File No. 1130-30
Form C26-5929-0

IBM Systems Reference Library

IBM 1130 Subroutine Library

This publication describes the subroutines in the IBM 1130 Subroutine Library. The library consists of Input/Output, Conversion, Arithmetic and Functional, and Selective Dump subroutines. Included in the descriptions are calling sequences for the subroutines and explanations of the parameters involved.

The section on Conversion subroutines describes the codes used to communicate with the 1130 System input/output devices. An appendix lists the codes, and shows their relationship to each other.

Copies of this and other IBM publications can be obtained through IBM Branch Offices.
Comments concerning the contents of this publication may be addressed to:
IBM, Product Publications Department, San Jose, Calif. 95114

CONTENTS

PREFACE

This publication describes the methods by which the programmer can use the IBM 1130 Subroutine Library to increase the efficiency of his programs and decrease the time necessary for writing and testing them. The Subroutine Library contains input/output, data conversion, arithmetic and functional, and selective dump subroutines. These subroutines are available for use with both the 1130 Assembler and the 1130 FORTRAN Compiler. When using the assembler, the user calls the subroutines via a calling sequence. When using the FORTRAN compiler the appropriate subroutines are called by the compiler whenever a read, write, or arithmetic statement is encountered. This publication describes each subroutine and the calling sequences to be used.

The reader is provided with sufficient information about the IBM 1130 subroutines so that plans can be made to use them when the system becomes available. It is assumed that the reader is familiar with the methods of data handling and the functions of instructions used in the IBM 1130 Computing System. He must also be familiar with the assembler or compiler to be used in conjunction with the subroutines. The following IBM publications provide this information:

- IBM 1130 Computing System Principles of Operation (A26-5881)
- IBM 1130 Computing System Input/Output Units (A26-5890)
- IBM 1130 FORTRAN Language (Form C26-5933)
- IBM 1130 Assembler Language (Form C26-5927)

MACHINE CONFIGURATION

The use of the Subroutine Library requires the following machine configuration:

- IBM 1131 Central Processing Unit, with a minimum of 4096 words of core storage
- IBM 1442 Card Read Punch, or IBM 1054 Paper Tape Reader with IBM 1055 Paper Tape Punch

In addition, the following input/output units and features can be controlled by the Input/Output section of the Subroutine Library:

- Console Keyboard
- Console Printer
- Disk Storage
- IBM 1132 Printer
- IBM 1627 Plotter

It is very often necessary to repeat the same group of instructions many times during the execution of a program. Examples of this are the series of instructions necessary for decimal-to-binary conversion, computing square roots, or reading from a card reader. It is not desirable to write out the necessary instructions each time a function is needed. Instead, the instructions needed are written only once and the main program is then arranged to transfer to this block of instructions each time they are required. Such a block of instructions is called a "subroutine."

These subroutines normally perform such basic functions that they may be used in the solution of many types of problems. For example, a subroutine which computes a square root can be used in a wide variety of problems. Another example of such a subroutine would be one which reads data from an input device and stores it in the computer.

There are two methods of using subroutines with respect to the main program. One method is to insert the subroutine into the main program at each point where it is to be used. Subroutines designed for this type of usage are called "open subroutines." The open subroutine is "sandwiched" into a program as though it were part of the original coding of the program. This type of subroutine usage is normally restricted to the cases where the main program uses the subroutine only once.

When the main program uses a subroutine several times, which is the common situation, it is apparent that the open subroutine is not desirable. Here, the second method of employing subroutines is used. The subroutine used in these situations is called a "closed subroutine." A closed subroutine may be executed several times within one main program, but the set of instructions comprising the subroutine need appear only once. The transfer of control from the main program to the subroutine takes place from a set of instructions known as the calling sequence or basic linkage. The calling sequence transfers control to the subroutine, and through parameters, gives the subroutine any control information required.

The parameters of a calling sequence vary with the type of subroutine being called. For example, an input/output subroutine requires several parameters to identify the associated input/output device, storage area, amount of data to be transferred, etc., whereas an arithmetic/functional subroutine requires, at most, one parameter representing an argument. The calling sequences used with the 1130 System subroutines take the form of a CALL statement which specifies the subroutine, followed by DC statements which make up the parameter list. The calling sequences for the various subroutines are presented later in this manual.

The subroutines for each class of I/O equipment are self-contained, so that only those subroutines required by the current job are in core storage at program execution time.

The IBM 1130 input/output subroutines were designed for one purpose - to reduce the amount of time spent by the programmer in accomplishing the input and output of data from and to the various input/output (I/O) devices attached to the computer. They handle all of the details peculiar to each device, including the usually complex interrupt functions, and are capable of controlling many input/output devices simultaneously and asynchronously. In assuming the burden of the details of I/O operation, the subroutines will permit user attention to be directed to the problem-solving aspects of each individual job, rather than accessory I/O "housekeeping."

In order to better understand the subsequent descriptions of the individual I/O subroutines, the user should be familiar with certain characteristics which are common to the I/O subroutines, namely:

- Methods of data transfer
- Interrupt servicing capabilities
- Subroutine operation
- General error handling procedures
- Basic calling sequence

METHODS OF DATA TRANSFER

IBM 1130 I/O devices and their related subroutines can be grouped according to their method of transmitting and/or receiving data. There are two basic groups. The first operates via direct program control. Direct program control requires a programmed I/O operation for each word or character transferred. A character interrupt occurs whenever a character I/O operation is completed. This method is used for the following low-speed serial devices: 1442 Card Read Punch, 1054/1055 Paper Tape Attachment, Console Printer, Console Keyboard, 1132 Printer, and 1627 Plotter.

The second group operates via a Data Channel. A Data Channel requires an I/O operation only to initiate the data transfer. The device is provided with control information, word-counts, and data from the user's I/O area. Once initiated, the transfer takes place completely asynchronous to program execution. An operation-complete interrupt signals the end of the I/O operation when all of the data has been transmitted. The Data Channel is used for Disk Storage.

INTERRUPT SERVICING CAPABILITIES

The I/O subroutine package assumes the responsibility of servicing all input/output interrupts. This is accomplished via a set of interrupt identification routines which are loaded as part of the Subroutine Library. There is one interrupt identification routine for each interrupt level being used. This routine determines which device on that level caused the interrupt, preserves the contents of any registers to be used by the I/O subroutines, and transmits identifying information to the I/O subroutines.

The interrupt identification routines are loaded following the I/O subroutines. With this arrangement, the loader can load only those identification routines that are required. For example, if the user's main program does not call the card subroutine, there is no need to load the routine associated with interrupt level 0 since no interrupts will be forthcoming on that level.

When these routines are loaded, the core addresses assigned to them are inserted into the computer words which were reserved for that purpose (word 8 for interrupt level 0, 9 for 1, etc.). Interrupts occurring during execution of the user's program cause an automatic "Branch Indirect", via the interrupt level word, to the correct interrupt identification routine.

I/O SUBROUTINE OPERATION

This section briefly describes the internal makeup of the I/O subroutines. This description, along with some basic flow charts, will make it easier for the reader to understand the individual subroutine descriptions presented later in the manual.

Makeup of an I/O Subroutine

Each I/O subroutine is divided into two routines: a call routine and an interrupt response routine. The call routine is entered when a user's calling sequence is executed; the interrupt response routine is entered as a result of an I/O interrupt.

Call Routine

The call routine illustrated in Figure 1 has four basic functions:

1. Determine if any previous operations on the specified device are still in process
2. Check the calling sequence for legality
3. Save the calling sequence
4. Initiate the requested I/O operation

The flow diagram (Figure 1) is not exact for any one I/O subroutine. It is only a general picture of the internal operation of a call routine.

Determine Previous Operation. This function can be performed by simply using a programmed routine busy indicator to determine if the previous I/O operation is complete. The CARD1 subroutine is a good example. If an operation is started on the 1442, a subsequent CALL CARD1 for the 1442 will not be honored until the routine busy indicator is turned off. Of course, a call to any other I/O subroutine, such as TYPE1, will not be affected by the fact that the CARD1 subroutine is busy.

Save Calling Sequence. The call routine saves within itself all of the calling sequence information needed to perform an I/O operation. The user may modify a calling sequence even though an I/O operation is not yet complete. However, the I/O data area must be

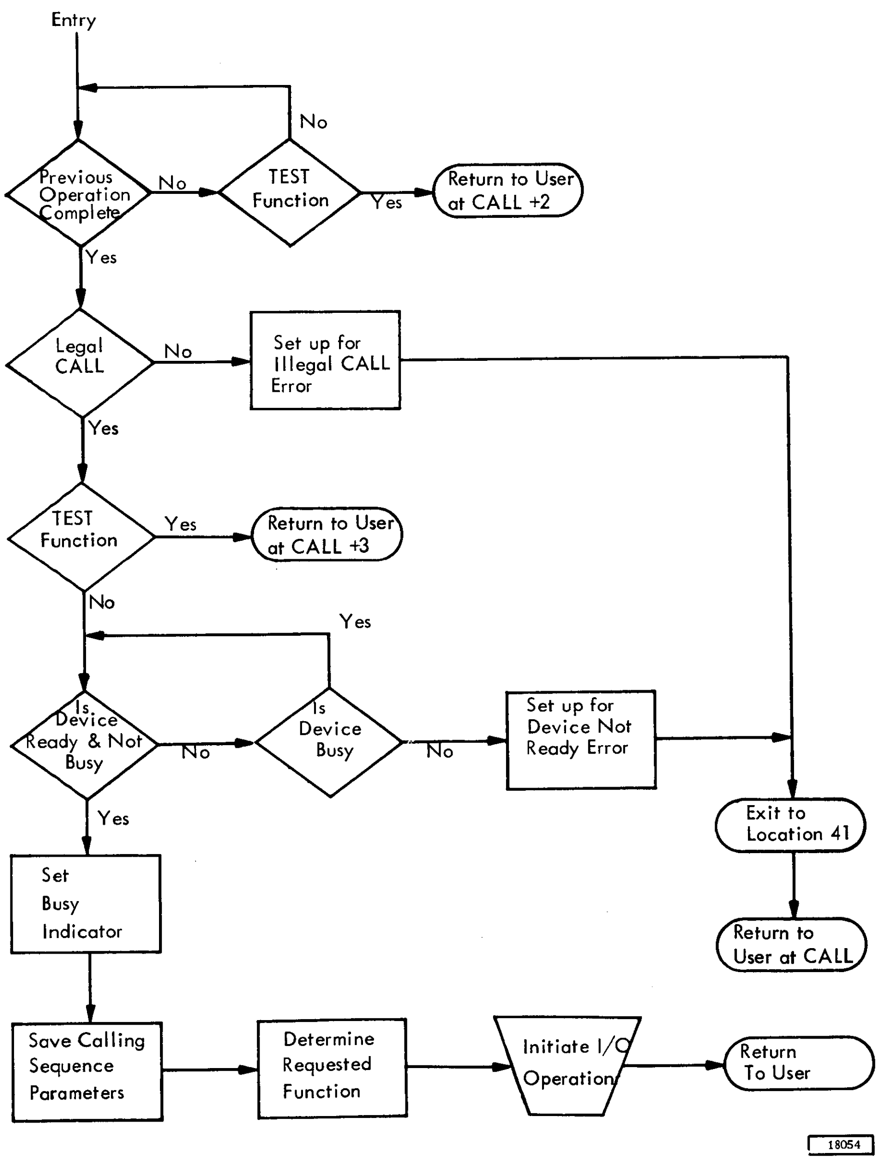


Figure 1. Diagram of Call Routine

left intact during an operation because the I/O subroutine is continually accessing and modifying that area.

Check legality of Calling Sequence. Calling sequences are checked for such items as illegal function character, illegal device identification code, etc.

Initiate I/O operation. The call routine only initiates an I/O operation. Subsequent character interrupts or operation complete interrupts are handled by the interrupt response routine.

Interrupt Response Routine

The I/O interrupt response routine, illustrated in Figure 2, is entered as a result of an I/O interrupt. The interrupt causes the user's program to exit to an interrupt identification routine which in turn exits to the I/O interrupt response routine. The interrupt response routine checks for errors, does any necessary data manipulation, initiates character operations, and initiates retry operations in case of errors. It then returns control to the interrupt identification routine which returns control to the user.

