Example of a User-Written TV Reference Subroutine

The following assembler statements apply only to the IBM 1130 Monitor System. These statements are not valid for use with the card or paper tape assembler.

LINK — Load Link Program

In the assembler language, the LINK statement is used to load and execute another complete program. 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 the Location Equivalence Table at execution time. The LINK statement causes four words to be generated in the object program. The first two words contain a two-word BSI instruction, which branches to a specified location within the Skeleton Supervisor program. The next two words contain the program name, left-justified, with blanks inserted in unused rightmost positions. The Supervisor Program uses the program name and begins the process required to load the new program.

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 first word generated by an EXIT statement.

The operand field is ignored and can, therefore, be used for comments. The EXIT statement causes a one-word branch instruction to be generated in the object program. The instruction branches to a fixed location in the Skeleton Supervisor program. At object time the branch is executed and control is returned to the Supervisor. The EXIT statement should be the last logical statement in a program.

DSA — Define Sector Address

The DSA statement allows the programmer to refer symbolically to a disk-stored data file or core-image program 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 21–25	Operation 27–30	F 32	T 33	35 40 45
LABEL	DSA			SYMBOLIC NAME

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 core-image program that is on disk at execution time.

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 publication IBM 1130 Subroutine Library, Form C26-5929.

Label 21–25	Operation 27–30	F 32	T 33	35 40 45
	•			
	•			
	LIBF			DISK1
	DC			/1000
	DC			IØAR
	DC			ERROR
	•			
IØAR	DSA			DATA
	BSS			319
	•			
	•			

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

Word 1 — Length (in words)
Word 2 — Sector Address
Word 3 — Not used

For a program they will contain:

Word 1 — Length (in words)
Word 2 — Sector Address
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 operation of the Disk I/O routine.)

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

Label 21–25	Operation 27–30	F 32	T 33	35 40 45
	•			
	•			
	LD			IØAR+2
	STO			BRNCH+1
	LIBF			DISK1
	DC			/1000
	DC			IØAR
	DC			ERROR
CALL	LIBF			DISK1
	DC			/0000
	DC			IOAR
	MDX			CALL
BRNCH	BSC	L		0
	•			
	•			
	•			
IØAR	DSA			PRGRM
	BSS			319
	•			
	•			

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 21–25	Operation 27–30	F 32	T 33	35 40 45
	LD			ADR2
	STO			IOAR
	LD			ADR2+1
	STO			IOAR+1
	LD			ADR2+2
	STO			BRNCH+1
	MDX			BRNCH-8
ADR2	DSA			PGRM2
	•			
	•			

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.

1. The called programs must be in core-image format.
2. If the calling program is converted to core image 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.

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 (card columns 35-71, or the corresponding frames of an HDNG statement in paper tape format).

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

When the 1132 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 typewriter is the principal printer, the heading line is preceded and followed by a single line feed, and otherwise functions as a comments statement.

APPENDIX A. CHARACTER CODE SUMMARY

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:

Binary	Hexadecimal	Decimal
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 three 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.

