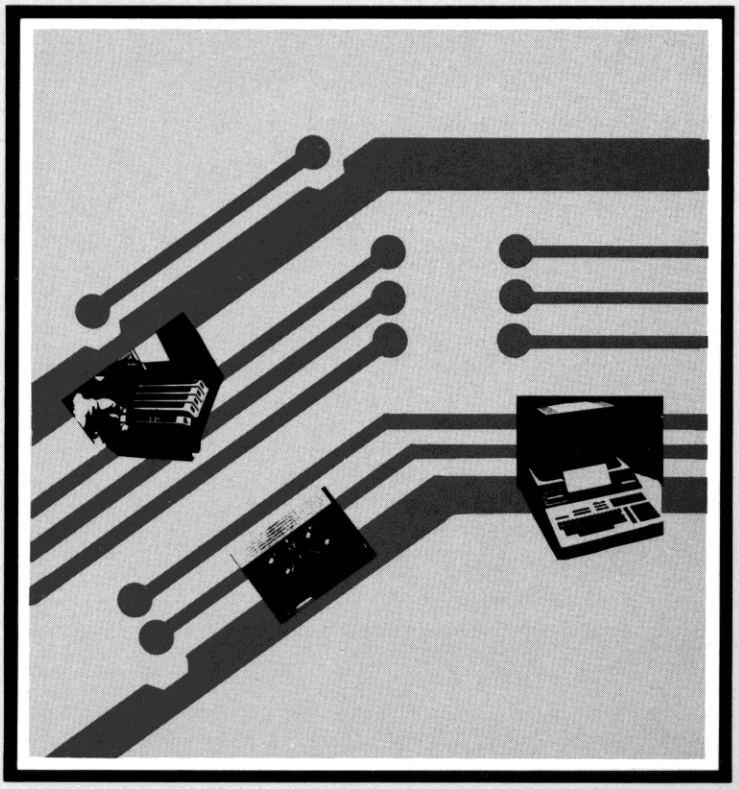
HP 9800 Computer Systems

Advanced Programming ROM

For the HP 9835/HP 9845



HEWLETT PACKARD

HEWLETT PACKARD

Warranty Statement

Hewlett-Packard products are warranted against defects in materials and workmanship. For Hewlett-Packard Desktop Computer Division products sold in the U.S.A. and Canada, this warranty applies for ninety (90) days from date of delivery.* Hewlett-Packard will, at its option, repair or replace equipment which proves to be defective during the warranty period. This warranty includes labor, parts, and surface travel costs, if any. Equipment returned to Hewlett-Packard for repair must be shipped freight prepaid. Repairs necessitated by misuse of the equipment, or by hardware, software, or interfacing not provided by Hewlett-Packard are not covered by this warranty.

HP warrants that its software and firmware designated by HP for use with a CPU will execute its programming instructions when properly installed on that CPU. HP does not warrant that the operation of the CPU, software, or firmware will be uninterrupted or error free.

NO OTHER WARRANTY IS EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. HEWLETT-PACKARD SHALL NOT BE LIABLE FOR CONSEQUENTIAL DAMAGES.

* For other countries, contact your local Sales and Service Office to determine warranty terms.

Advanced Programming ROM

Part No. 09845-93065
Microfiche No. 09845-96065

Hewlett-Packard Desktop Computer Division
3404 East Harmony Road, Fort Collins, Colorado 80525

Printing History

This manual is for use with the System 35A/B or 45B/C Desktop Computers. It is a slightly revised version of the Advanced Programming ROM Manual, part number 09845-92065.

The changes which were incorporated into this latest edition are summarized in the System 45 Manual Revision Package (P/N 09845-93099). This package outlines the changes and additions that have been made to System 45 manuals.

New editions of this manual will incorporate all material updated since the previous edition. Update packages may be issued between editions and contain replacement and additional pages to be merged into the manual by the user. Each updated page will be indicated by a revision date at the bottom of the page. A vertical bar in the margin indicates the changes on each page. Note that pages which are rearranged due to changes on a previous page are not considered revised.

The manual printing date and part number indicate its current edition. The printing date changes when a new edition is printed. (Minor corrections and updates which are incorporated at reprint do not cause the date to change.) The manual part number changes when extensive technical changes are incorporated.

April, 1981...First Edition; Updated pages: ii, 1, 2

Table of Contents

Chapter 1
General Information

Overview

The Advanced Programming ROM provides extended computing capabilities for your desktop computer. This ROM enables you to perform such functions as ordering list data in numerical or lexical order and searching lists for conditions which you specify. These features can be very useful in the areas of mathematics, statistics, and information processing. This manual explains and demonstrates the programming features provided by the Advanced Programming ROM.

You should be familiar with the basic operation of your system before attempting to use the Advanced Programming ROM and this manual. Refer to the System 35 Operating and Programming Manual or the System 45 BASIC Programming Manual for this information.

Equipment Supplied

HP 9835 Part Number	HP 9845B/C Part Number	Item
98336A	98414A	Advanced Programming ROM
09845-93065	09845-93065	Advanced Programming ROM Manual
09835-90448	09845-90448	Advanced Programming ROM Lexical Tables Cartridge

ROM Installation

The System 35 Advanced Programming ROM is plugged into a 9835 ROM drawer, which can then be inserted into any of the four ROM slots at the lower front of the computer. Refer to the System 35 Owner's Manual for installation procedures.

The System 45 Advanced Programming ROM is plugged into the right ROM drawer (black-labeled ROMs). Refer to the System 45 Installation, Operation and Test Manual for installation procedures.

Lexical Tables Cartridge

An Advanced Programming ROM Lexical Tables Cartridge is provided for use with the Advanced Programming ROM. The cartridge contains ASCII and local language collating tables which can be modified for particular collating applications. Descriptions of the tables and instructions for their use are found in Appendix A.

Manual Requirements

Before using this manual or the Advanced Programming ROM, you should be familiar with the basic operating procedures of your desktop computer as explained in the System 45 BASIC Programming Manual or the System 35 Operating and Programming Manual.

Chapter 2
Data Manipulation

page 9 • **MAT SORT** (orders data records within an array)
page 2 • **MAT REORDER** (orders an array according to the contents of an existing pointer array)
page 28 • **MAT SEARCH** (provides information about user-defined conditions within an array)

Terms

- **Record** – represents data which is manipulated as a unit within an array.
- **Key** – a data item within a record used to identify the record for sorting purposes.
- **Key specifier** – a format used in a syntax to specify primary and/or secondary keys within a record.
- **Pointer array** – a one-dimensional numeric array which contains the sorting order of specified records.
- **Location specifier** – a format used within a MAT SEARCH statement syntax to specify the locations to be searched.

Statement Syntax

MAT SORT source array (key specifier) [[substring specifier]]
[DES] [, (secondary key specifier) [[substring specifier]] [DES]...] [TO pointer array]

MAT REORDER object array BY pointer array [, dimension specifier]

MAT SEARCH source array (location specifier), condition; variable [, starting address]

condition:

LOC (relational operator-expression)
#LOC (relational operator-expression)
MAX
MIN

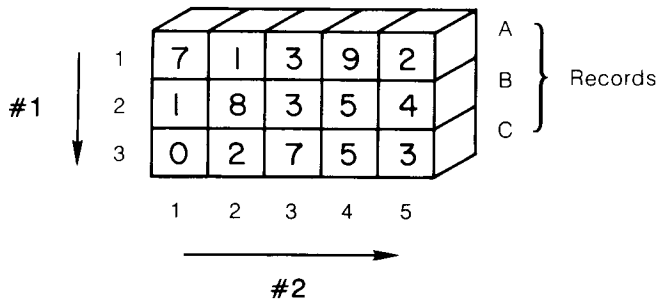
Introduction

The statements described in this chapter are used as programming aids to manipulate data. They concern the sorting and searching of numeric and string arrays. Each of these statements can be executed from within a program or from the keyboard. Examples of each statement are included to demonstrate its use.

Array Structure and Terminology

For a better understanding of the sorting and searching processes introduced in this manual, a brief description of array structure and terminology is helpful.

The following illustration represents a two-dimensional array containing numeric data.