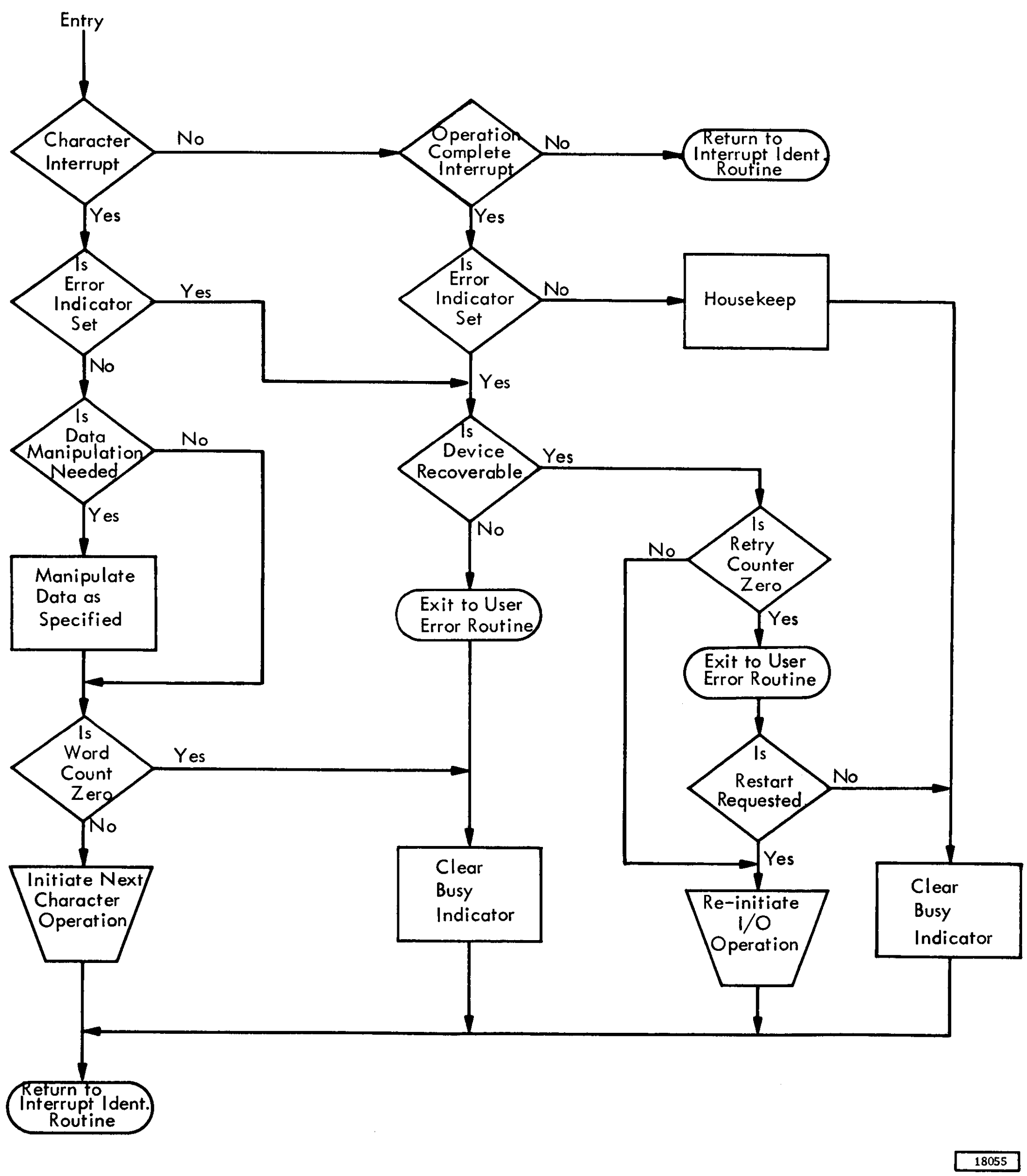


Figure 2. Diagram of Interrupt Response Routine

Character interrupts occur for devices under direct program control whenever a piece of data may be read or written; e.g., a card column punched or a paper tape character read. Operation complete interrupts occur for the disk storage on the Data Channel (and the 1442 under direct program control) whenever the specified block of data has been read or written, e.g., disk record read.

Error detection and recovery procedures are an important part of an I/O subroutine. However, little or nothing can be done about reinitiating an operation until a character interrupt or operation complete interrupt occurs. Therefore, the error indicators are not examined until one of these interrupts occurs.

A recoverable device is one which can be easily repositioned by the subroutine or operator and the I/O operation reinitiated. If the device is not recoverable or if the error cannot be corrected after a specified number of retries, the user is informed of the error condition. If the device is recoverable, the user may request, via his error routine, that the operation be reinitiated.

GENERAL ERROR HANDLING PROCEDURES

Each I/O subroutine has its own error detecting routines. (In this context, the term "error" includes such conditions as last card, channel 9, etc.). These routines categorize the error and choose an error procedure. Errors can be divided into two categories: those that are detected before an I/O operation is initiated, and those that detected after an I/O operation has been initiated. Appendix A contains a list of the errors detected by the I/O subroutines.

Pre-operation Checks

Before an I/O subroutine initiates an I/O operation, it checks the status of the device and the legality of the calling sequence parameters. If the device is not ready or a parameter is in error, the I/O subroutine stores the address of the CALL statement in core location 40 and exits to core location 41. The A-Register is loaded with an error code which defines one of the errors (see Appendix A).

The loader stores a WAIT instruction in core location 41 and an indirect branch instruction (BSC I 40) in locations 42 and 43. The user may replace these two instructions with an exit to his own error routine.

Post-Operation Checks

After an I/O operation has been started, certain conditions may be detected, about which the user should be informed. These conditions may be card jams for which manual intervention is needed to continue, read checks which have not been corrected after a specified number of retries, or indications of equipment readiness, such as a channel 12 indicator.

All of these conditions are detected during execution of the I/O interrupt response routine. (See Subroutine Operation.) The error procedure here is to execute a Branch and Store Instruction Counter instruction (BSI) to the error routine address specified in the related calling sequence. Identifying information will be placed in the A-Register (see Appendix A). When the error routine at that address returns control to the I/O subroutine (using the return link), the I/O subroutine examines the A-Register. If the user clears the A-Register before returning to the I/O subroutine, he is requesting that the error condition be ignored and that the operation be terminated. If the user does not clear the A-Register, he is requesting that the entire operation be restarted, in which case the I/O subroutine reinitiates the operation before returning to the user's main program.

NOTE: The user's error routine must return to the I/O subroutine, and must do so via the return link.

BASIC CALLING SEQUENCE

Each of the I/O subroutines described in this manual is entered via a calling sequence. These calling sequences follow a basic pattern; in fact, some look identical except for the name of the subroutine being called. In order not to burden the reader with redundant descriptions, this section presents the basic calling sequence and describes those parameters which are common to most of the subroutines.

BASIC CALLING SEQUENCE

```
CALL    Name
DC      Control parameter
DC      I/O area
DC      Error routine
```

This calling sequence, with the parameters shown, is basic to most of the input/output subroutines. Detailed descriptions of the above four parameters are omitted when the subroutines are described later in the manual. Unless otherwise specified, the subroutine returns control to the instruction immediately following the last parameter.

Name Parameter

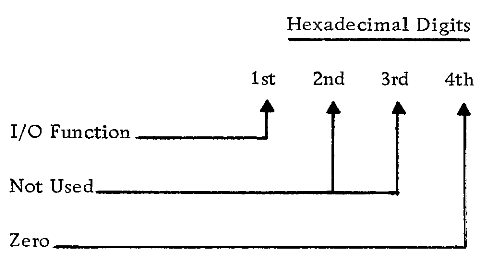
Each subroutine has a symbolic name. That name must be written in the CALL statement exactly as listed in Table 1. The name is recognized by the object program loader and the proper linkage is generated.

Control Parameter

This parameter, in the form of four hexadecimal digits, conveys necessary control data to the particular input/output subroutine.

It specifies the desired function (read, write, etc.), and other similar control information. Most subroutines do not have use for all four digits.

A typical control parameter is illustrated below:



NOTE: With the exception of the test function on paper tape, the 4th digit must be zero.

Table 1. I/O Subroutine Names

Subroutine	Name
Card	CARD1
Disk	DISK1
Printer	PRNT1
Keyboard/Console Printer	TYPE1
Console Printer	WRTY1
Paper Tape	PAPT
Plotter	PLOT

19024

Since the I/O function item is used in all subroutines, a description of its purpose is given here.

I/O Function

Each device has a set of functions which it is capable of performing. The function digit in the calling sequence specifies which I/O operation the user is requesting. Three of these functions, Read, Write and Test are used in most of the subroutines.

Read. The Read function causes a specified amount of data to be read from an input device and placed in a specified input area. Depending upon the device, an interrupt signals the subroutine either when the next character is ready or when all requested data has been read. When the specified number of characters has been read, the subroutine becomes available for another call to that device.

Write. The Write function causes a specified amount of data from the user's output area to be written (or punched) on an output device. As with the Read function, an interrupt signals the subroutine when the device can accept another character, or when all characters have been written. When the specified number of characters has been written, the subroutine becomes available for another call to that device.

Test. The Test function causes a check to be made as to the status of a previous operation in that subroutine. If the previous operation has been completed the subroutine branches to the CALL +3 core location; if the previous operation has not been completed, the subroutine branches to the CALL +2 core location. The Test function is illustrated below:

	CALL	Name
CALL + 1	DC	Control Parameter (specifying Test function)
CALL + 2	OP Code	xxxx....
CALL + 3	OP Code	xxxx....

18074

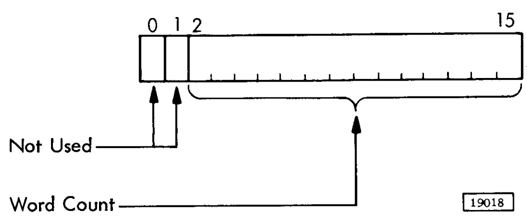
NOTE: Specifying the Test function always requires two statements (one CALL and one DC).

This function is useful in those situations where input data has been requested, and no processing can be done until that data is available.

I/O Area Parameter

The I/O area for a particular operation consists of one or more tables of control information and data. Each table is composed of a data area preceded by a control word (two control words for disk operations) which specifies how much data is to be transferred. The area parameter in the calling sequence is the address (symbolic or actual) of the control word(s) which precedes the data area.

The format of the control word used for all subroutines is shown below. The disk subroutine requires a second control word which is described along with that subroutine.



The word-count refers to the number of data words in the table. It is important to remember that the number of words in the table is not always the number of characters to be read or written because some codes pack several characters per word.

Error Parameter

The error parameter is the means by which an I/O subroutine can temporarily give control to the user in the event of certain error conditions. The parameter specifies the address to which the I/O subroutine will branch. The instruction sequence for setting up the error routine is shown below:

```
         CALL     NAME
          .        .
         DC       ERROR (error parameter)
ERROR    BSS      1       (return link)
          :        :      (error routine)
         BSC I    ERROR (branch to return link)
```

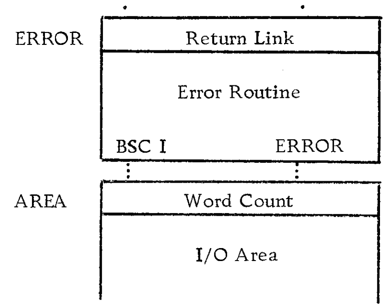
The return link is the address in the related I/O subroutine to which control must be returned upon completion of the error routine. The link will be inserted in location ERROR by a BSI instruction in the I/O subroutine when the subroutine branches to the error routine.

CARD SUBROUTINE

The card subroutine performs all I/O functions relative to the IBM 1442 Card Read Punch, namely, read, punch, feed, and select stacker.

Calling Sequence

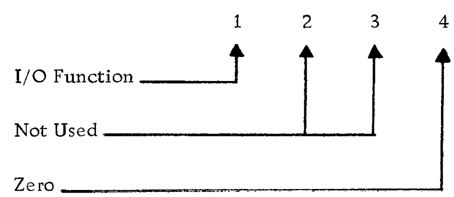
```
      CALL     CARD1
      DC       $xxx    (Control)
      DC       AREA    (I/O area)
      DC       ERROR (Error routine)
```



The calling sequence parameters are described in the following paragraphs.

Control Parameter

This parameter consists of four hexadecimal digits which are used as shown below:



I/O Function. The I/O function digit specifies a particular operation to be performed on a 1442 Card Read Punch. The allowable digits and the functions they represent are listed below and then described in detail.

Digit	Function
0	Test
1	Read
2	Punch
3	Feed
4	Select Stacker

- Test - Branches to CALL +2 if the previous operation has not been completed or to CALL +3 if the previous operation has been completed.

- Read - Reads one card and transfers a specified number of columns of data to the user's input area. The number of columns to be read (1-80) is specified by the user in the core location immediately preceding the input area. After initiating the card operation, the subroutine immediately clears the I/O area and stores a 1 in bit position 15 of each word in the I/O area, and returns control to the user's program. When each column is ready to be read, a column interrupt occurs. This permits the card subroutine to read the data from that column into the users input area (clearing bit 15), after which the user's program is again resumed. This sequence of events is repeated until the requested number of columns has been read. The data in the user's input area will be in card code format; that is, each 12-bit column image will be left-justified in one 16-bit word.

- Punch - Punches into one card the number of columns of data specified by the word-count found at the beginning of the user's output area. The punch operation is similar to the read operation. As each column comes under the punch dies, a column interrupt occurs; the card subroutine transfers a word from the user's output area to the punch, and then returns control to the user's program. This sequence is repeated until the requested number of columns has been punched. The character punched is the image of the leftmost 12 bits in the word.

- Feed - Initiates a card feed cycle. This advances all cards in the machine to the next station; i.e., a card at the punch station advances to the stacker; a card at the read station advances to the punch station; and a card in the hopper advances to the read station. No data is read or punched as a result of a feed operation and no column interrupts occur.

- Select Stacker - Selects stacker 2 for the card which is currently at the punch station. After the card passes the punch station it is directed to stacker 2.

Each card function described above requires a particular configuration of parameters.

Function	Parameters Required
Test	Control
Read	Control, I/O area, Error
Punch	Control, I/O area, Error
Feed	Control, Error
Select Stacker	Control

Any parameter not required for a particular function must be omitted.

I/O Area Parameter

The I/O area parameter is the label of the control word which precedes the user's I/O area. The control word consists of a word-count, which specifies the number of columns of data to be read or punched, always starting with column 1.

Error Parameter

The error parameter is the label of an error routine to be branched to in the event of certain errors (or last card condition). The types of errors that cause a branch to this routine are listed in Appendix A.

DISK SUBROUTINE

The purpose of the disk subroutine is to perform all reading and writing of data relative to Disk Storage. This includes the major functions: seek, read, and write, in conjunction with bit-count check and file-protection capabilities.