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

Ref No.	EBCDIC Binary 0123	EBCDIC Binary 4567	EBCDIC Hex	IBM Card Code Rows 12	11	0	9	8	7-1	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	Notes
0	0000	0000	00	12		0	9	8	1	B030	NUL					
1		0001	01	12			9		1	9010						
2		0010	02	12			9		2	8810						
3		0011	03	12			9		3	8410						
4		0100	04	12			9		4	8210	PF	Punch Off				
5*		0101	05	12			9		5	8110	HT	Horiz.Tab		6D (5)	41	(1)
6*		0110	06	12			9		6	8090	LC	Lower Case		6E (5)		
7*		0111	07	12			9		7	8050	DEL	Delete		7F (5)		
8		1000	08	12			9	8		8030						
9		1001	09	12			9	8	1	9030						
10		1010	0A	12			9	8	2	8830						
11		1011	0B	12			9	8	3	8430						
12		1100	0C	12			9	8	4	8230						
13		1101	0D	12			9	8	5	8130						
14		1110	0E	12			9	8	6	80B0						
15		1111	0F	12			9	8	7	8070						
16	0001	0000	10	12	11		9	8	1	D030						
17		0001	11		11		9		1	5010						
18		0010	12		11		9		2	4810						
19		0011	13		11		9		3	4410						
20*		0100	14		11		9		4	4210	RES	Restore		4C (5)	05	(2)
21*		0101	15		11		9		5	4110	NL	New Line		DD (5)	81	(3)
22*		0110	16		11		9		6	4090	BS	Backspace		5E (5)	11	
23		0111	17		11		9		7	4050	IDL	Idle				
24		1000	18		11		9	8		4030						
25		1001	19		11		9	8	1	5030						
26		1010	1A		11		9	8	2	4830						
27		1011	1B		11		9	8	3	4430						
28		1100	1C		11		9	8	4	4230						
29		1101	1D		11		9	8	5	4130						
30		1110	1E		11		9	8	6	40B0						
31		1111	1F		11		9	8	7	4070						
32	0010	0000	20		11	0	9	8	1	7030						
33		0001	21			0	9		1	3010						
34		0010	22			0	9		2	2810						
35		0011	23			0	9		3	2410						
36		0100	24			0	9		4	2210	BYP	Bypass				
37*		0101	25			0	9		5	2110	LF	Line Feed		3D (5)	03	
38*		0110	26			0	9		6	2090	EOB	End of Block		3E (5)		
39		0111	27			0	9		7	2050	PRE	Prefix				
40		1000	28			0	9	8		2030						
41		1001	29			0	9	8	1	3030						
42		1010	2A			0	9	8	2	2830						
43		1011	2B			0	9	8	3	2430						
44		1100	2C			0	9	8	4	2230						
45		1101	2D			0	9	8	5	2130						
46		1110	2E			0	9	8	6	20B0						
47		1111	2F			0	9	8	7	2070						
48	0011	0000	30	12	11	0	9	8	1	F030						
49		0001	31				9		1	1010						
50		0010	32				9		2	0810						
51		0011	33				9		3	0410						
52		0100	34				9		4	0210	PN	Punch On				
53*		0101	35				9		5	0110	RS	Reader Stop		0D (5)	09	(4)
54*		0110	36				9		6	0090	UC	Upper Case		0E (5)		
55		0111	37				9		7	0050	EOT	End of Trans.				
56		1000	38				9	8		0030						
57		1001	39				9	8	1	1030						
58		1010	3A				9	8	2	0830						
59		1011	3B				9	8	3	0430						
60		1100	3C				9	8	4	0230						
61		1101	3D				9	8	5	0130						
62		1110	3E				9	8	6	00B0						
63		1111	3F				9	8	7	0070						

NOTES: Typewriter Output
(1) Tabulate
(2) Shift to black
(3) Carrier Return
(4) Shift to red
(5) The Same in Either Case

Ref No.	EBCDIC Binary 0123	EBCDIC Binary 4567	EBCDIC Hex	IBM Card Code Rows 12	11	0	9	8	7-1	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
64*	0100	0000	40	no punches						0000	(space)	*	10 (5)	21
65		0001	41	12		0	9		1	B010				
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	¢		20 (U)	02
75*		1011	4B	12				8	3	8420	. (period)	4B	6B (L)	00
76*		1100	4C	12				8	4	8220	<		02 (U)	DE
77*		1101	4D	12				8	5	8120	(	4D	19 (U)	FE
78*		1110	4E	12				8	6	80A0	+	4E	70 (U)	DA
79*		1111	4F	12				8	7	8060	| (logical OR)		3B (U)	C6
80*	0101	0000	50	12						8000	&	50	70 (L)	44
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				
90*		1010	5A		11			8	2	4820	!		5B (U)	42
91*		1011	5B		11			8	3	4420	$	5B	5B (L)	40
92*		1100	5C		11			8	4	4220	*	5C	08 (U)	D6
93*		1101	5D		11			8	5	4120	)	5D	1A (U)	F6
94*		1110	5E		11			8	6	40A0	;		13 (U)	D2
95*		1111	5F		11			8	7	4060	¬ (logical NOT)		6B (U)	F2
96*	0110	0000	60		11					4000	- (dash)	60	40 (L)	84
97*		0001	61			0			1	3000	/	61	31 (L)	BC
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
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
126*		1110	7E					8	6	00A0	=	7E	01 (U)	C2
127*		1111	7F					8	7	0060	"		0B (U)	E2

*Any code other than those defined will be interpreted by PRNT1 as a space.