Array X

It is dimensioned 3 rows by 5 columns as follows:

```
OPTION BASE 1
DIM X(3,5)
```

The subscripts within parentheses (3,5) are written in the order in which the array is dimensioned (i.e., in the order of the numbered arrows). Throughout this manual dimensioning assumes OPTION BASE 1 (i.e., numbering begins with 1, not 0). The dimension statement specifies that the array name is X and that it is three rows by five columns.

An array **record** represents data (string or numeric) which is manipulated as a unit within an array. For example, if you arrange a list of names in a specified order, each name is considered a record within that list. Similarly, an array might contain a list of social security numbers. If you arrange them in a particular order, each social security number is considered a record within that array.

In the previous example, if array X is to be rearranged by rows, each row is considered a record and ordering is performed along the first dimension. That is, the positions of the rows in relationship to each other along the first dimension are rearranged. Likewise, if the array is to be reordered by column, each column is considered a record and reordering is performed along the second dimension.

In the previous illustration, assume that each row is a record (A,B, and C). If the records are to be sorted, a **key** is needed to determine how each row is to be ordered in relationship to the others. Assume that the records are to be sorted in ascending order according to the value of the first number in each record. The first column, then, contains the keys for this sort. The sorting process sorts the keys in ascending order along the dimension in which they lie. In so doing, the records in which the keys lie are rearranged.

The keys selected are described by the following format:

```
(*,1)
```

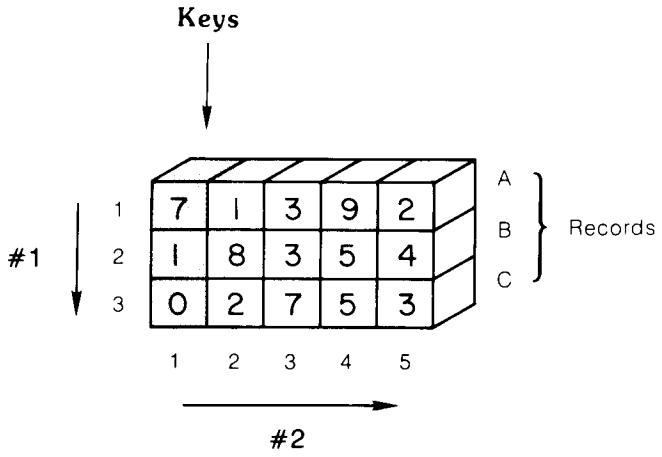
The asterisk indicates a varying subscript. The first dimension subscript is varied over its range of values (1 to 2 to 3) to designate a key in all three records (rows). The second dimension subscript is fixed at 1 to indicate that the keys lie in the first column of each record.

The format (*, 1) is called a **key specifier**. The asterisk replaces the subscript which corresponds to the dimension along which the records lie, in this case, the rows. The number 1 indicates where the key is located within each record. In this way, the key specifier indicates how the array is partitioned into records, and where the keys are located within those records.

If all the combinations of the key specifier in the previous example are listed, a description of the individual keys is obtained as follows.

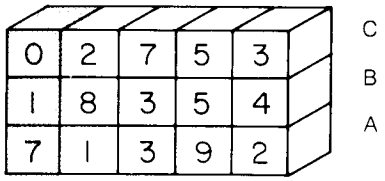
1st Subscript	2nd Subscript	Key
1	1	1st
2	1	2nd
3	1	3rd

The shaded cells in the next figure represent the keys described by the key specifier.



ARRAY X

Upon execution, the sorting process arranges the records in ascending order along the first dimension according to the values of their respective keys. The sorted array is shown next.

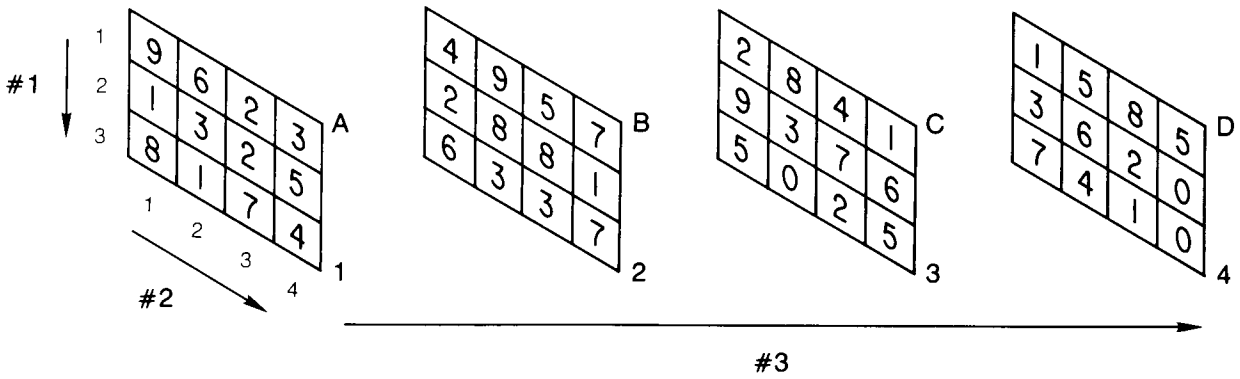


ARRAY X AFTER SORTING

The records, or rows, are repositioned in relationship to each other according to the value of their keys. The contents of the records are left unchanged.

Note that any corresponding numbers within the rows could be selected as keys. The column containing the keys would be described by the key specifier to reflect the proper keys.

As another example, assume array Y is a three dimensional numeric array as shown.



ARRAY Y WITH KEYS

It is dimensioned in the order of the numbered arrows (3 rows by 4 columns by 4 planes):

```
OPTION BASE 1
DIM Y(3,4,4)
```

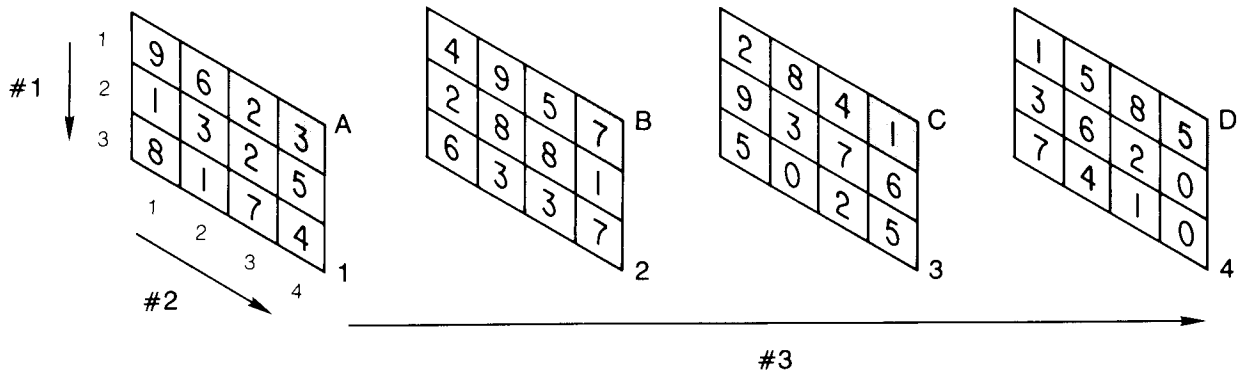
Assume that the array is divided into records (A,B,C, & D) along the third dimension, that is, each plane is considered a record. The keys selected must also lie along the third dimension since there must be one for each record. If the upper right number in each record is designated as a key, the key specifier is

```
(1,4,*)
```

where the asterisk indicates that the third subscript is varied over its entire range of values (1 through 4). In this way, a total of four keys is described as shown.