Sector Numbering and File Protection

In the interest of providing disk manipulation features which would permit versatile and orderly control of disk operations, two important conventions have been adopted. They are concerned with a sector-numbering scheme and a file-protection mechanism, and successful use of the disk subroutine can be expected only if user programs are built within the framework of these conventions.

The primary concern behind the conventions has been the safety of data recorded on the disk. Toward this end, the file-protection scheme plays a major role, but does so in a manner that is dependent upon the sector-numbering technique. The latter contributes to data safety by allowing the disk subroutine to verify the correct positioning of the access arm before it actually performs any writing operation. This requires that sector identifications be pre-recorded on each sector and that subsequent writing to the disk be done in a manner that preserves the existing identification. The disk subroutine has been organized to comply with this requirement.

The details of the numbering scheme are as follows: each disk sector is assigned an address from the sequence 0, 1, ..., 1599 corresponding to the sector's position in the ascending sequence of cylinder and sector numbers from logical cylinder 0 (innermost) sector 0, through logical cylinder 199 (outermost) sector 7. (The disk subroutine can address 200 cylinders, each cylinder containing eight sectors, each sector containing 320 words.)

The sector address is recorded by the user in the sector's first word, and occupies the rightmost eleven bit positions. Of these eleven positions, the three low-order positions identify the sector number (0-7) within the cylinder. Utilization of this first word for identification purposes diminishes the per sector availability of data words to 319; therefore, transmission of full sectors of data is performed in units of this amount.

File-protection is provided to guard against the inadvertent destruction of previously recorded data. By having the normal writing function, Write, uniformly test for the file-protection status of sectors it is about to write, this control can to a large degree be achieved. Implementation takes the form of having each sector carry in the sign position of its sector identification word the file-protect status for that sector. The file-protect status can be set or modified by the Write with File-Protect option function.

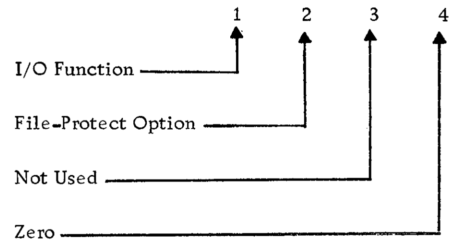
Calling Sequence

```
         CALL      DISK1
         DC        $xxxx      (Control)
         DC        AREA       (I/O area)
         DC        ERROR      (Error routine)
          :          :
ERROR   | Return Link       |
        | Error Routine     |
        | BSC 1       ERROR |
          :          :
AREA    | Word Count        |
        | Sector Address    |
        |   I/O Area        |
```

The calling sequence parameters are described in the following paragraphs.

Control Parameter

This parameter consists of four hexadecimal digits which are used as shown:



I/O Function. The I/O function digit specifies a particular operation to be performed on Disk Storage. The allowable digits and the functions they represent are listed below and then described in detail.

Digit	Function
0	Test
1	Read
2	Write
3	Write with file-protect option
4	Write Immediate
5	Seek

- Test - Branches to CALL +2 if the previous operation has not been completed or to CALL +3 if the previous operation has been completed.

- Read - Positions the access arm and reads data into the user's I/O area until the specified number of words has been transmitted. Although sector identification words are read and checked for agreement with expected values, they are neither transmitted to the I/O data area nor are they counted in the tally of words conveyed. The file-protect status of the initial sector read is placed in the sign position of the sector address word. (See I/O Area Parameter.) In the next bit to the right, the subroutine places a 0 if all sectors read had the same file-protect status, or a 1 if a mixture of file-protect status was encountered.

  If, during the reading of a sector, a read check occurs, up to ten retries are attempted. If the error persists after this time, the function is temporarily discontinued, an error code is placed in the accumulator, the address of the faulty sector is placed in the Q-Register, and an exit is made to the error routine specified by the ERROR parameter.

Upon return from this routine, the function is either reinitiated or terminated depending on whether the accumulator is non-zero or zero, respectively.

- Write - Positions the access arm, reads the sector identification word(s) of the designated (and a sufficient number of subsequent) sector(s), ascertains the file-protect status of each, and writes the contents of the indicated I/O data area into consecutive disk sectors. Writing begins at the designated sector and continues until the specified number of words has been transmitted, provided that no sector was found to have been file-protected. If no sectors were found to be file-protected, the subroutine executes a bit-count check of the sectors written.

  If any errors are detected, the operation is retried up to ten times. If the function cannot be accomplished by this time, an appropriate error code is placed in the A-Register, the address of the faulty sector is placed in the Q-Register, and exit is made to the error routine designated by the ERROR parameter. Upon return from this error routine, the function is either reinitiated or terminated depending upon whether the A-Register is non-zero or zero respectively.

  If a sector was found to have been file-protected, the subroutine discontinues the function, places an appropriate error code in the accumulator, places the address of the file-protected sector in the Q-Register, and exits to the error routine designated by the ERROR parameter. Upon return from this error routine, the disk subroutine either reinitiates or terminates the function depending on whether the accumulator is non-zero or zero, respectively.

NOTE: As each sector is written, the subroutine supplies the sector identification word. This word is neither obtained from the I/O area nor is it counted in the tally of words conveyed.

- Write with File-Protect Option - Positions the access arm and reads the sector identification word from each sector that is to be written in order to verify the proper positioning of the arm. Then, without regard for the file-protect status of the sectors that are encountered, the subroutine writes the contents of the indicated I/O data area into consecutive disk sectors, beginning at the designated sector, until the specified number of words has been transmitted. As each sector is written, the subroutine places the requested file-protect status in the sign position of the corresponding sector identification word. (This word is supplied by the subroutine, but is not counted in the tally of the words transmitted.)

  If any errors occur, the subroutine attempts up to ten retries of the function for the sector in which the error occurred. If the function cannot be accomplished by this time, an appropriate error code is placed in the A-Register, the address of the faulty sector is placed in the Q-Register, and an exit to the error routine designated by the ERROR parameter is effected. Upon return from this routine the subroutine either reinitiates or terminates the function depending upon whether the A-Register is non-zero or zero, respectively.

- Write Immediate - Writes data with <u>no</u> attempt to position the access arm, check for file-protect status or check for errors. Writing begins at the sector number specified by the rightmost three bits of the sector address. This function is provided to fulfill the need for more rapid writing to the disk than is provided in the previously described Write functions.

- Seek - Moves the indicated device's access arm to the cylinder bearing the sector address designated in the disk I/O area control word.

Each disk function described above requires a particular configuration of parameters.

Function	Parameters Required
Test	Control
Read	Control, I/O area, Error
Write	Control, I/O area, Error
Write with File-Protect option	Control, I/O area, Error
Write immediate	Control, I/O area
Seek	Control, I/O area, Error

Any parameter not required for a particular function must be omitted.

<u>File-Protect option.</u> This digit specifies the file-protect status that is to be imparted to sectors written by means of the Write with File-Protect Option function. The digit must be a 0 for no file protection or a 1 for file protection. For any other function this digit has no meaning and is therefore ignored.

I/O Area Parameter

The I/O area parameter is the label of the first of two control words which precede the user's I/O area. The first word contains a count of the number of words that are to be transmitted during the disk operation. This count need not be limited by sector or cylinder size, since the disk subroutine crosses sector and cylinder boundaries, if necessary, in order to process the specified number of words. The second word contains the sector address where reading or writing is to begin. After a read operation the two high-order positions of the sector address word will contain file-protect information. (See description of Read operation.) Following the two control words is the user's data area.

Error Parameter

The error parameter is the label of an error routine to be branched to in the event of certain errors. The types of errors that cause a branch to this routine are listed in Appendix A.

PRINTER SUBROUTINE

The printer subroutine handles all print and carriage control functions relative to the IBM 1132 Printer. Only one line of data can be printed with one call to a printer subroutine. The data in the output area must be in EBCDIC form, packed two characters per computer word. (See Data Codes.) Besides its print function, the printer subroutine performs spacing and skipping operations.

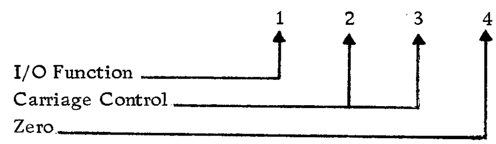
Calling Sequence

```
         CALL      PRNT1
         DC        $xxxx       (Control)
         DC        AREA        (I/O area)
         DC        ERROR       (Error routine
          .          .
          .          .
ERROR   | Return Link       |
        |                   |
        |  Error Routine    |
        |                   |
        | BSC I      ERROR  |
          .          .
          .          .
AREA    | Word Count        |
        |                   |
        |  I/O Area         |
```

The calling sequence parameters are described in the following paragraphs.

Control Parameter

This parameter consists of four hexadecimal digits which are used as shown below.



I/O Function. The I/O function digit specifies a particular operation to be performed on a 1132 printer. The allowable digits and the functions they represent are listed below and then described in detail.

Digit	Function
0	Test
1	Print/no checks
2	Print/with checks
3	Control Carriage

- Test - Branches to CALL +2 if the previous operation has not been completed or to CALL +3 if the previous operation has been completed.
- Print/no checks - Prints characters from the user's I/O area, ignoring channel 9 and 12 indications.
- Print/with checks - Prints characters from the user's I/O area, checking for channel 9 and 12 indications. If either of these conditions is detected, the subroutine branches to the user's error routine. This branch occurs after the line of data has been printed.

Carriage Control. Digits 2 and 3 specify the carriage control functions listed in Table 2. An "immediate" request is executed before the next print operation; an "after print" request is executed after the next print operation, and replaces the normal space operation.

Each print function described above requires a particular configuration of parameters.

Function	Parameters Required
Test	Control
Print/no checks	Control, I/O area
Print/with checks	Control, I/O area, Error
Control Carriage	Control

Any parameter not required for a particular function must be omitted.

I/O Area Parameter

The I/O area parameter is the label of the control word which precedes the user's I/O area. The control word consists of a word-count which specifies the number of computer words of data to be printed. The data must be in EBCDIC format, packed two characters per computer word.

Error Parameter

The error parameter is the label of an error routine to be branched to in the event of certain conditions. The types of conditions that cause a branch to this routine are listed in Appendix A.

Table 2. Carriage Control Operations

Character #2: Immediate Carriage Operations
Print Functions Not Used
Control Function 1 - Immediate Skip To Channel 1 2 - Immediate Skip To Channel 2 3 - Immediate Skip To Channel 3 4 - Immediate Skip To Channel 4 5 - Immediate Skip To Channel 5 6 - Immediate Skip To Channel 6 9 - Immediate Skip To Channel 9 C - Immediate Skip To Channel 12 D - Immediate Space Of 1 E - Immediate Space Of 2 F - Immediate Space Of 3
Character #3: After-Print Carriage Operations
Print Functions 0 - Space One Line After Printing 1 - Suppress Space After Printing
Control Function 1 - Skip After Print To Channel 1 2 - Skip After Print To Channel 2 3 - Skip After Print To Channel 3 4 - Skip After Print To Channel 4 5 - Skip After Print To Channel 5 6 - Skip After Print To Channel 6 9 - Skip After Print To Channel 9 C - Skip After Print To Channel 12 D - Space 1 After Print E - Space 2 After Print F - Space 3 After Print

19025

KEYBOARD—CONSOLE PRINTER SUBROUTINES

There are two I/O subroutines for the transfer of data to and from the Console Printer and the Console Keyboard. The subroutine TYPE1 handles input and output; WRTY1 handles output only. If a particular program does not require keyboard input, it is advantageous to use the WRTY1 subroutine because it occupies less core storage than the TYPE1 subroutine.

Only the TYPE1 subroutine is described; the WRTY1 subroutine is identical except that it does not contain the Read-Print function.

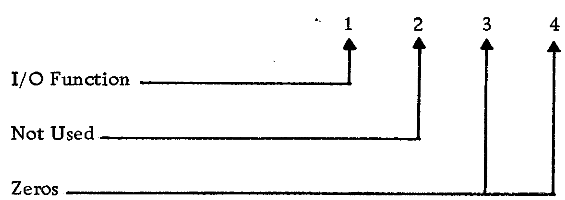
<u>Calling Sequence</u>

```
        CALL        TYPE1 or WRTY1
        DC          $xxxx    (Control)
        DC          AREA     (I/O area)
         .           .
AREA   | Word Count |
       |  I/O Area  |
```

The parameters used in the above calling sequence are described in the following paragraphs.

Control Parameter

This parameter consists of four hexadecimal digits which are used as shown below:



- Test - Branches to CALL +2 if the previous operation has not been completed or to CALL +3 if the previous operation has been completed.

- Read-Print - Reads from the keyboard and prints on the console printer the requested number of characters. The operation sequence is as follows:
  1. The calling sequence is analyzed by the call routine which then unlocks the keyboard.
  2. When a key is pressed, a character interrupt signals the interrupt response routine that a character is ready to be read into core storage.
  3. The interrupt response routine converts the keyboard data to the Typewriter code (see Data

Codes), printing each character on the console printer as the character is read and unlocking the keyboard for entry of the next character, if any.

4. Printer interrupts occur whenever the console printer has completed its print operation. When the interrupt has been received, the routine checks whether the final character has been read and printed. If so, the operation is considered complete.
5. Items 2 to 4 are repeated until the specified number of characters has been read and printed. The characters read into the I/O area are in card code format; that is, each 12-bit image is left-justified in one 16-bit word.

Three control characters are recognized by the typewriter subroutine:

Backspace. The operator presses the backspace key whenever the previous character is in error. The interrupt response routine, sensing the control character, backspaces the console printer and prints a slash (/) through the previous character. In addition, the subroutine prepares to replace the previous character in the I/O area with the next character.