Ref No.	EBCDIC Binary 0123	EBCDIC Binary 4567	EBCDIC Hex	IBM Card Code Rows 12	11	0	9	8	7-1	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
128	1000	0000	80	12		0		8	1	B020				
129		0001	81	12		0			1	B000	a			
130		0010	82	12		0			2	A800	b			
131		0011	83	12		0			3	A400	c			
132		0100	84	12		0			4	A200	d			
133		0101	85	12		0			5	A100	e			
134		0110	86	12		0			6	A080	f			
135		0111	87	12		0			7	A040	g			
136		1000	88	12		0		8		A020	h			
137		1001	89	12		0	9			A010	i			
138		1010	8A	12		0		8	2	A820				
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	j			
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				
162		0010	A2		11	0			2	6800	s			
163		0011	A3		11	0			3	6400	t			
164		0100	A4		11	0			4	6200	u			
165		0101	A5		11	0			5	6100	v			
166		0110	A6		11	0			6	6080	w			
167		0111	A7		11	0			7	6040	x			
168		1000	A8		11	0		8		6020	y			
169		1001	A9		11	0	9			6010	z			
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 Binary 0123	EBCDIC Binary 4567	EBCDIC Hex	IBM Card Code Rows 12	11	0	9	8	7-1	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
192	1100	0000	C0	12		0				A000	(+ zero)			
193*		0001	C1	12					1	9000	A	C1	61 (U)	3C or 3E
194*		0010	C2	12					2	8800	B	C2	62 (U)	18 or 1A
195*		0011	C3	12					3	8400	C	C3	73 (U)	1C or 1E
196*		0100	C4	12					4	8200	D	C4	64 (U)	30 or 32
197*		0101	C5	12					5	8100	E	C5	75 (U)	34 or 36
198*		0110	C6	12					6	8080	F	C6	76 (U)	10 or 12
199*		0111	C7	12					7	8040	G	C7	67 (U)	14 or 16
200*		1000	C8	12				8		8020	H	C8	68 (U)	24 or 26
201*		1001	C9	12			9			8010	I	C9	79 (U)	20 or 22
202		1010	CA	12		0	9	8	2	A830				
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	A0B0				
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	C0B0				
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	60B0				
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	E0B0				
255		1111	FF	12	11	0	9	8	7	E070				

The tables printed below are used to convert decimal numbers to hexadecimal and hexadecimal numbers to decimal. In the descriptions that follow, the explanation of each step is followed by an example in parentheses.

<u>Decimal to Hexadecimal Conversion.</u> Locate the decimal number (0489) in the body of the table. The two high-order digits (1E) of the hexadecimal number are in the left column on the same line, and the low-order digit (9) is at the top of the column. Thus, the hexadecimal number 1E9 is equal to the decimal number 0489.