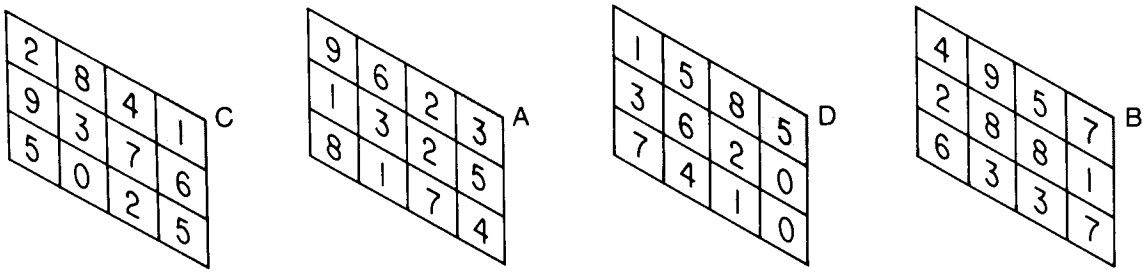
1st Subscript	2nd Subscript	3rd Subscript	Key
1	4	1	1st
1	4	2	2nd
1	4	3	3rd
1	4	4	4th

The shaded cells in the following illustration represent the four keys described by the key specifier (1,4,*).



ARRAY Y WITH KEYS

If the array records are sorted in ascending order, they are rearranged along the third dimension according to the values of their keys. The following illustration represents the array Y after it is sorted.



ARRAY Y AFTER SORTING

Note that the records have been rearranged according to the ascending values of their keys.

MAT SORT Statement

Sorting Numeric Data

The MAT SORT statement is used to order data records in an array. If the data is numeric, it can be sorted in either ascending or descending order. If the array contains string data, it can be sorted in either lexical (alphabetical) or reverse lexical order.

Syntax:

MAT SORT numeric source array (key specifier)

The source array represents the array in which sorting is performed. The key specifier allows you to specify the keys by which the records of data are sorted. **If the source array is one-dimensional, no key specifier need be included.**

Example:

```
MAT SORT A(1,*,3)
```

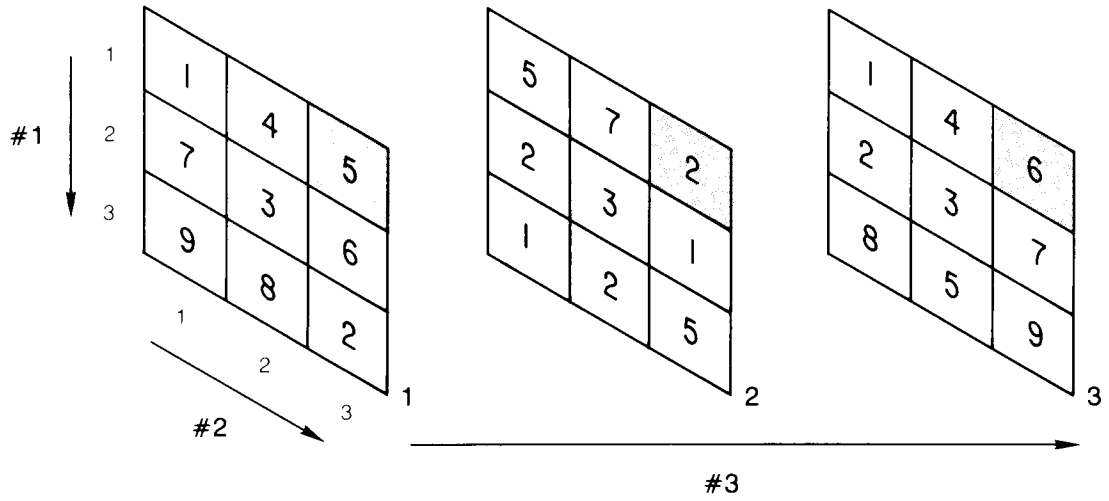
In this case, A is a numeric array in which sorting is performed. The key specifier indicates that A is a three dimensional array and that its records lie along the second dimension. The position of the asterisk within the key specifier determines which subscript is varied, and therefore, how the array is divided into records. If the first subscript were to identify records instead of the second, MAT SORT A(*,2,3) would be entered. The array in which sorting is performed can have no more than six dimensions. Numeric sorting comparisons are performed in the INTEGER mode for integer-precision arrays and in the REAL mode for short or real-precision arrays.

The default order of sorting is ascending. In the previous example statement, the records are sorted in ascending order. If descending order is desired, the correct entry is

```
MAT SORT A(1,*,3) DES
```

where the letters DES included after the key specifier indicate descending order.

The following example serves to explain the MAT SORT process. Assume array Data is a three dimensional numeric array containing random numbers.



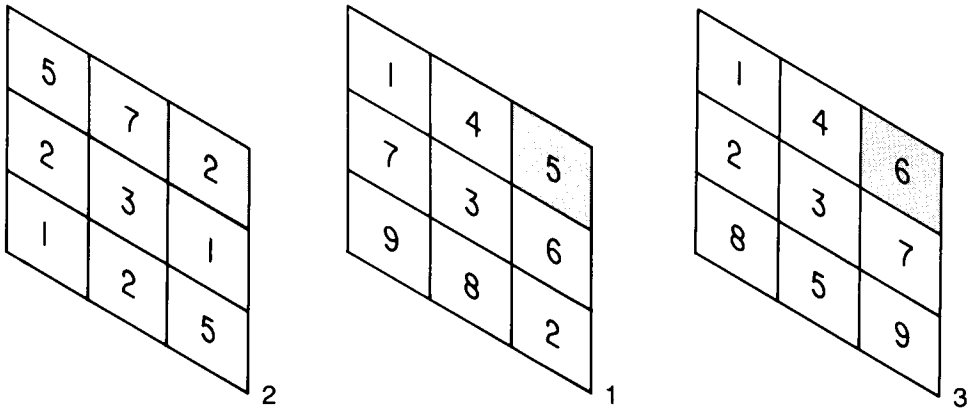
ARRAY Data BEFORE SORTING

This illustration is a graphical representation of the array and its contents. Each record is numbered as are its rows and columns. The numbered arrows represent the order in which the array was originally dimensioned.

Assume that the records are to be sorted in ascending order and that the numbers in the upper right location in each record are designated as keys. Since the records all lie along the third dimension, the third subscript is replaced by an asterisk in the key specifier. The proper sorting statement is

```
MAT SORT Data(1,3,*)
```

where the first two numbers indicate the upper right location of each record. The asterisk indicates that the records are selected by varying the third subscript over its range of values (1 to 2 to 3). After the sort is performed, the array is rearranged as shown next.



ARRAY Data AFTER SORTING

Note that the order of the records is changed according to the value of their keys.

A pointer array can be specified in a sorting statement to maintain a record of how the source array should be rearranged.

Syntax:

```
MAT SORT source array (key specifier) TO pointer array
```

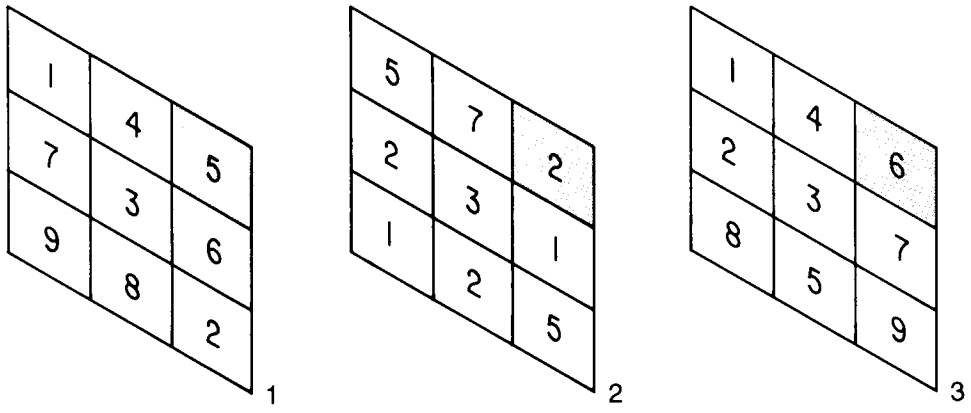
When a pointer array is included in a MAT SORT statement, the sorting process does **not** rearrange the source array. Instead, it fills the pointer array with a series of numbers representing the order of the source array records as if they were sorted. In this way, the source array is not disturbed, but the order in which its records should be sorted is maintained for future use.

The pointer array must be a one-dimensional numeric array and it must be the same length as the range of records to be sorted. The pointer array does not contain the contents of the records, but rather, a series of numbers representing the order in which the records would be sorted.