Re-entry. When the interrupt response routine recognizes the re-entry control character, it assumes that the entire message is in error and is to be re-entered. The routine prints two slashes on the console printer and restores the carrier to a new line. In addition, the routine prepares to replace the old message in the I/O area with the new message.

End-of-Message. When the interrupt response routine recognizes the end-of-message control character, it assumes the message has been completed, stores the character in the I/O area, and terminates the operation.

- Print - Print the specified number of characters on the console printer. Printer interrupts occur whenever the console printer has completed a print operation. When the interrupt has been received, the character count is checked. If the specified number of characters has not been written, printing is initiated for the next character. This sequence continues until the specified number of characters has been printed. Data to be printed must be in Typewriter code, packed two characters per 16-bit word.

Each typewriter function described above requires a particular configuration of parameters.

Function	Parameters Required
Test	Control
Read-Print	Control, I/O area
Print	Control, I/O area

Any parameter not required for a particular function must be omitted.

I/O Area Parameter

The I/O area parameter is the label of the control word which precedes the user's I/O area. The control word consists of a word-count which specifies the number of words to be read into or printed from. This word-count is equal to the number of characters if the Read-Print function is requested but not if the Print function is requested.

Operator Request Function

By pressing the Operator Request key on the keyboard, the operator can inform the program that he wants to enter some data from the keyboard. The interrupt that results from such a request causes the typewriter routine to execute an indirect BSI instruction to core location 44. The user must have in core location 44 the address of an operator request routine.

PAPER TAPE SUBROUTINE

The paper tape subroutine handles the transfer of data from IBM 1054 Paper Tape Reader to core storage and from core storage to the IBM 1055 Paper Tape Punch. Any number of characters may be transferred via one calling sequence. If desired, both the reader and the punch may be operated simultaneously.

When called, the paper tape subroutine starts the reader or punch and then, as interrupts occur, transfers data to or from the user's I/O area. The data is packed two characters per computer word by the subroutine when reading, and must be in that form when the subroutine is called for a punch function.

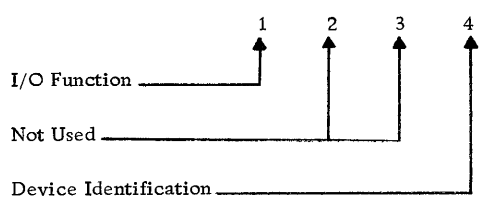
Calling Sequence

```
         CALL    PAPT
         DC      $xxxx     (Control)
         DC      AREA      (I/O area)
         DC      $xxxx     (Character)
          :        :
AREA    | Word Count |
        |            |
        |  I/O Area  |
```

The parameters used in the above calling sequence are described in the following paragraphs.

Control Parameter

This parameter consists of four hexadecimal digits which are used as shown below:



I/O Function. The I/O function digit specifies a particular operation to be performed on a 1054/1055 Paper Tape attachment. The allowable digits and the functions they represent are listed below and then described in detail.

Digit	Function
0	Test
1	Read/No Check
2	Punch
3	Read/Check

- Test - Branches to CALL +2 if previous operation has not been completed or to CALL +3 if the previous operation has been completed.

- Read/No Check - Reads paper tape characters into the specified number of words in the I/O area. Initiating reader motion causes an interrupt to occur when a character can be read into core. If the specified number of words has not been filled, reader motion is again initiated. If the specified number has been filled, the reader is halted. No check is made to determine if any of the input characters are Stop or Delete characters. (See Character Parameter.) An even number of characters is always read.

- Punch - Punches paper tape characters into the tape from the words in the I/O area. Each character punched causes an interrupt which indicates that the next character can be accepted. When the specified number of words in the I/O area has been punched the operation is terminated. An even number of characters is always punched.

- Read/Check - Reads paper tape characters into the I/O area, checking each character to see if it is a Delete or Stop character. When the specified number of words has been filled (two characters per word) the operation is terminated. A Delete character is not placed in the I/O area and therefore does not enter into the count of the total number of words to be filled. A Stop character is transferred into the I/O area and causes the operation to be terminated even if the specified number of words has not been filled.

Each Paper Tape function described above requires a particular configuration of parameters.

Function	Parameters Required
Test	Control
Read/No Check	Control, I/O area
Punch	Control, I/O area
Read/Check	Control, I/O area, Character

Any parameter not required for a particular function must be omitted.

Device Identification. When the Test function is specified, the subroutine must be told which device (reader or punch) is to be tested for an "operation complete" indication. (Remember that both the reader and the punch can operate simultaneously.) If the device identification is a 0, the subroutine tests for a "reader complete" indication; if it is a 1, the subroutine tests for a "punch complete" indication.

I/O Area Parameter

The I/O area parameter is the label of the control word which precedes the user's I/O area. The control word consists of a word-count which specifies the number of words to be read into or punched from. Since characters are packed two per word in the I/O area, this count is one-half the number of characters transferred. Because an entire 8-bit channel image is transferred by the subroutine, any combination of channel punches is acceptable. The data may be a binary value or a character code. The code most often used is the Paper Tape BCD code. (See Data Codes.)

Character Parameter

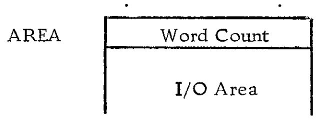
This parameter is used to specify which 8-bit images are to be recognized as Stop and Delete characters. Bits 0-7 specify the Stop character; bits 8-15 specify the Delete character. If either character is all zeros, no checking for that character is done by the Read/Check function.

PLOTTER SUBROUTINE

The plotter subroutine converts hexadecimal digits found in the user's output area into actuating signals which control the movement of the 1627 recording pen. Each hexadecimal digit in the output area is translated into plotter operations: either the drawing of a line segment, or the raising or lowering of the recording pen. The amount of data that can be recorded with one calling sequence is limited only by the size of the corresponding output area.

Calling Sequence

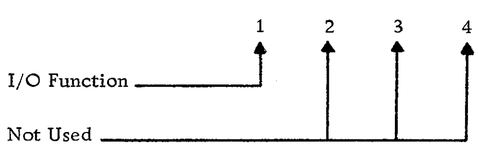
```
        CALL    PLOT
        DC      $xxxx      (Control)
        DC      AREA       (I/O area)
         :        :
```



The calling sequence parameters are described in the following paragraphs.

Control Parameter

This parameter consists of four hexadecimal digits which are used as shown below:



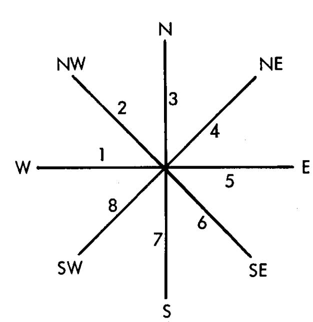
I/O Function. The I/O function digit specifies a particular operation to be performed on the 1627 Plotter. The allowable digits and the functions they represent are listed below and then described in detail.

Digit	Function
0	Test
1	Write

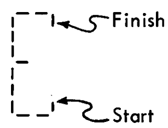
- Test - Branches to CALL +2 if the previous operation has not been completed or to CALL +3 if the previous operation has been completed.
- Write - Transforms hexadecimal digits found in the output area into signals which actuate the plotter. Table 3 lists the hexadecimal digits and the plotting actions they represent. Figure 3 shows the binary and hexadecimal configurations for drawing the letter E.

Table 3. Plotter Control Digits

Hexadecimal Digit	Plotter Action (See Diagram Below)
0	Pen Down
1	Line Segment - West
2	Line Segment - Northwest
3	Line Segment - North
4	Line Segment - Northeast
5	Line Segment - East
6	Line Segment - Southeast
7	Line Segment - South
8	Line Segment - Southwest
9	Pen Up
A	Repeat the previous digit the number of times specified by the next digit (Maximum - 15 times)
B	Repeat the previous digit the number of times specified by the next two digits (Maximum-255 times)
C	Repeat the previous digit the number of times specified by the next three digits (Maximum-4095 times)
D	Not Used
E	Not Used
F	Not Used



18071

Binary	Hexadecimal	Figure
0000011100010001	0711	
0011101000100101	3A25	
1001000100000011	9103	
1010001001010101	A255	
0111100111111111	79FF	

18056

Figure 3. Plotter Example

Each plot function described above requires a particular configuration of parameters.

Function	Parameters Required
Test	Control
Write	Control, I/O area

Any parameter not required for a particular function must be omitted.

I/O Area Parameter

The I/O area parameter is the label of the control wo which precedes the user's I/O area. The control wor consists of a word-count which specifies the number of computer words of data to be used.

EDIT PROGRAM

In its form as supplied by IBM, the I/O subroutine deck is not initialized for direct use by a customer installation, for missing from the deck are the particulars which define and differentiate the user's system from others of its kind. The IBM 1130 EDIT program accepts as input a statement of the system configuration (including assigned interrupt levels, area codes, and device numbers), and the subroutine deck provided by IBM. Data from the system configuration cards is integrated with the master deck and a systems deck is produced incorporating this data. Subroutines of the master deck which are not applicable to the current system are not punched into the systems deck, and are, in effect, edited out. The newly produced systems deck has all of the subroutines necessary to handle the input-output equipment attached, along with the necessary interrupt processing routines.

The basic unit of information within the 1130 System is the 16-bit binary word. This information may be interpreted in a variety of ways, depending on the circumstances. For example, in purely internal computer operations, computer words may be interpreted as instructions, as addresses, as binary integers, or as floating-point numbers (see Arithmetic and Functional Subroutines).

This section is concerned with interpretations of the bit configurations which relate computer information with the outside world. These interpretations are made necessary by the following considerations:

1. A compact notation is needed by the programmer to externally represent the bit configuration within each computer word. This is provided in the "hexadecimal" notation.
2. A code is required for representing alphameric (mixed alphabetic and numeric) data within the computer. This is provided by the Extended Binary Coded Decimal Interchange Code (EBCDIC).
3. The design and operation of the various input/output devices is such that many of them impose a unique correspondence between character representations in the external medium and the associated bit configurations within the computer. Conversion subroutines are needed to convert inputs from these devices into a form on which the computer can operate, and to prepare computed results for output on the devices.

This section of the manual describes the subroutines for converting data representations between these various codes.

DATA CODES

In addition to the 16-bit binary internal representation, the conversion subroutines handle the following five codes:

1. Hexadecimal Notation
2. Extended Binary Coded Decimal Interchange Code (EBCDIC)
3. IBM Card Code
4. Paper Tape Code
5. Output Typewriter Code

A list of these codes can be found in Appendix B.

Hexadecimal Notation

Although binary numbers facilitate the operations of computers, they are bulky and awkward to handle by the programmer. A long string of 1's and 0's cannot be effectively transmitted from one individual to another. The hexadecimal number system is often used as a shorthand method of communicating binary numbers. Because of the simple relationship of hexadecimal to binary, numbers can easily be converted from one system to another.

In hexadecimal notation a single digit is used to represent a four-bit binary value as shown in Figure 4. Thus, a 16-bit word in the 1130 System can be expressed as four hexadecimal digits. For example, the binary value

1101001110111011

can be separated into four sections as follows:

Binary	1101/	0011/	1011/	1011
Hexadecimal	D	3	B	B

Another advantage of hexadecimal notation is that fewer positions are required when output data is printed, or punched in cards or paper tape. In the example above, only four card columns would be required to contain the data from a 16-bit binary word.

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

18057

Figure 4. Hexadecimal Notation

Extended Binary Coded Decimal Interchange Code (EBCDIC)

EBCDIC is the standard code for internal representation of alphameric and special characters. The code occupies eight binary bits per character, making it possible to store either one or two characters per computer word. The eight bits allow 256 different possible codes. (At present, not all of these combinations have been assigned to represent characters.) The complete EBCDIC code is shown in Appendix B. The user should note that the codes for paper tape, and typewriter are given in hexadecimal notation.

To make the conversion subroutines more efficient, most of them will not recognize all 256 codes. The asterisked codes in Appendix B constitute the subset which is recognized by most of the conversion subroutines.

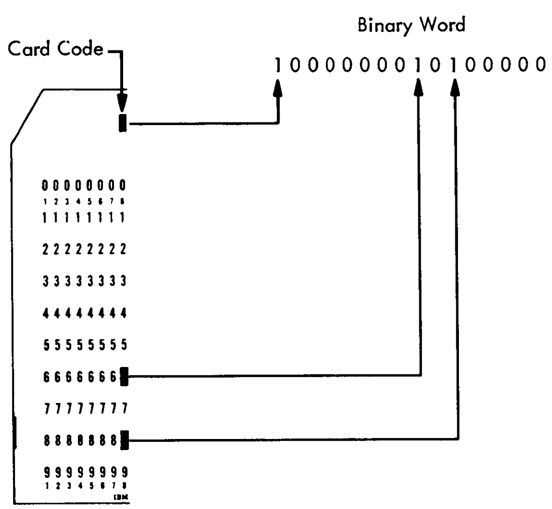
IBM Card Code

The IBM Card Code is used by the 1442 Card Read Punch and in the input from the Console Keyboard.

This code defines a character by a combination of punches in a card column. Card-code data is taken from, or placed into, the leftmost twelve bits of a computer word as shown below:

Card Row	12	11	0	1	2	3	4	5	6	7	8	9	-	-	-	-
Computer Word	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

For example, a plus sign with a card code of 12, 6, 8 punches is placed into core storage in the binary configuration illustrated in the following diagram.



18103

Paper Tape Code (8-bit)

The paper tape code is a 6-bit subset of the Extended BCD Interchange code. It may be used with the 1054/1055 Paper Tape attachment. This code represents a character with a stop position, a check position, and six positions representing the 6-bit code BA8421. Paper tape characters can be packed two per computer word as shown below.