<u>Hexadecimal to Decimal Conversion.</u> Locate the first two digits (1E) of the hexadecimal number (1E9) in the left column. Follow the line of figures across the page to the column headed by the low-order digit (9). The decimal number (0489) located at the junction of the horizontal line and the vertical column is the equivalent of the hexadecimal number.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00_	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
01_	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
02_	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
03_	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
04_	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
05_	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
06_	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
07_	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
08_	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
09_	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
0A_	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
0B_	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
0C_	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
0D_	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
0E_	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
0F_	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255
10_	0256	0257	0258	0259	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	0270	0271
11_	0272	0273	0274	0275	0276	0277	0278	0279	0280	0281	0282	0283	0284	0285	0286	0287
12_	0288	0289	0290	0291	0292	0293	0294	0295	0296	0297	0298	0299	0300	0301	0302	0303
13_	0304	0305	0306	0307	0308	0309	0310	0311	0312	0313	0314	0315	0316	0317	0318	0319
14_	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	0330	0331	0332	0333	0334	0335
15_	0336	0337	0338	0339	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	0350	0351
16_	0352	0353	0354	0355	0356	0357	0358	0359	0360	0361	0362	0363	0364	0365	0366	0367
17_	0368	0369	0370	0371	0372	0373	0374	0375	0376	0377	0378	0379	0380	0381	0382	0383
18_	0384	0385	0386	0387	0388	0389	0390	0391	0392	0393	0394	0395	0396	0397	0398	0399
19_	0400	0401	0402	0403	0404	0405	0406	0407	0408	0409	0410	0411	0412	0413	0414	0415
1A_	0416	0417	0418	0419	0420	0421	0422	0423	0424	0425	0426	0427	0428	0429	0430	0431
1B_	0432	0433	0434	0435	0436	0437	0438	0439	0440	0441	0442	0443	0444	0445	0446	0447
1C_	0448	0449	0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	0460	0461	0462	0463
1D_	0464	0465	0466	0467	0468	0469	0470	0471	0472	0473	0474	0475	0476	0477	0478	0479
1E_	0480	0481	0482	0483	0484	0485	0486	0487	0488	0489	0490	0491	0492	0493	0494	0495
1F_	0496	0497	0498	0499	0500	0501	0502	0503	0504	0505	0506	0507	0508	0509	0510	0511
20_	0512	0513	0514	0515	0516	0517	0518	0519	0520	0521	0522	0523	0524	0525	0526	0527
21_	0528	0529	0530	0531	0532	0533	0534	0535	0536	0537	0538	0539	0540	0541	0542	0543
22_	0544	0545	0546	0547	0548	0549	0550	0551	0552	0553	0554	0555	0556	0557	0558	0559
23_	0560	0561	0562	0563	0564	0565	0566	0567	0568	0569	0570	0571	0572	0573	0574	0575
24_	0576	0577	0578	0579	0580	0581	0582	0583	0584	0585	0586	0587	0588	0589	0590	0591
25_	0592	0593	0594	0595	0596	0597	0598	0599	0600	0601	0602	0603	0604	0605	0606	0607
26_	0608	0609	0610	0611	0612	0613	0614	0615	0616	0617	0618	0619	0620	0621	0622	0623
27_	0624	0625	0626	0627	0628	0629	0630	0631	0632	0633	0634	0635	0636	0637	0638	0639
28_	0640	0641	0642	0643	0644	0645	0646	0647	0648	0649	0650	0651	0652	0653	0654	0655
29_	0656	0657	0658	0659	0660	0661	0662	0663	0664	0665	0666	0667	0668	0669	0670	0671
2A_	0672	0673	0674	0675	0676	0677	0678	0679	0680	0681	0682	0683	0684	0685	0686	0687
2B_	0688	0689	0690	0691	0692	0693	0694	0695	0696	0697	0698	0699	0700	0701	0702	0703
2C_	0704	0705	0706	0707	0708	0709	0710	0711	0712	0713	0714	0715	0716	0717	0718	0719
2D_	0720	0721	0722	0723	0724	0725	0726	0727	0728	0729	0730	0731	0732	0733	0734	0735
2E_	0736	0737	0738	0739	0740	0741	0742	0743	0744	0745	0746	0747	0748	0749	0750	0751
2F_	0752	0753	0754	0755	0756	0757	0758	0759	0760	0761	0762	0763	0764	0765	0766	0767
30_	0768	0769	0770	0771	0772	0773	0774	0775	0776	0777	0778	0779	0780	0781	0782	0783
31_	0784	0785	0786	0787	0788	0789	0790	0791	0792	0793	0794	0795	0796	0797	0798	0799
32_	0800	0801	0802	0803	0804	0805	0806	0807	0808	0809	0810	0811	0812	0813	0814	0815
33_	0816	0817	0818	0819	0820	0821	0822	0823	0824	0825	0826	0827	0828	0829	0830	0831
34_	0832	0833	0834	0835	0836	0837	0838	0839	0840	0841	0842	0843	0844	0845	0846	0847
35_	0848	0849	0850	0851	0852	0853	0854	0855	0856	0857	0858	0859	0860	0861	0862	0863
36_	0864	0865	0866	0867	0868	0869	0870	0871	0872	0873	0874	0875	0876	0877	0878	0879
37_	0880	0881	0882	0883	0884	0885	0886	0887	0888	0889	0890	0891	0892	0893	0894	0895
38_	0896	0897	0898	0899	0900	0901	0902	0903	0904	0905	0906	0907	0908	0909	0910	0911
39_	0912	0913	0914	0915	0916	0917	0918	0919	0920	0921	0922	0923	0924	0925	0926	0927