For example, assume that the sorting statement in the previous example is modified to include the pointer array Point. The pointer array must be dimensioned to be three elements in length to accommodate the number of records in the source array (`DIM Point(3)`). The sorting statement is modified as follows:

```
MAT SORT Data(1,3,*) TO Point
```

Upon execution, the source array remains unchanged as shown.



ARRAY Data AFTER SORTING

However, the pointer array now contains the order of the records as if they were rearranged.

2	1	3
1	2	3

POINTER ARRAY Point AFTER SORTING

Note that the numbers in the pointer array correspond to the sorted order of the records in the previous example. However, the source array is not rearranged.

The numbers in the pointer array are the values that the varied subscript would assume in designating each record. For example, if Data was dimensioned DIM Data (3,3,−1:1), then Point would contain

0	-1	1
1	2	3

This allows the pointer array to be used for indirect reference into the original array. Thus, after sorting this example, Data (3,1, Point(2)) = 9.

If two items described by the key specifier are identical, a secondary key specifier can be used to complete the sort. The sorting process utilizes the order of the data described by the secondary key specifier to arrange the records containing identical primary keys. Should further identical data be described by the secondary key specifier, additional key specifiers can be included. The asterisks must appear in the same respective positions in all related key specifiers.

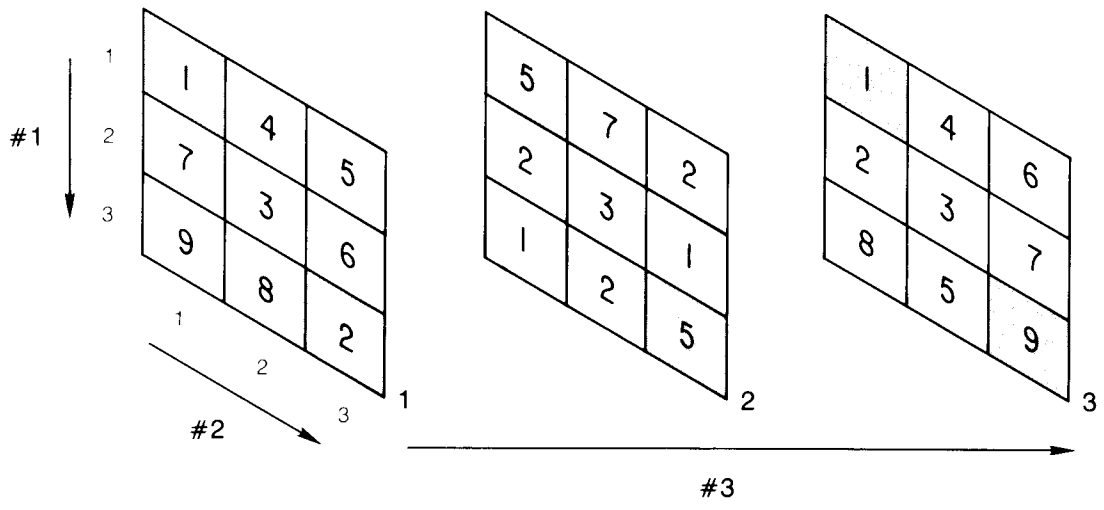
Syntax:

MAT SORT source array ⟨key specifier⟩ [, ⟨secondary key specifier⟩...]

Note that commas are used to separate individual key specifiers. All key specifiers in a given MAT SORT statement must partition the array into records in the same way. That is, the asterisk must always appear in the same position.

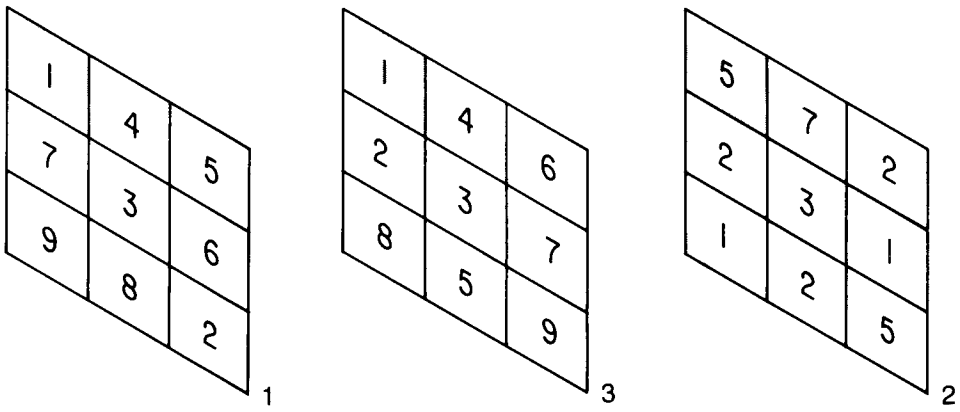
Referring to the previous array example, assume that the records are to be sorted using their upper left location contents as keys. The correct sorting statement is MAT SORT Data(1,1,*), where (1,1,*) specifies the upper left numbers as keys. An ambiguity would develop, however, since the number "1" appears in the keys of both records one and three. The values of another series of keys can be used to determine which "1" should be ordered first. Assume that the numbers in the lower right locations are described by a secondary key specifier. The proper sorting statement is:

```
MAT SORT Data(1,1,*),(3,3,*)
```



ARRAY Data WITH PRIMARY & SECONDARY KEY SPECIFIERS

Since identical numbers have been encountered in records one and three, the sorting process examines the lower right keys of the records (as described by the secondary key specifier(3,3,*)) to complete the sort. Since the "2" of record one precedes the "9" of record three, record one is ordered before record three. Upon execution of the sorting statement, array Data is rearranged as shown.



ARRAY Data AFTER SORTING

If identical items cannot be differentiated, then the relative order of the sorted records is indeterminate (due to the nature of the sort).

The program shown next demonstrates a use of numeric sorting. A list of salespeople, their districts, and their sales volumes is sorted according to district and volume.

```
10   ! ***************************************************************
20   ! ***       This program demonstrates the use of the          ***
30   ! ***                    MAT SORT statement.                  ***
40   ! ***************************************************************
50                                     !
60     OPTION BASE 1                   ! Select OPTION BASE.
70     DIM Salespeople$(9)[20],Sales(12,2,9),Rank(9) ! Dimension
80                                     ! source and pointer arrays.
90                                     !
100  ! ***************************************************************
110  ! *** Data represents district, sales volume, salespeople.  ***
120  ! ***************************************************************
130                                    !
140      DATA 3,3000,1,6000,2,1900,1,1200,3
150      DATA 1400,1,2500,3,900,2,2200,2,1700
160      DATA JILL TANDY,DON DEEDS,RICK HILL,SAM SPADE,NICK DANGER
170      DATA JOE JONES,HARRY WHITE,BARB SMITH,H.J. STEED
180                                    !
190  ! ***************************************************************
200  ! ***             Print headings and sort.                    ***
210  ! ***************************************************************
220                                    !
230      Month=6                       ! Select month of sales survey.
240      PRINT TAB(10);"   DATA FOR MONTH";Month;LIN(1)
250      PRINT "SALESPEOPLE";TAB(17);"DISTRICT";TAB(33);"SALES";LIN(1)
```

```
260                                     !
270                                     !
280       FOR I=1 TO 9
290         READ Sales(Month,1,I),Sales(Month,2,I)  ! Input district
300       NEXT I                        ! and sales volume.
310                                     !
320     MAT READ Salespeople$           ! Input salespeople.
330                                     !
340       FOR I=1 TO 9
350         PRINT Salespeople$(I);TAB(19);Sales(Month,1,I);TAB(32);
360         PRINT Sales(Month,2,I)
370       NEXT I                        ! Print the unsorted sales data.
380                                     !
390     WAIT 5000                       ! Wait 5 seconds before continuing.
400     PRINT PAGE                      ! Clear the screen.
410                                     !
420     PRINT TAB(10);"  RESULTS FOR MONTH";Month;LIN(1)  ! Print
430                                     ! a new heading.
440     MAT SORT Sales(Month,1,*),(Month,2,*) DES TO Rank ! Sort
450                                     ! the sales by district; use
460                                     ! sales volume as a secondary key.
470                                     !
480  ! ***************************************************************
490  ! *** Establish FOR/NEXT loop for printing sales results.   ***
500  ! ***************************************************************
510     District=-1                     ! Start with invalid district no.
520       FOR I=1 TO 9
530         IF District=Sales(Month,1,Rank(I)) THEN Skip ! Test: Has
540                                     ! district changed?
550                                     ! If so, skip print routine.
560                                     ! If not, continue.
570         District=Sales(Month,1,Rank(I)) ! Set District to new value.
580                                     !
590         PRINT TAB(10);"  DISTRICT";District;"RESULTS" ! Print heading.
600 Skip:   PRINT Salespeople$(Rank(I));TAB(30);"  ";Sales(Month,2,Rank(I))
610       NEXT I                        ! Go on to next salesperson.
620   END
```