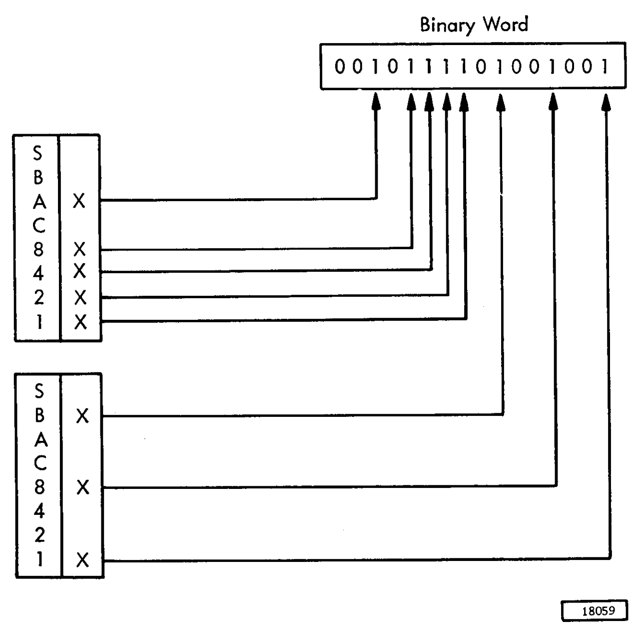
	1st								2nd							
Paper Tape Characters	S	B	A	C	8	4	2	1	S	B	A	C	8	4	2	1
Computer Word	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

18105

The binary configuration of paper tape code for the characters ? R is shown in Figure 5.

Typewriter Code

The Typewriter code is the 8-bit Console Printer code. Typewriter characters can be packed two per computer word.



18059

Figure 5. 8-Bit Paper Tape Code for ? R

DESCRIPTIONS

Eleven data conversion subroutines are provided.

BINDC Binary value to IBM card-coded decimal value.

DCBIN IBM card-coded decimal value to binary value.

BINHX Binary value to IBM card-coded hexadecimal value.

HXBIN IBM card-coded hexadecimal value to a binary value.

HOLEB IBM card code characters to EBCDIC subset; EBCDIC subset to IBM card code characters.

SPEED IBM card code characters to EBCDIC (256 character code).

PAPEB Paper tape code to EBCDIC subset; EBCDIC subset to paper tape code.

PAPHL Paper tape code to IBM card code characters: IBM card code characters to paper tape code.

PAPPR Paper tape code to typewriter code.

HOLPR IBM card code to typewriter code.

EBPRT EBCDIC subset to typewriter code.

NOTE: In addition to the subroutines listed above, there are three conversion tables used by some of the conversion subroutines.

PRTY—Typewriter code.

EBPA—EBCDIC and paper tape codes.

HOLL—Card code.

The first four of these subroutines change numeric data from its input form to a binary form, or from a binary form to an appropriate output data code. The last seven convert entire messages, one character at a time, from one input/output code to another. The different types of conversions offered by these subroutines are illustrated in Figure 6.

Error checking

All of the subroutines except SPEED will accept only the codes asterisked in Appendix B. It is considered an error if any input character does not belong to the specified input code or cannot be converted to the specified output code. A space character, in the output code, is stored in the output area for the input character in error.

Converted From	Converted To						
	Binary	IBM Card Code – Hexadecimal	IBM Card Code	Paper Tape	Printer EBCDIC Subset (80 Char)	Printer EBCDIC (256 Char)	Typewriter
Binary		BINHX	BINDC				
IBM Card Code – Hexadecimal	HXBIN						
IBM Card Code	DCBIN			PAPHL	HOLEB	SPEED	HOLPR
Paper Tape			PAPHL		PAPEB		PAPPR
EBCDIC (Subset)			HOLEB	PAPEB			EBPRT

19021

Figure 6. Types of Conversions

If any such error occurs, the Carry indicator is turned off and the Overflow indicator is turned on when the conversion subroutine returns control to the user. Otherwise, the settings of the Carry and Overflow indicators are not altered by the subroutine.

BINDC

This subroutine converts a 16-bit binary value to its decimal equivalent in five IBM card-coded characters and one sign character. The five characters and the sign are placed into six computer words as illustrated in Figure 7.

Calling Sequence

```
                CALL        BINDC
                DC          OUTPT
                 :            :
OUTPT           BSS           6
```

Input

Input is a 16-bit binary value in the A-Register.

Output

Output is a 12-bit IBM card-coded sign character (plus or minus) in location OUTPT, and five 12-bit IBM card-coded numerical characters in OUTPT +1 through OUTPT +5.

DCBIN

This subroutine converts a decimal value in five IBM card-coded characters and a sign character to a 16-bit binary word. The conversion is the reverse of the BINDC subroutine conversion illustrated in Figure 7.

Value of +1538

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0

Card Code

OUTPT — Sign +

12	11	0	1	2	3	4	5	6	7	8	9				
1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0

OUTPT +1 — 0

12	11	0	1	2	3	4	5	6	7	8	9				
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

OUTPT +2 — 1

12	11	0	1	2	3	4	5	6	7	8	9				
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0

OUTPT +3 — 5

12	11	.0	1	2	3	4	5	6	7	8	9				
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

OUTPT +4 — 3

12	11	0	1	2	3	4	5	6	7	8	9				
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

OUTPT +5 — 8

12	11	0	1	2	3	4	5	6	7	8	9				
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

18061

Figure 7. BINDC Conversion

Calling Sequence

```
            CALL     DCBIN
            DC       INPUT
             .         .
             .         .
INPUT       BSS        6
```

Input

Input is a 12-bit IBM card-coded sign character in location INPUT and five 12-bit IBM card-coded decimal characters in INPUT +1 through INPUT +5.

Output

Output is a 16-bit binary word in the A-Register, containing the converted value.

Error Conditions Detected

Any character other than an IBM card-coded plus, ampersand, space, or minus as the sign, or 0 through 9 as a decimal digit is considered an error. Any converted value greater than +32767 or less than -32768 is considered an error.

BINHX

This subroutine converts a 16-bit binary word into hexadecimal notation in four IBM card-coded characters as illustrated in Figure 8.

Calling Sequence

```
            CALL     BINHX
            DC       OUTPT
             .         .
             .         .
INPUT       BSS        4
```

Input

Input is a 16-bit binary word in the A-Register.

Output

Output is four 12-bit IBM card-coded hexadecimal digits in location OUTPT through OUTPT +3.

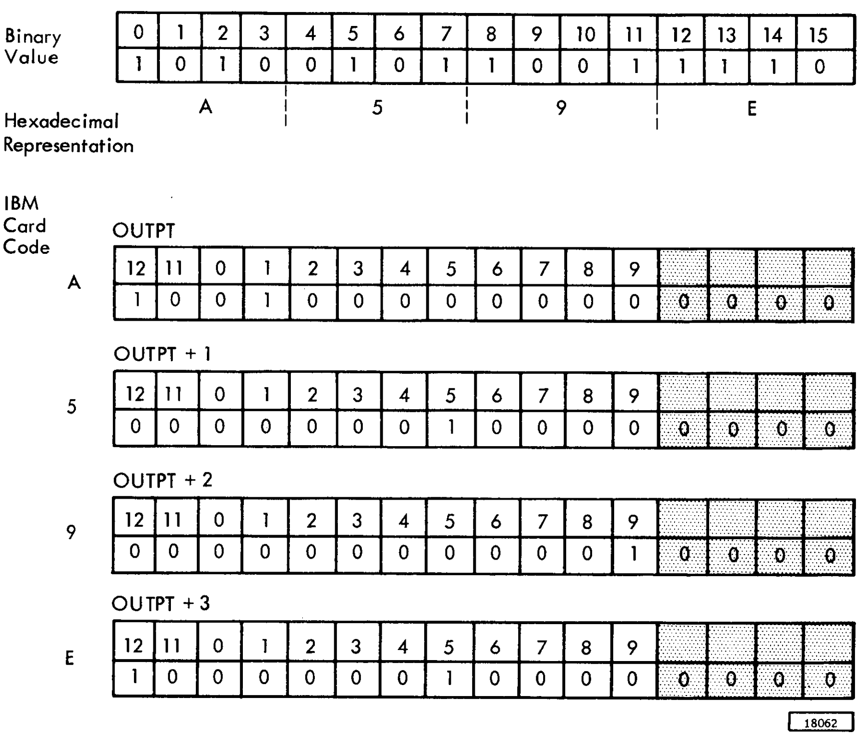


Figure 8. BINHX Conversion

HXBIN

This subroutine converts four 12-bit IBM card-coded hexadecimal characters into one 16-bit binary word. The conversion is the reverse of the BINHX subroutine conversion illustrated in Figure 8.

Calling Sequence

```
                CALL        HXBIN
                DC          INPUT
                 .            .
                 .            .
      INPUT     BSS           4
```

Input

Input is four 12-bit IBM card-coded hexadecimal digits in INPUT through INPUT +3.

Output

Output is a 16-bit binary word in the A-Register.

Error Conditions Detected

Any character other than an IBM card-coded 0 through 9 or A through F is considered an error.

HOLEB

This subroutine converts IBM card code to the EBCDIC subset or converts the EBCDIC subset to IBM card code. This code conversion is illustrated in Figure 9.

Only the character codes asterisked in Appendix B are available in this subroutine.

Calling Sequence

```
              CALL      HOLEB
              DC        $xxxx      (Control)
              DC        INPUT
              DC        OUTPT
              DC        nnnnn      (Character count)
               .          .
      INPUT   [-----------]
               .          .
      OUTPT   [-----------]
```

Control Parameter

Four hexadecimal digits. Digits 1-3 are not used. The fourth digit specifies the direction of conversion:
- 0 - IBM card code to EBCDIC
- 1 - EBCDIC to IBM card code.

Input

Input is either IBM card coded or EBCDIC characters, (as specified by the control parameter) starting in location INPUT. EBCDIC characters must be packed two characters per one binary word. IBM card coded characters are stored one character to each binary word.

Output

Output is either IBM card coded or EBCDIC characters starting in location OUTPT. Characters are

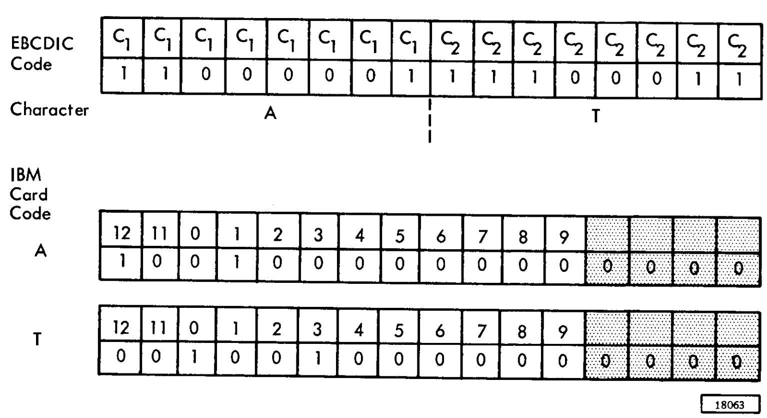


Figure 9. HOLEB Conversion

packed as described above.

If the direction of the conversion is IBM card code input to EBCDIC output, the input area may overlap the output area if the address INPUT is equal to or greater than the address OUTPT. If the direction of the conversion is EBCDIC input to IBM card code output, the input area may overlap the output area if the address INPUT + (n/2) - 1 is equal to or greater than the address OUTPT +n-1, where n is the character-count specified. The subroutine starts processing at location INPUT.

Character Count

This number specifies the number of characters to be converted; it is not equal to the number of binary words used for the EBCDIC characters because those characters are packed two per binary word. If an odd count is specified, bits 8 through 15 of the last word in the output area are not altered.

Error Conditions Detected

Any input character which has no equivalent among the IBM card code or EBCDIC characters asterisked in Appendix B is considered an error.

SPEED

This subroutine converts IBM card-coded characters to EBCDIC characters, accepting all 256 character codes as defined in Appendix B. The conversion time is much faster than the conversion time of the previous subroutine because the conversion can take place while the CARD subroutine is reading in a card and because a different conversion method is possible when all 256 EBCDIC characters are used.

If the SPEED subroutine is called before a card reading operation is completed, the SPEED subroutine synchronizes with the CARD subroutine by checking bit 15 of the word to be processed before converting the word.

If that bit is a one, the SPEED subroutine waits in a loop until the CARD subroutine sets the bit to a zero.

Calling Sequence

```
              CALL      SPEED
              DC        $xxxx     (Control)
              DC        INPUT
              DC        OUTPT
              DC        nnnnn     (Character count)
               .          .
               .          .
       INPUT  |-----------|
              |- - - - - -|
               .          .
               .          .
       OUTPT  |-----------|
              |- - - - - -|
```

Control Parameter

Four hexadecimal digits. Digits 1-3 are not used. The fourth digit indicates whether the output is to be packed one or two characters per binary word.

0 — Packed, two EBCDIC characters per binary word.

1 — Unpacked, one EBCDIC character per binary word (left-justified).

Input

Input is IBM card-coded characters starting in location INPUT.

Output

Output is EBCDIC characters starting in location OUTPT. Characters may be packed or unpacked.

The input area should not overlap the output area because of restart problems resulting from card feed errors.

Character Count

This number specifies the number of IBM card coded characters to be converted.

Error Conditions Detected

Any input character not defined among the IBM card code characters in Appendix B is considered an error.

Note that all IBM card code punch combinations, except multiple punches in rows 1-7 are legal.

PAPEB

This subroutine converts paper tape code to EBCDIC or converts EBCDIC to paper tape code. The relationship of codes for converting paper tape codes to EBCDIC is illustrated in Figure 10.