3A_	0928	0929	0930	0931	0932	0933	0934	0935	0936	0937	0938	0939	0940	0941	0942	0943
3B_	0944	0945	0946	0947	0948	0949	0950	0951	0952	0953	0954	0955	0956	0957	0958	0959
3C_	0960	0961	0962	0963	0964	0965	0966	0967	0968	0969	0970	0971	0972	0973	0974	0975
3D_	0976	0977	0978	0979	0980	0981	0982	0983	0984	0985	0986	0987	0988	0989	0990	0991
3E_	0992	0993	0994	0995	0996	0997	0998	0999	1000	1001	1002	1003	1004	1005	1006	1007
3F_	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	1018	1019	1020	1021	1022	1023

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
40_	1024	1025	1026	1027	1028	1029	1030	1031	1032	1033	1034	1035	1036	1037	1038	1039
41_	1040	1041	1042	1043	1044	1045	1046	1047	1048	1049	1050	1051	1052	1053	1054	1055
42_	1056	1057	1058	1059	1060	1061	1062	1063	1064	1065	1066	1067	1068	1069	1070	1071
43_	1072	1073	1074	1075	1076	1077	1078	1079	1080	1081	1082	1083	1084	1085	1086	1087
44_	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	1102	1103
45_	1104	1105	1106	1107	1108	1109	1110	1111	1112	1113	1114	1115	1116	1117	1118	1119
46_	1120	1121	1122	1123	1124	1125	1126	1127	1128	1129	1130	1131	1132	1133	1134	1135
47_	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151
48_	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167
49_	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183
4A_	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199
4B_	1200	1201	1202	1203	1204	1205	1206	1207	1208	1209	1210	1211	1212	1213	1214	1215
4C_	1216	1217	1218	1219	1220	1221	1222	1223	1224	1225	1226	1227	1228	1229	1230	1231
4D_	1232	1233	1234	1235	1236	1237	1238	1239	1240	1241	1242	1243	1244	1245	1246	1247
4E_	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263
4F_	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279
50_	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295
51_	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311
52_	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	1325	1326	1327
53_	1328	1329	1330	1331	1332	1333	1334	1335	1336	1337	1338	1339	1340	1341	1342	1343
54_	1344	1345	1346	1347	1348	1349	1350	1351	1352	1353	1354	1355	1356	1357	1358	1359
55_	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375
56_	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391
57_	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407
58_	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423
59_	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439
5A_	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455
5B_	1456	1457	1458	1459	1460	1461	1462	1463	1464	1465	1466	1467	1468	1469	1470	1471
5C_	1472	1473	1474	1475	1476	1477	1478	1479	1480	1481	1482	1483	1484	1485	1486	1487
5D_	1488	1489	1490	1491	1492	1493	1494	1495	1496	1497	1498	1499	1500	1501	1502	1503
5E_	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519
5F_	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535
60_	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551
61_	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567
62_	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583
63_	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599
64_	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615
65_	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631
66_	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647
67_	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663
68_	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679
69_	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695
6A_	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711
6B_	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727
6C_	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743
6D_	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759
6E_	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775
6F_	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791
70_	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807
71_	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823
72_	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839
73_	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855
74_	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871
75_	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887
76_	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903
77_	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919
78_	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935
79_	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951
7A_	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967
7B_	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983
7C_	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
7D_	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015
7E_	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031
7F_	2032	2033	2034	2035	2036	2037	2038	2039	2040	2041	2042	2043	2044	2045	2046	2047