The program first prints the unsorted list. The sort is performed by district using sales volume as the secondary key specifier. If the program is run, the results shown next are obtained.

```
                    DATA FOR MONTH 6

SALESPEOPLE     DISTRICT        SALES

JILL TANDY          3           3000
DON DEEDS           1           6000
RICK HILL           2           1900
SAM SPADE           1           1200
NICK DANGER         3           1400
JOE JONES           1           2500
HARRY WHITE         3           900
BARB SMITH          2           2200
H.J. STEED          2           1700
```

```
                RESULTS FOR MONTH 6

                DISTRICT 1 RESULTS
DON DEEDS                              6000
JOE JONES                              2500
SAM SPADE                              1200
                DISTRICT 2 RESULTS
BARB SMITH                             2200
RICK HILL                              1900
H.J. STEED                             1700
                DISTRICT 3 RESULTS
JILL TANDY                             3000
NICK DANGER                            1400
HARRY WHITE                            900
```

Sorting String Data

The sorting process for string data follows that for numeric data except that ordering is lexical rather than numeric.

Syntax:

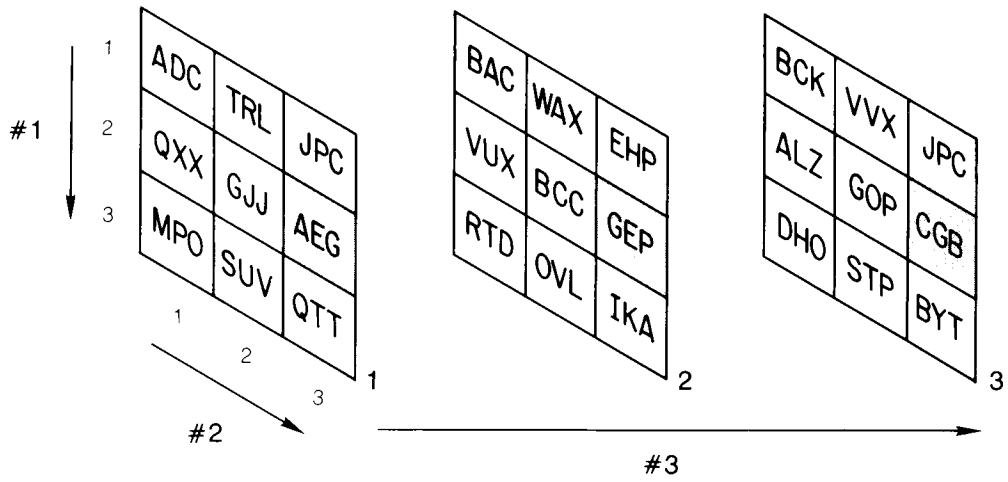
MAT SORT string source array (key specifier) [TO pointer array]

The source array in this case contains string data. The key specifier specifies which locations are designated as keys. The optional pointer array is a one-dimensional numeric array which contains the order in which the records should be sorted. As with numeric arrays, the source array is not actually rearranged when a pointer array is used.

String sorting uses the LEX function on the string data to perform the ordering (see Chapter 3). The default order is lexical (ascending), but DES can be used to specify reverse lexical (descending) order. The order can also be determined by a user-defined table as described in the LEXICAL ORDER IS section.

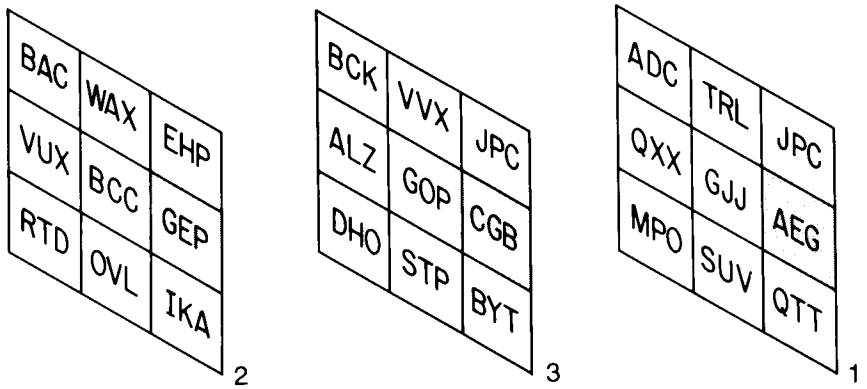
For example, assume array A$ is dimensioned to be a three-dimensional string array containing three characters per element (DIM A$(3,3,3)[3]). Assume that the middle location in the right-hand column of each record (plane) is designated as a key. If the records are to be sorted along the third dimension in reverse lexical order, the correct sorting statement is:

```
MAT SORT A$(2,3,*) DES
```



ARRAY A$ WITH KEYS

This illustration represents the source array. The sorting process sorts the records according to the ASCII values of the data in the key locations. The sorting process begins comparing key characters until dissimilar characters are found. Since the first characters of all the data in the keys are different, sorting can be performed without comparing further characters. When the sort is complete, the array is rearranged as shown.

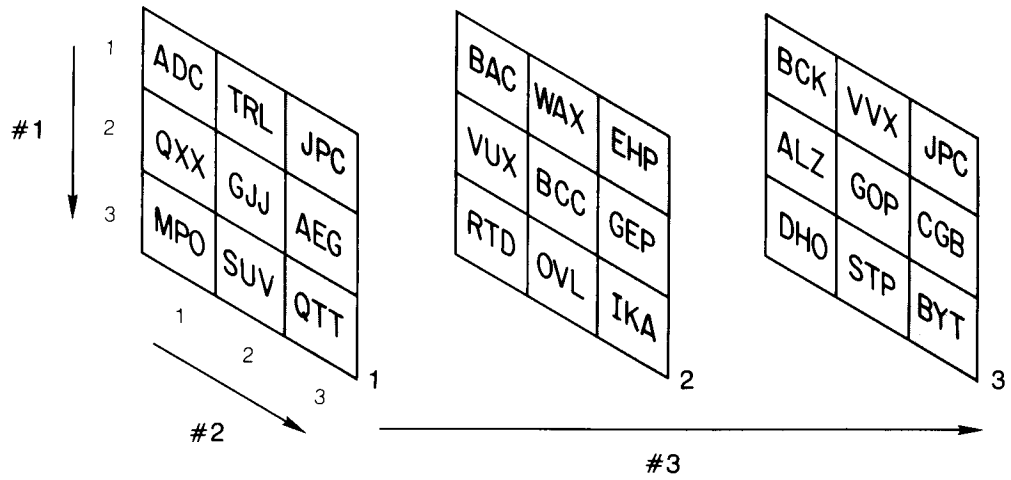


Note that the string data within each memory location has not been rearranged. The records, however, have been rearranged to reflect the new order of the keys of string data. It is important to note that strings of unequal length present no problem. Any unused dimensioned character spaces are filled with the null string and are sorted accordingly. (Example: AB precedes ABC.)

As with numeric sorting, a secondary key specifier can be used to order records which contain identical keys. In the previous array example, assume that the characters in the upper right locations of the records are designated as keys and that the records are to be sorted along the third dimension in lexical order. The sorting process begins comparing characters until dissimilar characters are found. After comparing all the characters in the primary keys of records one and three, the sort recognizes identical data. The sort process could then utilize a secondary key specifier to perform the sort. (All characters must match in order to be considered identical data).