Calling Sequence

```
         CALL    PAPEB
         DC      $xxxx         (Control)
         DC      INPUT
         DC      OUTPT
         DC      nnnnn         (Character count)
          :        :
INPUT    [- - - - - - - - - - -]
          :        :
OUTPT    [- - - - - - - - - - -]
```

Control Parameter

Four hexadecimal digits. Digits 1-3 are not used. The fourth digit indicates the direction of conversion.

0 — Paper tape to EBCDIC
1 — EBCDIC to paper tape

Input

Input is either paper tape or EBCDIC characters, as specified by the control parameter, starting in location INPUT. Both character codes are packed two per computer word.

Output

Output is either EBCDIC or paper tape characters starting in OUTPT. Both character codes are in packed format. The address INPUT must be equal to or greater than the address OUTPT if overlap of the input and output areas is desired. The subroutine starts processing at location INPUT.

Character Count

This number specifies the number of paper tape or EBCDIC characters to be converted. (This count is not equal to the number of binary words used in the input area.) If an odd-count is specified, bits 8-15 of the last word in the output area are not altered.

Error Conditions Detected

Any input character which has no equivalent among the asterisked codes in Appendix B is considered an error.

PAPHL

This subroutine converts paper tape code to IBM card code, or IBM card code to paper tape. Figure 11 illustrates the relationship of the two codes for converting paper tape codes to IBM card code.

Calling Sequence

```
         CALL    PAPHL
         DC      $xxxx         (Control)
         DC      INPUT
         DC      OUTPT
         DC      nnnnn         (Character count)
          :        :
INPUT    [- - - - - - - - - - -]
          :        :
OUTPT    [- - - - - - - - - - -]
```

Paper Tape Code

S	B	A	C	8	4	2	1	S	B	A	C	8	4	2	1
0	1	1	0	0	0	0	1	0	0	1	0	0	0	1	1
			A								T				

EBCDIC Code

C_1	C_1	C_1	C_1	C_1	C_1	C_1	C_1	C_2	C_2	C_2	C_2	C_2	C_2	C_2	C_2
1	1	0	0	0	0	0	1	1	1	1	0	0	0	1	1
			A								T				

18064

Figure 10. PAPEB Conversion

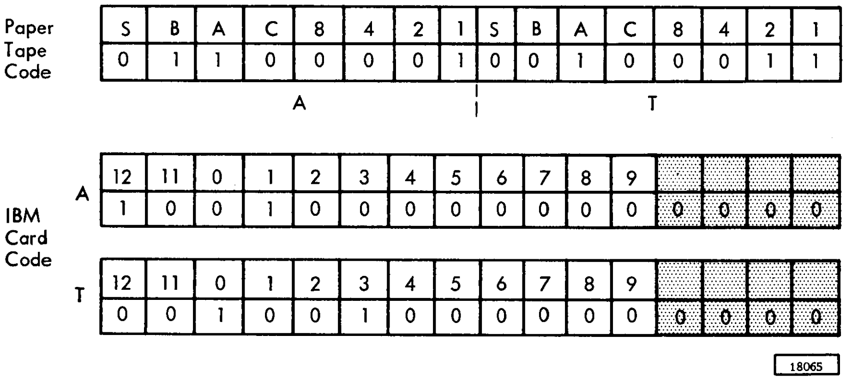


Figure 11. PAPHL Conversion

Control Parameter

Four hexadecimal digits. Digits 1-3 are not used. The fourth digit indicates the type of conversion.

0 — paper tape to IBM card code
1 — IBM card code to paper tape

Input

Input is either paper tape or IBM card code characters, as specified by the control parameter, starting in location INPUT. Paper Tape characters are packed two per binary word; IBM card code characters are not packed.

Output

Output is either IBM card code or paper tape code characters starting in location OUTPT. Paper tape codes are packed two per binary word; IBM card codes are not packed.

If the direction of the conversion is IBM card code input to paper tape output, the input area may overlap the output area if the address INPUT is equal to or greater than the address OUTPT. If the direction of the conversion is paper tape input to IBM card code output, the input area may overlap the output area if the address INPUT + (n/2) - 1 is equal to or greater than the address OUTPT + n - 1, where n is the character count specified in the parameter list. The subroutine starts processing at location INPUT.

Character Count

This number specifies the number of paper tape or IBM card code characters to be converted. (This count is not equal to the number of binary words for the paper tape characters because those characters are packed two per binary word. If an odd count is specified, bits 8-15 of the last word in the output area are not altered.)

Error Conditions Detected

Any input character which has no equivalent among the asterisked codes in Appendix B is considered an error.

PAPPR

This subroutine converts paper tape code to typewriter code. The conversion is illustrated in Figure 12.

Calling Sequence

```
          CALL    PAPPR
          DC      $xxxx      (Control)
          DC      INPUT
          DC      OUTPT
          DC      nnnnn      (Character count)
           :        :
INPUT    |-----------|
          :        :
OUTPT    |-----------|
```

Paper Tape Code

	S	B	A	C	8	4	2	1	S	B	A	C	8	4	2	1
	0	1	1	0	0	0	0	1	0	0	1	0	0	0	1	1
Character				A								T				

	S	B	A	C	8	4	2	1								
	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0
Character				9												

Typewriter Code

	C_1	C_1	C_1	C_1	C_1	C_1	C_1	C_1	C_2	C_2	C_2	C_2	C_2	C_2	C_2	C_2
	0	0	1	1	1	1	0	0	1	0	0	1	1	1	0	0
Character				A								T				

	C_1	C_1	C_1	C_1	C_1	C_1	C_1	C_1								
	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
					9											

19022

Figure 12. PAPPR Conversion

Control Parameter

Four hexadecimal digits. Digits 1-3 are not used. The fourth digit must be a zero.

Input

Input is paper tape-coded characters starting in location INPUT. Paper tape-coded characters are packed two per binary word.

Output

Output is typewriter coded characters starting in location OUTPT. The characters are packed two per binary word.

The address INPUT must be equal to or greater than the address OUTPT if overlap of the input and output areas is desired. The subroutine starts processing at location INPUT.

Character Count

This number represents the number of paper tape coded characters to be converted. If an odd count is specified, bits 8-15 of the last word in the output area are not altered.

Error Conditions Detected

Any input character which has no equivalent among the asterisked characters listed in Appendix B is considered an error.

HOLPR

This subroutine converts IBM card-coded characters to typewriter-coded characters. The relationship of the coded characters for the conversion typewriter coded characters is illustrated in Figure 13.

Calling Sequence

```
        CALL   HOLPR
        DC     $xxxx        (Control)
        DC     INPUT
        DC     OUTPT
        DC     nnnnn        (Character count)
         :       :
INPUT   |- - - - - - - - - - -|
         :       :
OUTPT   |- - - - - - - - - - -|
```

Control Parameter

Four hexadecimal digits. Digits 1-3 are not used. The fourth digit must be a zero.

Input

Input is IBM card coded-characters starting in location INPUT. The characters are not packed.

IBM Card Code

	12	11	0	1	2	3	4	5	6	7	8	9				
A	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0

	12	11	0	1	2	3	4	5	6	7	8	9				
T	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0

	12	11	0	1	2	3	4	5	6	7	8	9				
9	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

	C_1	C_1	C_1	C_1	C_1	C_1	C_1	C_1	C_2	C_2	C_2	C_2	C_2	C_2	C_2	C_2
Typewriter Code	0	0	1	1	1	1	0	0	1	0	0	1	1	1	0	0
	A								T							

	C_1	C_1	C_1	C_1	C_1	C_1	C_1	C_1								
	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
	9															

19023

Figure 13. HOLPR Conversion

Output

Output is typewriter-coded characters starting in location OUTPT. The characters are packed two per binary word.

The input area may overlap the output area if the address INPUT is equal to or greater than the address OUTPT. The subroutine starts processing at location INPUT.

Character Count

This number specifies the number of IBM card-coded characters to be converted. This count is equal to the number of words in the input area. If an odd-count is specified, bits 8-15 of the last word in the output area are not altered.

Error Conditions Detected

Any input character which has no equivalent among the asterisked characters listed in Appendix B is considered an error.

EBPRT

This subroutine converts EBCDIC-coded characters to typewriter-coded characters. The relationship of the coded characters for the conversion to typewriter characters is shown in Figure 14.

Calling Sequence

```
        CALL    EBPRT
        DC      $xxxx     (Control)
        DC      INPUT
        DC      OUTPT
        DC      nnnnn     (Character count)
         :        :
INPUT  |----------|
       |----------|
         :        :
OUTPT  |----------|
       |----------|
```

EBCDIC Code	C_1	C_1	C_1	C_1	C_1	C_1	C_1	C_1	C_2	C_2	C_2	C_2	C_2	C_2	C_2	C_2
	1	1	0	0	0	0	0	1	1	1	1	0	0	0	1	1
					A							T				

Typewriter Code	C_1	C_1	C_1	C_1	C_1	C_1	C_1	C_1	C_2	C_2	C_2	C_2	C_2	C_2	C_2	C_2
	0	0	1	1	1	1	0	0	1	0	0	1	1	1	0	0
					A							T				

18068

Figure 14. EBPRT Conversion

Control Parameter

Four hexadecimal digits. Digits 1-3 are not used. The fourth digit must be a zero.

Input

Input is EBCDIC-coded characters starting in location INPUT. EBCDIC characters are packed two per word.

Output

Output is typewriter-coded characters starting in location OUTPT. The characters are packed two per binary word.

The address INPUT must be equal to or greater than the address OUTPT if overlap of the input and output areas is desired. The subroutine starts processing at location INPUT.

Character Count

This number specifies the number of EBCDIC characters to be converted. This count is not equal to the number of binary words in the input area. If an odd count is specified, bits 8-15 of the last word in the output area are not altered.

Error Conditions Detected

Any input character which has no equivalent among the asterisked characters listed in Appendix B is considered an error.

The IBM 1130 Subroutine Library includes a selection of arithmetic and functional subroutines which are most frequently required because of their general applicability. There are 28 basic subroutines, some of which have several entry points. The various additional entry points allow indexed linkage, and/or a choice of format when working with floating-point numbers.

Table 4 lists the arithmetic and functional subroutines that are included in the Subroutine Library. After a brief description of floating-point data formats, the particulars of each subroutine are presented.

FLOATING POINT DATA FORMATS

Many of the IBM 1130 arithmetic and functional subroutines offer two ranges of precision. These ranges are called standard range and extended range. The standard range provides 23 bits of precision, while the extended range provides up to 31 bits of precision.

Table 4. Arithmetic and Functional Subroutines

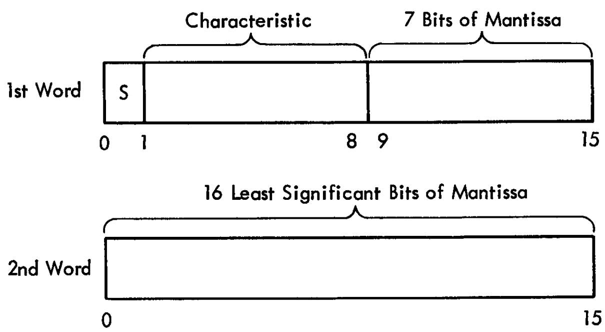
DESCRIPTION	NAME	
	Standard Range	Extended Range
Floating Point		
Add/Subtract	*FADD/*FSUB	*EADD/*ESUB
Multiply	*FMPY	*EMPY
Divide	*FDIV	*EDIV
Load/Store Pseudo-Accumulator	*FLD/*FSTO	*ELD/*ESTO
Trigonometric Sine/Cosine	FSIN/FCOS	ESIN/ECOS
Trigonometric Arctangent	FATN	EATN
Square Root	FSQR	ESQR
Natural Logarithm	FLN	ELN
Exponential (e^x)	FEXP	EEXP
Floating-Point Base to a Fixed Exponent	*FAXI	*EAXI
Floating-Point Base to a Floating Exponent	*FAXB	*EAXB
Integer Float/Unfloat	FLT/FIX	EFLT/EFIX
Fixed Point		
Fixed Base to a Fixed Exponent	*FIXI	
Fixed-Point Square Root	XSQR	
Fixed-Point Double Precision Multiply	XMD	
Fixed-Point Double Precision Divide	XDD	
Special Function		
Floating-Point Reverse Subtract	*FSBR	*ESBR
Floating-Point Reverse Divide	*FDVR	*EDVR
Floating-Point Reverse Sign	FSNR	ESNR
Floating-Point Absolute Value	FABS	EABS
Fixed-Point Reverse Subract	*XSBR	
Fixed-Point Reverse Divide	*XDVR	
Fixed-Point Reverse Sign	XSNR	
Fixed-Point Absolute Value	XABS	
NOTE: By adding an X to those names prefixed with an asterisk, the user can cause the contents of Index Register 1 to be added to the argument address specified by the subroutine calling sequence to form the effective argument address. For example, FADD would become FADDX.		

18117

To achieve correct results from a particular subroutine, the input arguments must be in the proper format.

Standard-Range Format

Standard-range floating-point numbers are stored in the core storage as shown below:



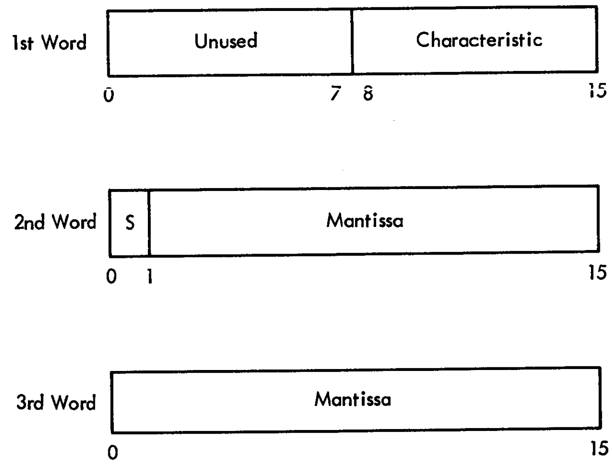
18112

Numbers can consist of up to 23 significant bits with a binary exponent ranging from -128 to +127. Two adjacent storage locations are required for each number. The first (lowest) location must be even-numbered. The sign of the mantissa is contained in bit zero of the first word. Bits one through 8 represent the characteristic and the remaining 23 bits represent the mantissa (absolute value).