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
80 _	2048	2049	2050	2051	2052	2053	2054	2055	2056	2057	2058	2059	2060	2061	2062	2063
81 _	2064	2065	2066	2067	2068	2069	2070	2071	2072	2073	2074	2075	2076	2077	2078	2079
82 _	2080	2081	2082	2083	2084	2085	2086	2087	2088	2089	2090	2091	2092	2093	2094	2095
83 _	2096	2097	2098	2099	2100	2101	2102	2103	2104	2105	2106	2107	2108	2109	2110	2111
84 _	2112	2113	2114	2115	2116	2117	2118	2119	2120	2121	2122	2123	2124	2125	2126	2127
85 _	2128	2129	2130	2131	2132	2133	2134	2135	2136	2137	2138	2139	2140	2141	2142	2143
86 _	2144	2145	2146	2147	2148	2149	2150	2151	2152	2153	2154	2155	2156	2157	2158	2159
87 _	2160	2161	2162	2163	2164	2165	2166	2167	2168	2169	2170	2171	2172	2173	2174	2175
88 _	2176	2177	2178	2179	2180	2181	2182	2183	2184	2185	2186	2187	2188	2189	2190	2191
89 _	2192	2193	2194	2195	2196	2197	2198	2199	2200	2201	2202	2203	2204	2205	2206	2207
8A _	2208	2209	2210	2211	2212	2213	2214	2215	2216	2217	2218	2219	2220	2221	2222	2223
8B _	2224	2225	2226	2227	2228	2229	2230	2231	2232	2233	2234	2235	2236	2237	2238	2239
8C _	2240	2241	2242	2243	2244	2245	2246	2247	2248	2249	2250	2251	2252	2253	2254	2255
8D _	2256	2257	2258	2259	2260	2261	2262	2263	2264	2265	2266	2267	2268	2269	2270	2271
8E _	2272	2273	2274	2275	2276	2277	2278	2279	2280	2281	2282	2283	2284	2285	2286	2287
8F _	2288	2289	2290	2291	2292	2293	2294	2295	2296	2297	2298	2299	2300	2301	2302	2303
90 _	2304	2305	2306	2307	2308	2309	2310	2311	2312	2313	2314	2315	2316	2317	2318	2319
91 _	2320	2321	2322	2323	2324	2325	2326	2327	2328	2329	2330	2331	2332	2333	2334	2335
92 _	2336	2337	2338	2339	2340	2341	2342	2343	2344	2345	2346	2347	2348	2349	2350	2351
93 _	2352	2353	2354	2355	2356	2357	2358	2359	2360	2361	2362	2363	2364	2365	2366	2367
94 _	2368	2369	2370	2371	2372	2373	2374	2375	2376	2377	2378	2379	2380	2381	2382	2383
95 _	2384	2385	2386	2387	2388	2389	2390	2391	2392	2393	2394	2395	2396	2397	2398	2399
96 _	2400	2401	2402	2403	2404	2405	2406	2407	2408	2409	2410	2411	2412	2413	2414	2415
97 _	2416	2417	2418	2419	2420	2421	2422	2423	2424	2425	2426	2427	2428	2429	2430	2431
98 _	2432	2433	2434	2435	2436	2437	2438	2439	2440	2441	2442	2443	2444	2445	2446	2447
99 _	2448	2449	2450	2451	2452	2453	2454	2455	2456	2457	2458	2459	2460	2461	2462	2463
9A _	2464	2465	2466	2467	2468	2469	2470	2471	2472	2473	2474	2475	2476	2477	2478	2479
9B _	2480	2481	2482	2483	2484	2485	2486	2487	2488	2489	2490	2491	2492	2493	2494	2495
9C _	2496	2497	2498	2499	2500	2501	2502	2503	2504	2505	2506	2507	2508	2509	2510	2511
9D _	2512	2513	2514	2515	2516	2517	2518	2519	2520	2521	2522	2523	2524	2525	2526	2527
9E _	2528	2529	2530	2531	2532	2533	2534	2535	2536	2537	2538	2539	2540	2541	2542	2543
9F _	2544	2545	2546	2547	2548	2549	2550	2551	2552	2553	2554	2555	2556	2557	2558	2559
A0 _	2560	2561	2562	2563	2564	2565	2566	2567	2568	2569	2570	2571	2572	2573	2574	2575
A1 _	2576	2577	2578	2579	2580	2581	2582	2583	2584	2585	2586	2587	2588	2589	2590	2591
A2 _	2592	2593	2594	2595	2596	2597	2598	2599	2600	2601	2602	2603	2604	2605	2606	2607
A3 _	2608	2609	2610	2611	2612	2613	2614	2615	2616	2617	2618	2619	2620	2621	2622	2623
A4 _	2624	2625	2626	2627	2628	2629	2630	2631	2632	2633	2634	2635	2636	2637	2638	2639
A5 _	2640	2641	2642	2643	2644	2645	2646	2647	2648	2649	2650	2651	2652	2653	2654	2655
A6 _	2656	2657	2658	2659	2660	2661	2662	2663	2664	2665	2666	2667	2668	2669	2670	2671
A7 _	2672	2673	2674	2675	2676	2677	2678	2679	2680	2681	2682	2683	2684	2685	2686	2687
A8 _	2688	2689	2690	2691	2692	2693	2694	2695	2696	2697	2698	2699	2700	2701	2702	2703
A9 _	2704	2705	2706	2707	2708	2709	2710	2711	2712	2713	2714	2715	2716	2717	2718	2719
AA _	2720	2721	2722	2723	2724	2725	2726	2727	2728	2729	2730	2731	2732	2733	2734	2735
AB _	2736	2737	2738	2739	2740	2741	2742	2743	2744	2745	2746	2747	2748	2749	2750	2751
AC _	2752	2753	2754	2755	2756	2757	2758	2759	2760	2761	2762	2763	2764	2765	2766	2767
AD _	2768	2769	2770	2771	2772	2773	2774	2775	2776	2777	2778	2779	2780	2781	2782	2783
AE _	2784	2785	2786	2787	2788	2789	2790	2791	2792	2793	2794	2795	2796	2797	2798	2799
AF _	2800	2801	2802	2803	2804	2805	2806	2807	2808	2809	2810	2811	2812	2813	2814	2815
B0 _	2816	2817	2818	2819	2820	2821	2822	2823	2824	2825	2826	2827	2828	2829	2830	2831
B1 _	2832	2833	2834	2835	2836	2837	2838	2839	2840	2841	2842	2843	2844	2845	2846	2847
B2 _	2848	2849	2850	2851	2852	2853	2854	2855	2856	2857	2858	2859	2860	2861	2862	2863
B3 _	2864	2865	2866	2867	2868	2869	2870	2871	2872	2873	2874	2875	2876	2877	2878	2879
B4 _	2880	2881	2882	2883	2884	2885	2886	2887	2888	2889	2890	2891	2892	2893	2894	2895
B5 _	2896	2897	2898	2899	2900	2901	2902	2903	2904	2905	2906	2907	2908	2909	2910	2911
B6 _	2912	2913	2914	2915	2916	2917	2918	2919	2920	2921	2922	2923	2924	2925	2926	2927
B7 _	2928	2929	2930	2931	2932	2933	2934	2935	2936	2937	2938	2939	2940	2941	2942	2943
B8 _	2944	2945	2946	2947	2948	2949	2950	2951	2952	2953	2954	2955	2956	2957	2958	2959
B9 _	2960	2961	2962	2963	2964	2965	2966	2967	2968	2969	2970	2971	2972	2973	2974	2975
BA _	2976	2977	2978	2979	2980	2981	2982	2983	2984	2985	2986	2987	2988	2989	2990	2991
BB _	2992	2993	2994	2995	2996	2997	2998	2999	3000	3001	3002	3003	3004	3005	3006	3007
BC _	3008	3009	3010	3011	3012	3013	3014	3015	3016	3017	3018	3019	3020	3021	3022	3023
BD _	3024	3025	3026	3027	3028	3029	3030	3031	3032	3033	3034	3035	3036	3037	3038	3039
BE _	3040	3041	3042	3043	3044	3045	3046	3047	3048	3049	3050	3051	3052	3053	3054	3055
BF _	3056	3057	3058	3059	3060	3061	3062	3063	3064	3065	3066	3067	3068	3069	3070	3071