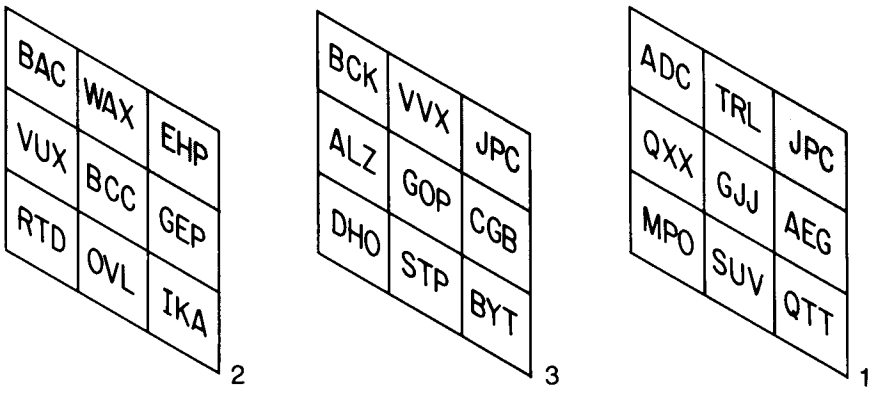
Assume that the data in the lower right locations is described by a secondary key specifier. The correct sorting statement is:

```
MAT SORT A$(1,3,*),(3,3,*)
```



Comparing the data described by the secondary key specifier, the sorting process would order record three before record one since the letter "B" precedes the letter "Q" in the STANDARD lexical order.

Upon execution of the sort statement, the array A$ is rearranged as shown.



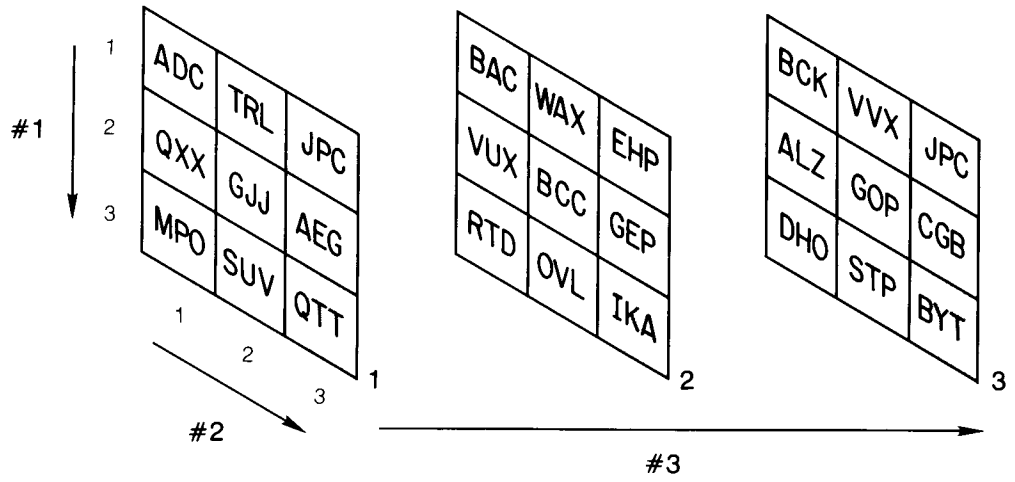
ARRAY A$ AFTER SORTING

When sorting string data, a substring specifier can be used to describe partial key specifier strings.

Syntax:

MAT SORT string source array〈key specifier〉 [[substring specifier]]

The optional substring specifier indicates which portion of the data string in the specified keys is used to order the records. Referring to the previous example, assume the second and third characters in the strings located in the lower right memory locations are designated as keys.



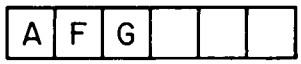
A$ KEYS WITH SUBSTRING SPECIFIER

The correct sorting statement is

```
MAT SORT A$(3,3,*) [2,3]
```

where [2,3] specifies that portion of each key string which begins at the second character and ends with the third. In this example, the sorting process begins sorting the records according to the values of the second and third characters in each key string. Sorting can be completed at the second characters since none of them are identical. After execution of the MAT SORT statement, the rearranged record order is 2-1-3.

A substring specifier must lie within the dimensioned length of the key string. The substring specifier can, however, describe an unused portion of the key string. For example, assume the following illustration represents the contents of the dimensioned length of a string.



A substring specifier such as [4;3] describes the unused portion of this string. For sorting purposes, the unused portion of this string is the null string. The null string occurs first in any lexical order.

Substring specifiers can be used with primary and secondary key specifiers simultaneously, as can descending specifiers. Example:

```
MAT SORT A$(1,*,3) [4;4] DES,(4,*,6) [3;3] DES
```

When used with multiple key specifiers, substring specifiers need not be of equal length.

A pointer array can also be included:

```
MAT SORT A$(1,*,3) [4;4] DES,(4,*,6) [3;3] DES TO B
```

Note that no punctuation is included between a key specifier and its related specifiers. A comma is included, however, between complete individual specifier entries.

The program shown next demonstrates string sorting. A list of names and phone numbers is read into a string array and then sorted according to last name.

```
10   ! ***************************************************************
20   ! ***        This program demonstrates the use of             ***
30   ! ***              the MAT SORT statement.                    ***
40   ! ***************************************************************
50                                   !
60     OPTION BASE 1                 ! Select OPTION BASE.
70     DIM Phones$(7,2)[20]          ! Dimension source array.
80                                   !
90       DATA RICHARD  HILL,818-8086
100      DATA MICHAEL  HILL,916-6502
110      DATA BARB     SMITH,917-6802  ! Name and phone
120      DATA HARRY    RULE,818-8048   ! number data.
130      DATA JULIE    TANDY,818-8080
140      DATA SAMUAL   SPADE,818-6809
150      DATA MICHAEL  SMITH,916-6800
160                                  !
170      MAT READ Phones$            ! Read data into source array.
180                                  !
190      MAT SORT Phones$(*,1)[10],(*,1)[1,9]  ! Sort data by last name;
200                                  ! use first name as secondary key.
210                                  !
220  ! ***************************************************************
230  ! ***        Print heading and sorted directory.              ***
240  ! ***************************************************************
250                                  !
260      PRINT TAB(4);"THE SORTED DIRECTORY:";LIN(1)  ! Print heading.
270        FOR I=1 TO 7
280          PRINT Phones$(I,1)[10];",";Phones$(I,1)[1,9];TAB(20);
290          PRINT Phones$(I,2)      ! Print the sorted directory.
300        NEXT I
310    END
```

Notice that the data are sorted by last names using the first names as a secondary key. If the program is run, the results shown next are obtained.

```
          THE SORTED DIRECTORY:

HILL,MICHAEL        916-6502
HILL,RICHARD        818-8086
RULE,HARRY          818-8048
SMITH,BARB          917-6802
SMITH,MICHAEL       916-6800
SPADE,SAMUAL        818-6809
TANDY,JULIE         818-8080
```

MAT REORDER Statement

The MAT REORDER statement is used to order an array according to the contents of an existing pointer array.

Syntax:

MAT REORDER object array BY pointer array [, dimension specifier]

The object array in this syntax is rearranged in the order specified by the contents of the pointer array. The maximum allowable number of dimensions of the object array is six. The optional dimension specifier selects the dimension of the object array along which records are ordered. The dimension specifier is a number from 1 to 6 or an expression which represents this number. If it is not specified, a value of 1 is assumed by the computer. The dimension specifier describes the object array dimensions in the manner shown next.

```
DIM Object_array(A,B,..., F)
                  ↑ ↑     ↑
Dimension Specifier: 1  2     6
```

For example, assume that the array Point is a pointer array containing a range of values determined by a previous sorting process as shown.

4	2	1	3

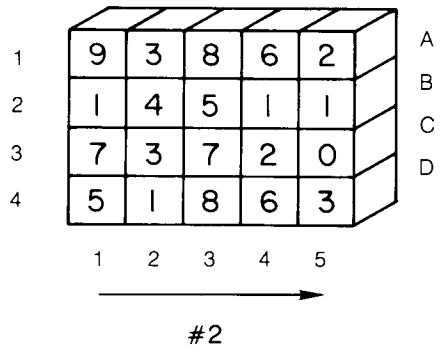
It is dimensioned as follows:

```
OPTION BASE 1
DIM Point (4)
```

Also, assume that array Object is a two-dimensional object array which contains numeric data and is dimensioned as shown.

```
OPTION BASE 1
DIM Object (4,5)
```

The following illustration represents array Object.