The characteristic is formed by adding +128 to the exponent. For example, an exponent of -32 would be represented by a characteristic of 128-32 or 96. An exponent of +100 would be represented by a characteristic of 100 + 128 or 228. Since 128 = 200_8 (80_{16}) the characteristic of a non-negative exponent always has a 1-bit in position 1, while the characteristic of a negative exponent always produces a 0-bit in position 1. A normal zero consists of all zero bits in both the characteristic and the mantissa.

Extended Range Format

Extended range floating-point numbers are stored in three adjacent core locations as shown in the following illustration. Numbers can consist of up to 31 significant bits with a binary exponent ranging from

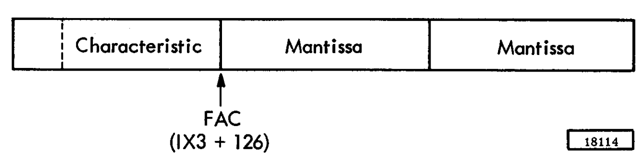


18113

-128 to +127. Bits zero through seven of the first word are unused; bits eight through 15 of the first word represent the characteristic of the exponent (formed in the same manner as in the standard-range format); bit zero of the second word contains the sign of the mantissa; and the remaining 31 bits represent the mantissa (absolute value).

FLOATING-POINT PSEUDO-ACCUMULATOR

IBM 1130 floating-point subroutines sometimes require a register or accumulator which can accommodate numbers in floating-point format. Since all of the hardware registers are only 16 bits in length, a pseudo-accumulator must be set up to contain two- and three-word floating-point numbers. The IBM 1130 pseudo-accumulator (designated FAC for Floating Accumulator) is a three-word register occupying the three highest locations of the transfer vector (see IBM 1130 Assembler Language, C26-5927). The user can refer to it by using Index Register 3 plus a fixed displacement. The format of the pseudo-accumulator is shown below.



18114

NOTE: The effective address of the mantissa will always be even.

PROGRAMMING CONSIDERATIONS

The IBM 1130 subroutines save the condition of the Overflow indicator upon entry and restore that condition before returning to the main program. If an error condition occurs during the execution of the subroutine, the Overflow indicator is forced ON regardless of the condition saved on entry.

Subroutines which use the hardware accumulator (Registers A and Q) do not save and restore its contents. Therefore, a main program should save the contents of the accumulator, if the accumulator is to be used.

CALLING SEQUENCES

The arithmetic and functional subroutines are called via a CALL statement, in some cases followed by a DC statement containing the actual or symbolic address of an argument. In the descriptions which follow, the notations (ARG) and (FAC) refer to the contents of the operand rather than its address. The name FAC refers to the floating-point pseudo-accumulator.

Floating Add

CALL	FADD, FADDX, EADD or EADDX
DC	ARG
Input	Floating augend in FAC Floating addend in location ARG
Result	(FAC)+(ARG) replaces (FAC)

Floating Subtract

CALL	FSUB, FSUBX, ESUB or ESUBX
DC	ARG
Input	Floating minuend in FAC Floating subtrahend in location ARG
Result	(FAC) - (ARG) replaces (FAC)

Floating Multiply

CALL	FMPY or EMPY
DC	ARG
Input	Floating multiplicand in FAC Floating multiplier in location ARG
Result	(FAC) times (ARG) replaces (FAC)

Floating Divide

CALL	FDIV, FDIVX, EDIV or EDIVX
DC	ARG
Input	Floating dividend in FAC Floating dividend in location ARG
Result	(FAC)/(ARG) replaces (FAC)

Load Pseudo-Accumulator

CALL	FLD, FLDX, ELD or ELDX
DC	ARG
Input	Floating-point number in location ARG
Result	(ARG) replaces (FAC)

Store Pseudo-Accumulator

CALL	FSTO, FSTOX, ESTO or ESTOX
Input	Floating-point number in FAC
Result	(FAC) replaces (ARG)

Floating Trigonometric Sine

CALL	FSIN or ESIN
Input	Floating-point argument (in radians) in FAC
Result	Sine of (FAC) replaces (FAC)

Floating Trigonometric Cosine

CALL	FCOS or ECOS
Input	Floating-point argument (in radians) in FAC
Result	Cosine of (FAC) replaces (FAC)

Floating Trigonometric Arctangent

CALL	FATN or EATN
Input	Floating-point argument in FAC
Result	Arctangent of (FAC) replaces (FAC); the result is within 90 degrees (in radians)

Floating Square Root

CALL	FSQR or ESQR
Input	Floating-point argument in FAC
Result	Square root of (FAC) replaces (FAC)

Floating Natural Logarithm

CALL	FLN or ELN
Input	Floating-point argument in FAC
Result	Log_e (FAC) replaces (FAC)

Floating Exponential

CALL	FEXP or EEXP
Input	Floating-point argument in FAC = n
Result	e^n replaces (FAC)

Floating Base to a Fixed Exponent

CALL	FAXI, FAXIX, EAXI or EAXIX
DC	ARG
Input	Floating-point base in FAC Fixed-point exponent in location ARG
Result	(FAC), raised to the exponent contained in ARG, replaces (FAC)

Floating Base to a Floating Exponent

CALL	FAXB, FAXBX, EAXB or EAXBX
DC	ARG
Input	Floating-point base in FAC Floating-point exponent in location ARC
Result	(FAC) raised to the exponent contained in ARG replaces (FAC)

Fix a Floating-Point Number

CALL	FIX or EFIX
Input	Floating-point number in FAC
Result	Fixed-point integer in the A-register

Float a Fixed-Point Number

CALL	FLT or EFLT
Input	Fixed-point integer in the A-Register
Result	Floating-point number in FAC

Fixed Base to a Fixed Exponent

CALL	FIXI or FIXIX
DC	ARG
Input	Fixed-point base in the A-Register Fixed-point exponent in location ARG
Result	(A-Register) raised to the exponent contained in ARG replaces (A-Register)

Fixed-Point Square Root

CALL	XSQR
Input	Fixed-point argument in the A-Register
Result	Square root of (A-Register) replaces (A-Register)

Fixed-Point Double-Precision Multiply

CALL	XMD
DC	ARG
Input	Double-word multiplicand in the A- and Q-Registers Double-word multiplier in location ARG (even address)
Result	Double-word product in the A- and Q-Registers

Fixed-Point Double-Precision Divide

CALL	XDD
DC	ARG
Input	Double-word dividend in the A and Q-Registers Double-word divisor in location ARG
Result	Double-word quotient in the A and Q-Registers

Floating-Point Reverse Subtract

CALL	FSBR, FSBRX, ESBR or ESBRX
DC	ARG
Input	Floating minuend in location ARG Floating subtrahend in FAC
Result	(ARG) - (FAC) replaces (FAC)

Floating-Point Reverse Divide

CALL	FDVR, FDVRX, EDVR or EDVRX
DC	ARG
Input	Floating dividend in location ARG Floating divisor in FAC
Result	(ARG)/(FAC) replaces (FAC)

Floating-Point Reverse Sign

CALL	FSNR or ESNR
Input	Floating-point number, X, in FAC
Result	-X replaces X in FAC

Floating-Point Absolute Value

CALL	FABS or EABS
Input	Floating-Point Number, X, in FAC
Result	Absolute value of X replaces X in FAC

Fixed-Point Reverse Subtract

CALL	XSBR or XSBRX
DC	ARG
Input	Fixed-point minuend in location ARG

	Fixed-point subtrahend in the A-Register
Result	(ARG) - (A-Register) replaces (A-Register)

Fixed-Point Reverse Divide

CALL	XDVR or XDVRX
DC	ARG
Input	Fixed-point dividend in location ARG
	Fixed-point divisor in the A-Register
Result	(ARG)/(A-Register) replaces (A-Register)

Fixed-Point Reverse Sign

CALL	XSNR
Input	Fixed-point number, X, in the A- and Q-Registers
Result	-X replaces X in the A- and Q-Registers

Fixed-Point Absolute Value

CALL	XABS
Input	Fixed-point number, X, in the A and Q Registers
Result	Absolute value of X replaces X in the A- and Q-Registers

SELECTIVE DUMP SUBROUTINES

The IBM 1130 Subroutine Library includes three dump subroutines: Dump Selected Data on Typewriter (Console Printer); Dump Selected Data on Printer; and Dump Status Area. These subroutines allow the user to dump selected portions of core storage during the execution of an object program.

DUMP SELECTED DATA ON TYPEWRITER/ PRINTER

There are two subroutines available for the purpose of selecting an area of core storage and having it dumped out on either the typewriter or the printer. Each of these subroutines has two entry points: one for hexadecimal output, and one for decimal output. The entry points for the various configurations are shown below:

Entry Point	Function of Subroutine
DMTYX	Dump on typewriter in hexadecimal form
DMTYD	Dump on typewriter in decimal form
DMPRX	Dump on printer in hexadecimal form
DMPRD	Dump on printer in decimal form

Calling Sequence

The calling sequence for any of the above functions is as follows:

```
CALL     ENTRY POINT
DC       START
DC       END
```

START and END represent the starting and ending addresses of the portion of core storage to be dumped.

Format

Before the actual dump appears on the selected output device, the user is given one line of status information. This line indicates the status of the Overflow and Carry triggers (ON or OFF), and the contents of the A- and Q-Registers and the three index registers. The register contents are given in both hexadecimal and decimal form regardless of which type of output was requested. The format of the status information is shown below:

OFF	ON	HHHH ± DDDDD	HHHH ± DDDDD
Overflow	Carry	A Register	Q Register

HHHH ± DDDDD HHHH ± DDDDD HHHH ± DDDDD

Index Registers 1, 2, and 3

All other data are dumped eight words to a line, with the address of the first word in each line printed to the left of the line. Hexadecimal data is printed four characters per word; decimal data is printed five digits per word with a preceding plus or minus sign.

Page numbers will not be printed for either subroutine. However, the printer subroutine does provide for automatic page overflow upon sensing a channel-12 punch in the carriage tape.

DUMP STATUS AREA

This subroutine provides a relatively easy and efficient means of dumping the first 80 words of core storage. These words contain status information relating to index registers, interrupt addresses, interval timers, etc. This information may frequently be required when testing a program. It may also be desirable to dump these words before loading, because pressing the Load key destroys the data in the first 80 words of core storage.

This subroutine is called via the following statement:

```
CALL     DMP80
```

The first 80 words of core storage are dumped on the typewriter in hexadecimal form with a space between each word. After typing the last word, the subroutine halts. Pressing the Start key returns control to the main program.

ADDING SUBROUTINES

The user may write subroutines in symbolic language and add them to the Subroutine Library.

The user-added subroutine can be called from either a Symbolic or FORTRAN program by using the appropriate CALL statement (see the publications: IBM 1130 Assembler Language, Form C26-5927, and IBM 1130 FORTRAN Language, Form C26-5933).

Briefly, to add a subroutine, it is necessary to:

1. Write the subroutine in symbolic language.
2. Precede the subroutine source deck (or tape) with one ENT statement for each subroutine entry point (10 maximum).
3. Assemble the subroutine in relocatable form, and add the assembled program to the subroutine library deck.

NOTE: The errors marked with an asterisk are those that are detected after an I/O operation has been initiated.

ERROR		CONTENTS OF A-REGISTER	
		Binary	Hexadecimal
Card			
	*Last card	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 0 0 0
	*Lost data *Feed check *Read check *Punch check	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1	0 0 0 1
	1442 not ready	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0	1 0 0 0
	Illegal CALL requested	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1	1 0 0 1
Keyboard-Console Printer			
	Device not ready	0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0	2 0 0 0
	Illegal CALL requested	0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1	2 0 0 1
Paper Tape			
	Device not ready	0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0	3 0 0 0
	Illegal CALL requested	0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 1	3 0 0 1
Disk			
	*Attempt to write in file protected area	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1	0 0 0 3
	*Read check remaining after ten attempts	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1	0 0 0 1
	*Write check remaining after ten attempts	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0	0 0 0 2
	Device not ready	0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0	5 0 0 0
	Illegal CALL requested	0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1	5 0 0 1
Printer			
	*Device not ready	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0	0 0 0 2
	*Channel 9 detected	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1	0 0 0 3
	*Channel 12 detected	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0	0 0 0 4
	End of forms	0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0	6 0 0 0
	Illegal CALL requested	0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1	6 0 0 1
Plotter			
	Plotter not ready	0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0	7 0 0 0
	Illegal CALL requested	0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1	7 0 0 1

19016

APPENDIX B. CHARACTER CODE CHART

* Recognized by all Conversion subroutines

NOTE: Codes that are not asterisked are recognized only by the SPEED subroutine.