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
C0 _	3072	3073	3074	3075	3076	3077	3078	3079	3080	3081	3082	3083	3084	3085	3086	3087
C1 _	3088	3089	3090	3091	3092	3093	3094	3095	3096	3097	3098	3099	3100	3101	3102	3103
C2 _	3104	3105	3106	3107	3108	3109	3110	3111	3112	3113	3114	3115	3116	3117	3118	3119
C3 _	3120	3121	3122	3123	3124	3125	3126	3127	3128	3129	3130	3131	3132	3133	3134	3135
C4 _	3136	3137	3138	3139	3140	3141	3142	3143	3144	3145	3146	3147	3148	3149	3150	3151
C5 _	3152	3153	3154	3155	3156	3157	3158	3159	3160	3161	3162	3163	3164	3165	3166	3167
C6 _	3168	3169	3170	3171	3172	3173	3174	3175	3176	3177	3178	3179	3180	3181	3182	3183
C7 _	3184	3185	3186	3187	3188	3189	3190	3191	3192	3193	3194	3195	3196	3197	3198	3199
C8 _	3200	3201	3202	3203	3204	3205	3206	3207	3208	3209	3210	3211	3212	3213	3214	3215
C9 _	3216	3217	3218	3219	3220	3221	3222	3223	3224	3225	3226	3227	3228	3229	3230	3231
CA _	3232	3233	3234	3235	3236	3237	3238	3239	3240	3241	3242	3243	3244	3245	3246	3247
CB _	3248	3249	3250	3251	3252	3253	3254	3255	3256	3257	3258	3259	3260	3261	3262	3263
CC _	3264	3265	3266	3267	3268	3269	3270	3271	3272	3273	3274	3275	3276	3277	3278	3279
CD _	3280	3281	3282	3283	3284	3285	3286	3287	3288	3289	3290	3291	3292	3293	3294	3295
CE _	3296	3297	3298	3299	3300	3301	3302	3303	3304	3305	3306	3307	3308	3309	3310	3311
CF _	3312	3313	3314	3315	3316	3317	3318	3319	3320	3321	3322	3323	3324	3325	3326	3327
D0 _	3328	3329	3330	3331	3332	3333	3334	3335	3336	3337	3338	3339	3340	3341	3342	3343
D1 _	3344	3345	3346	3347	3348	3349	3350	3351	3352	3353	3354	3355	3356	3357	3358	3359
D2 _	3360	3361	3362	3363	3364	3365	3366	3367	3368	3369	3370	3371	3372	3373	3374	3375
D3 _	3376	3377	3378	3379	3380	3381	3382	3383	3384	3385	3386	3387	3388	3389	3390	3391
D4 _	3392	3393	3394	3395	3396	3397	3398	3399	3400	3401	3402	3403	3404	3405	3406	3407
D5 _	3408	3409	3410	3411	3412	3413	3414	3415	3416	3417	3418	3419	3420	3421	3422	3423
D6 _	3424	3425	3426	3427	3428	3429	3430	3431	3432	3433	3434	3435	3436	3437	3438	3439
D7 _	3440	3441	3442	3443	3444	3445	3446	3447	3448	3449	3450	3451	3452	3453	3454	3455
D8 _	3456	3457	3458	3459	3460	3461	3462	3463	3464	3465	3466	3467	3468	3469	3470	3471
D9 _	3472	3473	3474	3475	3476	3477	3478	3479	3480	3481	3482	3483	3484	3485	3486	3487
DA _	3488	3489	3490	3491	3492	3493	3494	3495	3496	3497	3498	3499	3500	3501	3502	3503
DB _	3504	3505	3506	3507	3508	3509	3510	3511	3512	3513	3514	3515	3516	3517	3518	3519
DC _	3520	3521	3522	3523	3524	3525	3526	3527	3528	3529	3530	3531	3532	3533	3534	3535
DD _	3536	3537	3538	3539	3540	3541	3542	3543	3544	3545	3546	3547	3548	3549	3550	3551
DE _	3552	3553	3554	3555	3556	3557	3558	3559	3560	3561	3562	3563	3564	3565	3566	3567
DF _	3568	3569	3570	3571	3572	3573	3574	3575	3576	3577	3578	3579	3580	3581	3582	3583
E0 _	3584	3585	3586	3587	3588	3589	3590	3591	3592	3593	3594	3595	3596	3597	3598	3599
E1 _	3600	3601	3602	3603	3604	3605	3606	3607	3608	3609	3610	3611	3612	3613	3614	3615
E2 _	3616	3617	3618	3619	3620	3621	3622	3623	3624	3625	3626	3627	3628	3629	3630	3631
E3 _	3632	3633	3634	3635	3636	3637	3638	3639	3640	3641	3642	3643	3644	3645	3646	3647
E4 _	3648	3649	3650	3651	3652	3653	3654	3655	3656	3657	3658	3659	3660	3661	3662	3663
E5 _	3664	3665	3666	3667	3668	3669	3670	3671	3672	3673	3674	3675	3676	3677	3678	3679
E6 _	3680	3681	3682	3683	3684	3685	3686	3687	3688	3689	3690	3691	3692	3693	3694	3695
E7 _	3696	3697	3698	3699	3700	3701	3702	3703	3704	3705	3706	3707	3708	3709	3710	3711
E8 _	3712	3713	3714	3715	3716	3717	3718	3719	3720	3721	3722	3723	3724	3725	3726	3727
E9 _	3728	3729	3730	3731	3732	3733	3734	3735	3736	3737	3738	3739	3740	3741	3742	3743