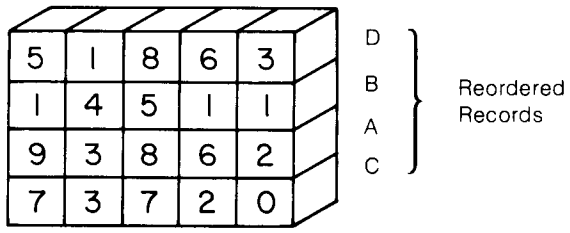


ARRAY Object

If array Object is to be rearranged along the first dimension according to the contents of array Point, the correct reordering statement is

```
MAT REORDER Object BY Point,1
```

where Object is the object array, Point is the pointer array, and 1 specifies that the object array is reordered along its first dimension (i.e., each row is considered a record). Upon execution of the reorder statement, array Object is rearranged as shown next.



ARRAY Object AFTER REORDERING

The records are rearranged in the order specified by the pointer array. Note that they are not necessarily rearranged in ascending or descending order. There is no necessary relationship among the records. They are merely rearranged according to the pointer array.

The pointer array must be dimensioned the same size as the dimension of the object array along which reordering is performed. In the previous example, the pointer array Point is dimensioned four elements in length as is the first dimension of array Object as shown next.

```
DIM Object(4,5), Point(4)
```

NOTE

If the pointer array contains duplicate numbers, unpredictable results occur. Also, if the pointer array contains numbers which are out of range for the specified dimension of the object array, an error message results when reordering is attempted.

As you may recall, a MAT SORT statement which contains a pointer array does not rearrange the source array upon execution. It merely fills the pointer array with the order of the records as if they were rearranged. The MAT REORDER statement can be used to rearrange that source array at a later time. For example,

```
MAT SORT A(1,*,3) TO B
```

fills the pointer array B with the proper sorted order of records, but it does not actually rearrange the source array A. In order to rearrange the source array, a MAT REORDER statement can be used. Example:

```
100 MAT SORT A(1,*,3) TO B
    .
    .
200 MAT REORDER A BY B, 2
```

The execution of line 100 fills the pointer array but does not rearrange the source array A. Line 200, however, does rearrange the source array according to the order defined earlier by the MAT SORT statement.

A program is now presented that demonstrates a possible use of the MAT SORT and MAT REORDER statements. Original$ is a one-dimensional array list of names and grade point averages.

The program fills the array list and sorts the data both by name and grade point average.

```
10  ! ************************************************************
20  ! ***          This program demonstrates the use           ***
30  ! ***              of the MAT SORT statement.               ***
40  ! ************************************************************
50                                        !
60      OPTION BASE 1                     ! Select OPTION BASE.
70      DIM Original$(6)[21],B(6),C(6),D(6) ! Dimension source
80                                        ! and pointer arrays.
90        MAT READ Original$              ! Read data into source array;
100                                       ! data in Lines 330-380.
101                                       !
110     PRINT "THE ORIGINAL SEQUENCE IS:",LIN(1)
120     PRINT Original$(*)                ! Print the unsorted list.
130                                       !
140     MAT SORT Original$(*)[1,16] TO B  ! Sort list by last names.
150     MAT SORT Original$(*)[18,21] DES,(*)[1,16] TO D ! Sort list by
160                                       ! grade points; use names
170                                       ! as secondary keys.
180       FOR I=1 TO 6
190         C(B(I))=I                     ! Compute the inverse permutation.
200       NEXT I
210                                       !
220     MAT REORDER Original$ BY B        ! Reorder list by name.
230     PRINT "REORDERED BY NAME:",LIN(1)
240     PRINT Original$(*)                ! Print the reordered list.
250                                       !
260     MAT REORDER C BY D                ! Combine permutations C and D
270                                       ! into C.
280     MAT REORDER Original$ BY C        ! Reorder list by grade point
290                                       ! using the combined permutation.
300     PRINT "REORDERED BY GPA:",LIN(1)
310     PRINT Original$(*)                ! Print reordered list.
320                                       !
330     DATA SMITH BILL H      3.75
340     DATA JONES BOB R       2.00
350     DATA BROWN MARY A      2.00
360     DATA SMITH GLEN C      2.90
370     DATA TAYLOR RALPH E    3.50
380     DATA JONES BONNIE R    3.50
390                                       !
400   END
```

Notice that the inverse permutation of pointer array B is computed. The use of this permutation allows two sorts to be performed on the original object array thereby eliminating the need for a duplicate array. If the program is run, the results shown next are obtained.

```
THE ORIGINAL SEQUENCE IS:

SMITH BILL H      3.75
JONES BOB R       2.00
BROWN MARY A      2.00
SMITH GLEN C      2.90
TAYLOR RALPH E    3.50
JONES BONNIE R    3.50


REORDERED BY NAME:

BROWN MARY A      2.00
JONES BOB R       2.00
JONES BONNIE R    3.50
SMITH BILL H      3.75
SMITH GLEN C      2.90
TAYLOR RALPH E    3.50


REORDERED BY GPA:

SMITH BILL H      3.75
JONES BONNIE R    3.50
TAYLOR RALPH E    3.50
SMITH GLEN C      2.90
BROWN MARY A      2.00
JONES BOB R       2.00
```

It is important to note that several arrays can be reordered by the same pointer array. The program shown next utilizes this feature. Several "parallel" arrays are reordered according to the contents of a single pointer array.

```
10   ! ********************************************************************
20   ! ***          This program demonstrates the use                   ***
30   ! ***            of the MAT REORDER statement.                     ***
40   ! ********************************************************************
50                                    !
60       OPTION BASE 1                !Select OPTION BASE.
70       DIM Id_no(4),Jobs$(4),Clock_no(4),Order(4) ! Dimension source and
80                                    ! pointer arrays.
90        MAT READ Id_no              ! Read appropriate data into arrays.
100       MAT READ Jobs$              ! Data is in Lines 230,240,250.
110       MAT READ Clock_no
120                                   !
130  ! ********************************************************************
140  ! ***       Print the heading and reorder the arrays.              ***
150  ! ********************************************************************
160                                   !
170  PRINT TAB(5);"ID NUMBER";TAB(20);"JOB";TAB(30);"CLOCK NUMBER";LIN(2);
180  CALL Print(Id_no(*),Jobs$(*),Clock_no(*)) ! Call the Print subroutine
190                                   ! using the array data as parameters.
200       PRINT LIN(1)
210
```

```
220       MAT SORT Id_no(*) TO Order ! Sort ID numbers to pointer array.
230                                  !
240       MAT REORDER Jobs$ BY Order
250       MAT REORDER Id_no BY Order ! Reorder all arrays by the same pointer.
260       MAT REORDER Clock_no BY Order
270                                  !
280       PRINT "     ---------------------------------------- ";LIN(1)
290       CALL Print(Id_no(*),Jobs$(*),Clock_no(*)) ! Call subroutine
300                                  ! for printing the reordered arrays.
310       DATA 2325,4157,9020,1025
320       DATA MANAGER,GUARD,TYPIST,NURSE ! Data is entered in the proper order
330       DATA 307,118,476,534       ! to create "parallel" arrays.
340                                  !
350     END
360   !
370   ! ********************************************************************
380   ! ***   This subroutine prints data in the specified order.    ***
390   ! ********************************************************************
400                                  !
410     SUB Print(Id_no(*),Jobs$(*),Clock_no(*))
420       FOR I=1 TO 4
430         PRINT TAB(6);Id_no(I);TAB(19);Jobs$(I);TAB(33);Clock_no(I);
440       NEXT I
450     SUBEND
```

Notice that the individual object arrays are reordered by the same pointer array. If the program is run, the results shown next are obtained.

```
ID NUMBER      JOB          CLOCK NUMBER

  2325         MANAGER        307
  4157         GUARD          118
  9020         TYPIST         476
  1025         NURSE          534

 ----------------------------------------

  1025         NURSE          534
  2325         MANAGER        307
  4157         GUARD          118
  9020         TYPIST         476
```

MAT SEARCH Statement

Searching Numeric Arrays

The purpose of the MAT SEARCH statement is to provide information about user-defined conditions within an array. This information is returned to a variable for recall and examination.