Ref No.	EBCDIC Binary 0123	4567	EBCDIC Hex	IBM Card Code Rows 12	11	0	9	8	7-1	IBM Card Code Hex	Graphic and Control Name		Paper Tape Hex	Typewriter Hex
0	0000	0000	00	12		0	9	8	1	B03	NUL			
1	↓	0001	01	12			9		1	901				
2		0010	02	12			9		2	881				
3		0011	03	12			9		3	841				
4		0100	04	12			9		4	821	PF	Punch Off		
5*		0101	05	12			9		5	811	HT	Horiz.Tab		41
6		0110	06	12			9		6	809	LC	Lower Case		
7*		0111	07	12			9		7	805	DEL	Delete	FF	
8		1000	08	12			9	8		803				
9		1001	09	12			9	8	1	903				
10		1010	0A	12			9	8	2	883				
11		1011	0B	12			9	8	3	843				
12		1100	0C	12			9	8	4	823				
13		1101	0D	12			9	8	5	813				
14		1110	0E	12			9	8	6	80B				
15	↓	1111	0F	12			9	8	7	807				
16	0001	0000	10	12	11		9	8	1	D03				
17	↓	0001	11		11		9		1	501				
18		0010	12		11		9		2	481				
19		0011	13		11		9		3	441				
20*		0100	14		11		9		4	421	RES	Restore		05
21*		0101	15		11		9		5	411	NL	New Line		81
22*		0110	16		11		9		6	409	BS	Backspace		11
23		0111	17		11		9		7	405	IDL	Idle		
24		1000	18		11		9	8		403				
25		1001	19		11		9	8	1	503				
26		1010	1A		11		9	8	2	483				
27		1011	1B		11		9	8	3	443				
28		1100	1C		11		9	8	4	423				
29		1101	1D		11		9	8	5	413				
30		1110	1E		11		9	8	6	40B				
31	↓	1111	1F		11		9	8	7	407				
32	0010	0000	20		11	0	9	8	1	703				
33	↓	0001	21			0	9		1	301				
34		0010	22			0	9		2	281				
35		0011	23			0	9		3	241				
36		0100	24			0	9		4	221	BYP	Bypass		
37*		0101	25			0	9		5	211	LF	Line Feed		03
38*		0110	26			0	9		6	209	EOB	End of Block	80	
39		0111	27			0	9		7	205	PRE	Prefix		
40		1000	28			0	9	8		203				
41		1001	29			0	9	8	1	303				
42		1010	2A			0	9	8	2	283				
43		1011	2B			0	9	8	3	243				
44		1100	2C			0	9	8	4	223				
45		1101	2D			0	9	8	5	213				
46		1110	2E			0	9	8	6	20B				
47	↓	1111	2F			0	9	8	7	207				
48	0011	0000	30	12	11	0	9	8	1	F03				
49	↓	0001	31				9		1	101				
50		0010	32				9		2	081				
51		0011	33				9		3	041				
52		0100	34				9		4	021	PN	Punch On		
53*		0101	35				9		5	011	RS	Ribbon Shift		09
54		0110	36				9		6	009	UC	Upper Case		
55		0111	37				9		7	005	EOT	End of Trans.		
56		1000	38				9	8		003				
57		1001	39				9	8	1	103				
58		1010	3A				9	8	2	083				
59		1011	3B				9	8	3	043				
60		1100	3C				9	8	4	023				
61		1101	3D				9	8	5	013				
62		1110	3E				9	8	6	00B				
63	↓	1111	3F				9	8	7	007				

19017

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	Graphic and Control Name	Paper Tape Hex	Typewriter Hex
64*	0100	0000	40	no punches						000	(space)	10	21
65		0001	41	12		0	9		1	B01			
66		0010	42	12		0	9		2	A81			
67		0011	43	12		0	9		3	A41			
68		0100	44	12		0	9		4	A21			
69		0101	45	12		0	9		5	A11			
70		0110	46	12		0	9		6	A09			
71		0111	47	12		0	9		7	A05			
72		1000	48	12		0	9	8		A03			
73		1001	49	12				8	1	902			
74*		1010	4A	12				8	2	882	¢	7A	02
75*		1011	4B	12				8	3	842	. (period)	6B	00
76*		1100	4C	12				8	4	822	<	7C	DE
77*		1101	4D	12				8	5	812	(	6D	FE
78*		1110	4E	12				8	6	80A	+	6E	DA
79*		1111	4F	12				8	7	806	| (logical OR)	7F	C6
80*	0101	0000	50	12						800	&	70	44
81		0001	51	12	11		9		1	D01			
82		0010	52	12	11		9		2	C81			
83		0011	53	12	11		9		3	C41			
84		0100	54	12	11		9		4	C21			
85		0101	55	12	11		9		5	C11			
86		0110	56	12	11		9		6	C09			
87		0111	57	12	11		9		7	C05			
88		1000	58	12	11		9	8		C03			
89		1001	59		11			8	1	502			
90*		1010	5A		11			8	2	482	!	4A	42
91*		1011	5B		11			8	3	442	$	5B	40
92*		1100	5C		11			8	4	422	*	4C	D6
93*		1101	5D		11			8	5	412	)	5D	F6
94*		1110	5E		11			8	6	40A	;	5E	D2
95*		1111	5F		11			8	7	406	¬ (logical NOT)	4F	F2
96*	0110	0000	60		11					400	- (dash)	40	84
97*		0001	61			0			1	300	/	31	BC
98		0010	62		11	0	9		2	681			
99		0011	63		11	0	9		3	641			
100		0100	64		11	0	9		4	621			
101		0101	65		11	0	9		5	611			
102		0110	66		11	0	9		6	609			
103		0111	67		11	0	9		7	605			
104		1000	68		11	0	9	8		603			
105		1001	69			0		8	1	302			
106		1010	6A	12	11					C00			
107*		1011	6B			0		8	3	242	, (comma)	3B	80
108*		1100	6C			0		8	4	222	%	2C	06
109*		1101	6D			0		8	5	212	_ (underscore)	3D	BE
110*		1110	6E			0		8	6	20A	>	3E	46
111*		1111	6F			0		8	7	206	?	2F	86
112	0111	0000	70	12	11	0				E00			
113		0001	71	12	11	0	9		1	F01			
114		0010	72	12	11	0	9		2	E81			
115		0011	73	12	11	0	9		3	E41			
116		0100	74	12	11	0	9		4	E21			
117		0101	75	12	11	0	9		5	E11			
118		0110	76	12	11	0	9		6	E09			
119		0111	77	12	11	0	9		7	E05			
120		1000	78	12	11	0	9	8		E03			
121		1001	79					8	1	102			
122*		1010	7A					8	2	082	:	20	82
123*		1011	7B					8	3	042	#	0B	C0
124*		1100	7C					8	4	022	@	1C	04
125*		1101	7D					8	5	012	' (apostrophe)	0D	E6
126*		1110	7E					8	6	00A	=	0E	C2
127*		1111	7F					8	7	006	"	1F	E2

19017

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	Graphic and Control Name	Paper Tape Hex	Typewriter Hex
128	1000	0000	80	12		0		8	1	B02			
129		0001	81	12		0			1	B00	a		
130		0010	82	12		0			2	A80	b		
131		0011	83	12		0			3	A40	c		
132		0100	84	12		0			4	A20	d		
133		0101	85	12		0			5	A10	e		
134		0110	86	12		0			6	A08	f		
135		0111	87	12		0			7	A04	g		
136		1000	88	12		0		8		A02	h		
137		1001	89	12		0	9			A01	i		
138		1010	8A	12		0		8	2	A82			
139		1011	8B	12		0		8	3	A42			
140		1100	8C	12		0		8	4	A22			
141		1101	8D	12		0		8	5	A12			
142		1110	8E	12		0		8	6	A0A			
143		1111	8F	12		0		8	7	A06			
144	1001	0000	90	12	11			8	1	D02			
145		0001	91	12	11				1	D00	j		
146		0010	92	12	11				2	C80	k		
147		0011	93	12	11				3	C40	l		
148		0100	94	12	11				4	C20	m		
149		0101	95	12	11				5	C10	n		
150		0110	96	12	11				6	C08	o		
151		0111	97	12	11				7	C04	p		
152		1000	98	12	11			8		C02	q		
153		1001	99	12	11		9			C01	r		
154		1010	9A	12	11			8	2	C82			
155		1011	9B	12	11			8	3	C42			
156		1100	9C	12	11			8	4	C22			
157		1101	9D	12	11			8	5	C12			
158		1110	9E	12	11			8	6	C0A			
159		1111	9F	12	11			8	7	C05			
160	1010	0000	A0		11	0		8	1	702			
161		0001	A1		11	0			1	700			
162		0010	A2		11	0			2	680	s		
163		0011	A3		11	0			3	640	t		
164		0100	A4		11	0			4	620	u		
165		0101	A5		11	0			5	610	v		
166		0110	A6		11	0			6	608	w		
167		0111	A7		11	0			7	604	x		
168		1000	A8		11	0		8		602	y		
169		1001	A9		11	0	9			601	z		
170		1010	AA		11	0		8	2	682			
171		1011	AB		11	0		8	3	642			
172		1100	AC		11	0		8	4	622			
173		1101	AD		11	0		8	5	612			
174		1110	AE		11	0		8	6	60A			
175		1111	AF		11	0		8	7	606			
176	1011	0000	B0	12	11	0		8	1	F02			
177		0001	B1	12	11	0			1	F00			
178		0010	B2	12	11	0			2	E80			
179		0011	B3	12	11	0			3	E40			
180		0100	B4	12	11	0			4	E20			
181		0101	B5	12	11	0			5	E10			
182		0110	B6	12	11	0			6	E08			
183		0111	B7	12	11	0			7	E04			
184		1000	B8	12	11	0		8		E02			
185		1001	B9	12	11	0	9			E01			
186		1010	BA	12	11	0		8	2	E82			
187		1011	BB	12	11	0		8	3	E42			
188		1100	BC	12	11	0		8	4	E22			
189		1101	BD	12	11	0		8	5	E12			
190		1110	BE	12	11	0		8	6	E0A			
191		1111	BF	12	11	0		8	7	E06			

19017

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	Graphic and Control Name	Paper Tape Hex	Typewriter Hex
192	1100	0000	C0	12		0				A00	(+ zero)		
193*		0001	C1	12					1	900	A	61	3C or 3E
194*		0010	C2	12					2	880	B	62	18 or 1A
195*		0011	C3	12					3	840	C	73	1C or 1E
196*		0100	C4	12					4	820	D	64	30 or 32
197*		0101	C5	12					5	810	E	75	34 or 36
198*		0110	C6	12					6	808	F	76	10 or 12
199*		0111	C7	12					7	804	G	67	14 or 16
200*		1000	C8	12				8		802	H	68	24 or 26
201*		1001	C9	12			9			801	I	79	20 or 22
202		1010	CA	12		0	9	8	2	A83			
203		1011	CB	12		0	9	8	3	A43			
204		1100	CC	12		0	9	8	4	A23			
205		1101	CD	12		0	9	8	5	A13			
206		1110	CE	12		0	9	8	6	A0B			
207	↓	1111	CF	12		0	9	8	7	A07			
208	1101	0000	D0		11	0				600	(- zero)		
209*		0001	D1		11				1	500	J	51	7C or 7E
210*		0010	D2		11				2	480	K	52	58 or 5A
211*		0011	D3		11				3	440	L	43	5C or 5E
212*		0100	D4		11				4	420	M	54	70 or 72
213*		0101	D5		11				5	410	N	45	74 or 76
214*		0110	D6		11				6	408	O	46	50 or 52
215*		0111	D7		11				7	404	P	57	54 or 56
216*		1000	D8		11			8		402	Q	58	64 or 66
217*		1001	D9		11		9			401	R	49	60 or 62
218		1010	DA	12	11		9	8	2	C83			
219		1011	DB	12	11		9	8	3	C43			
220		1100	DC	12	11		9	8	4	C23			
221		1101	DD	12	11		9	8	5	C13			
222		1110	DE	12	11		9	8	6	C0B			
223	↓	1111	DF	12	11		9	8	7	C07			
224	1110	0000	E0			0		8	2	282			
225		0001	E1		11	0	9		1	701			
226*		0010	E2			0			2	280	S	32	98 or 9A
227*		0011	E3			0			3	240	T	23	9C or 9E
228*		0100	E4			0			4	220	U	34	B0 or B2
229*		0101	E5			0			5	210	V	25	B4 or B6
230*		0110	E6			0			6	208	W	26	90 or 92
231*		0111	E7			0			7	204	X	37	94 or 96
232*		1000	E8			0		8		202	Y	38	A4 or A6
233*		1001	E9			0	9			201	Z	29	A0 or A2
234		1010	EA		11	0	9	8	2	683			
235		1011	EB		11	0	9	8	3	643			
236		1100	EC		11	0	9	8	4	623			
237		1101	ED		11	0	9	8	5	613			
238		1110	EE		11	0	9	8	6	60B			
239	↓	1111	EF		11	0	9	8	7	607			
240*	1111	0000	F0			0				200	0	1A	C4
241*		0001	F1						1	100	1	01	FC
242*		0010	F2						2	080	2	02	D8
243*		0011	F3						3	040	3	13	DC
244*		0100	F4						4	020	4	04	F0
245*		0101	F5						5	010	5	15	F4
246*		0110	F6						6	008	6	16	D0
247*		0111	F7						7	004	7	07	D4
248*		1000	F8					8		002	8	08	E4
249*		1001	F9				9			001	9	19	E0
250		1010	FA	12	11	0	9	8	2	E83			
251		1011	FB	12	11	0	9	8	3	E43			
252		1100	FC	12	11	0	9	8	4	E23			
253		1101	FD	12	11	0	9	8	5	E13			
254		1110	FE	12	11	0	9	8	6	E0B			
255	↓	1111	FF	12	11	0	9	8	7	E07			

19017

C26-5929-0

IBM 1130 Printed in U.S.A. C26-5929-0

IBM®
International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N. Y. 10601