EA _	3744	3745	3746	3747	3748	3749	3750	3751	3752	3753	3754	3755	3756	3757	3758	3759
EB _	3760	3761	3762	3763	3764	3765	3766	3767	3768	3769	3770	3771	3772	3773	3774	3775
EC _	3776	3777	3778	3779	3780	3781	3782	3783	3784	3785	3786	3787	3788	3789	3790	3791
ED _	3792	3793	3794	3795	3796	3797	3798	3799	3800	3801	3802	3803	3804	3805	3806	3807
EE _	3808	3809	3810	3811	3812	3813	3814	3815	3816	3817	3818	3819	3820	3821	3822	3823
EF _	3824	3825	3826	3827	3828	3829	3830	3831	3832	3833	3834	3835	3836	3837	3838	3839
F0 _	3840	3841	3842	3843	3844	3845	3846	3847	3848	3849	3850	3851	3852	3853	3854	3855
F1 _	3856	3857	3858	3859	3860	3861	3862	3863	3864	3865	3866	3867	3868	3869	3870	3871
F2 _	3872	3873	3874	3875	3876	3877	3878	3879	3880	3881	3882	3883	3884	3885	3886	3887
F3 _	3888	3889	3890	3891	3892	3893	3894	3895	3896	3897	3898	3899	3900	3901	3902	3903
F4 _	3904	3905	3906	3907	3908	3909	3910	3911	3912	3913	3914	3915	3916	3917	3918	3919
F5 _	3920	3921	3922	3923	3924	3925	3926	3927	3928	3929	3930	3931	3932	3933	3934	3935
F6 _	3936	3937	3938	3939	3940	3941	3942	3943	3944	3945	3946	3947	3948	3949	3950	3951
F7 _	3952	3953	3954	3955	3956	3957	3958	3959	3960	3961	3962	3963	3964	3965	3966	3967
F8 _	3968	3969	3970	3971	3972	3973	3974	3975	3976	3977	3978	3979	3980	3981	3982	3983
F9 _	3984	3985	3986	3987	3988	3989	3990	3991	3992	3993	3994	3995	3996	3997	3998	3999
FA _	4000	4001	4002	4003	4004	4005	4006	4007	4008	4009	4010	4011	4012	4013	4014	4015
FB _	4016	4017	4018	4019	4020	4021	4022	4023	4024	4025	4026	4027	4028	4029	4030	4031
FC _	4032	4033	4034	4035	4036	4037	4038	4039	4040	4041	4042	4043	4044	4045	4046	4047
FD _	4048	4049	4050	4051	4052	4053	4054	4055	4056	4057	4058	4059	4060	4061	4062	4063
FE _	4064	4065	4066	4067	4068	4069	4070	4071	4072	4073	4074	4075	4076	4077	4078	4079
FF _	4080	4081	4082	4083	4084	4085	4086	4087	4088	4089	4090	4091	4092	4093	4094	4095

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

The table to the left gives the decimal, binary, and hexadecimal coding for the full range of four binary bits, from zero through F_{16} and 15_{10}.

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 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.

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		

C26-5927-2

IBM 1130 Printed in U. S. A. C26-5927-2

IBM

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

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

READER'S COMMENT FORM

IBM 1130 Assembler Language — Form C26-5927-2

- 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?	☐	☐
• Did you find the material:		
Easy to read and understand?	☐	☐
Organized for convenient use?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Written for your technical level?	☐	☐

- What is your occupation? ____________________
- How do you use this publication?

As an introduction to the subject?	☐	As an instructor in a class?	☐
For advanced knowledge of the subject?	☐	As a student in a class?	☐
For information about operating procedures?	☐	As a reference manual?	☐

Other ____________________

- Please give specific page and line references with your comments when appropriate. If you wish a reply, be sure to include your name and address.

COMMENTS

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

C26-5927-2

fold

fold

FIRST CLASS
PERMIT NO. 2078
SAN JOSE, CALIF.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY . . .

IBM Corporation
Monterey & Cottle Rds.
San Jose, California
95114

Attention: Programming Publications, Dept. 452

fold

fold

IBM 1130 Printed in U.S.A. C26-5927-2

IBM®

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

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]