Syntax:

MAT SEARCH source array(location specifier), condition; variable
[, starting address]

The source array represents the array in which searching is performed. The location specifier defines the locations within the souce array which are searched. No location specifier need be included if the source array is one-dimensional. The condition is that value or location for which you are searching. When the condition is satisfied, the memory content or location which satisfies it is returned to the variable. The optional starting address specifier allows you to select a location within the range defined by the location specifier at which you want searching to begin. Numeric searching comparisons are performed in the INTEGER mode for integer-precision arrays and in the REAL mode for short or real-precision arrays.

The conditions available in the MAT SEARCH process are entered in the following form:

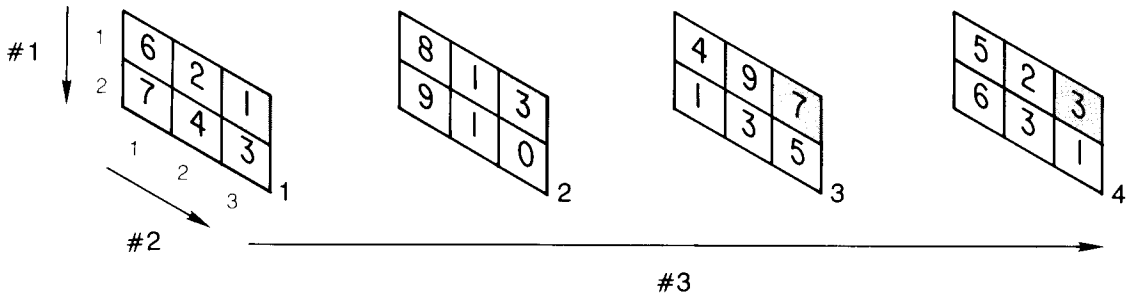
LOC(relational operator – expression)
#LOC(relational operator – expression)
MAX
MIN

The list of relational operators includes: >,<,=,#,>=,<=,<>. For an integer search, the expression is rounded before the comparison is performed. A LOC condition causes the search process to scan the specified locations until it finds the first value which satisfies the condition. The default value for a relational operator is =.

A #LOC condition causes the search process to scan the specified locations and return the total number of locations whose contents satisfy the condition.

MAX and MIN conditions do not include an argument in parentheses because they automatically cause the search process to scan all specified locations to find the maximum and minimum values. To return the result of a MAX or MIN search, a string variable is used with string data, and a numeric variable with numeric data.

To demonstrate the MAT SEARCH process, assume that array A is a three dimensional numeric array containing random numbers as shown below.



ARRAY A WITH SEARCH LOCATIONS

Assume that the shaded locations are to be searched until one is found which contains a value greater than 5. Let B represent the variable to which the result is returned.

The shaded memory locations must be described by a location specifier. Since they lie along the third dimension, the records in which they lie are defined accordingly. The correct location specifier is:

```
(1,3,*)
```

Note that the location specifier is written in the same format as a key specifier for a MAT SORT statement. The subscripts are written in the same order as the array dimensions (the numbered arrows). The first two subscripts define the upper right location, and the asterisk indicates that the third subscript is varied over its range of values to describe all three records.

The correct search statement for this example is

```
MAT SEARCH A(1,3,*),LOC(>5);B
```

where A is the source array, (1,3,*) defines the locations to be searched, LOC(>5) is the condition, and B is the variable to which the result is returned.

After execution of the search statement, the number 3 is returned to the variable B. This is the first record whose specified location satisfies the condition(i.e., it contains a value greater than 5). Searching begins at the location in that record described by the smallest number in the range of varied subscripts and continues to the largest.

If a condition is not satisfied upon completion of the searching process, a value one greater than the upper limit of the varied subscript is returned to the numeric variable. For example, if the specified locations in the previous example are searched for a value greater than 8, none is found. Therefore, a value of 5 representing a number one greater than the record containing the last searched location is returned to B. Note that if the upper limit of the varied subscript is 32 767, the value returned by an unsuccessful search is − 32 768.

The location specifier initially defines the range of locations to be searched. If you do not wish to search the entire range, a starting address specifier can be used to designate where the search is to begin. In the previous example, assume that the search process is to scan only the last three records. The correct search statement is

```
MAT SEARCH A(1,3,*),LOC(>5);B,2
```

where the number 2 directs the search to begin at the specified location in the second record and proceed to the last record. Upon execution, the number 3 indicating the third record is returned to the variable B because the content of its specified location is the first to satisfy the condition.

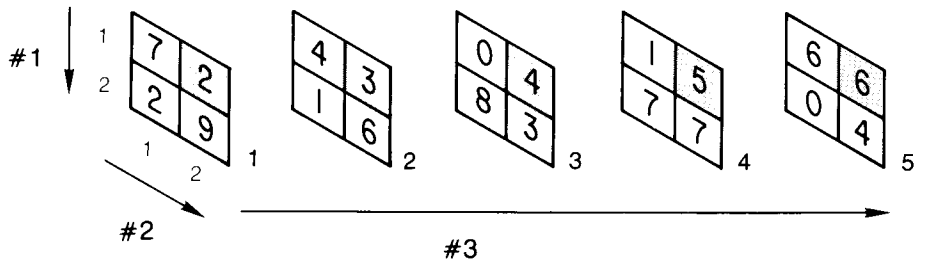
The following program is included to demonstrate a search for maximum and minimum values and number of occurrences. The array Numbers is filled with random numbers from which the maximum and minimum values are selected.

```
MAXIMUM = 9          MINIMUM = 1
```

If this program is run, the following results are obtained.

```
10   ! ***************************************************************
20   ! ***       This program demonstrates the use of the          ***
30   ! ***       MAT SEARCH statement to find maximum and           ***
40   ! ***             minimum values within an array.             ***
50   ! ***************************************************************
60                                      !
70     OPTION BASE 1                    ! Select the OPTION BASE.
80     DIM Numbers(11)                  ! Dimension source array.
90                                      !
100         FOR I=1 TO 11               ! Establish input loop.
110           DISP "NUMBER";I;          ! Display entry number.
120           INPUT Numbers(I)          ! Input data to source array.
130         NEXT I                      ! Loop to input next data item.
140                                     !
150  ! ***************************************************************
160  ! *** The data used in this example: 6,1,9,2,8,3,8,9,1,7,5 ***
170  ! ***************************************************************
180                                     !
190       MAT SEARCH Numbers(*),MAX;Max  ! Search for maximum value.
200       MAT SEARCH Numbers(*),MIN;Min  ! Search for minimum value.
210                                     !
220       PRINT "MAXIMUM =";Max,"MINIMUM =";Min
230     END
```

Normally, a LOC search ends at the first location which satisfies the specified condition. However, additional satisfactory values may exist beyond that location. By setting the starting address to be one greater than the content of the variable, a search can be continued from the first location which satisfied the LOC condition. This automatically continues a search from where a previous search left off. All satisfactory values can be obtained in this way. As an example, assume that array B is a three dimensional numeric array containing random numbers as shown.



ARRAY B

Assume that the shaded memory locations are to be searched for values greater than 2. A single MAT SEARCH scan would stop at the second record since it is the first one encountered whose memory location satisfies the condition.

However, by constructing a loop and using the proper starting address specifier, the search can be made to continue through the range of specified locations. The following program demonstrates this feature.

```
10  ! ************************************************************
20  ! ***       This program demonstrates the use              ***
30  ! ***          of the MAT SEARCH statement.                 ***
40  ! ************************************************************
50                                  !
60      OPTION BASE 1               ! Select OPTION BASE.
70      DIM B(2,2,5)                ! Dimension numeric array B.
80                                  !
90        MAT READ B                ! Read data into array B. Data is
100                                 ! found in Line 240.
110       C=0                       ! Initialize C so first search begins
95                                  ! at record 1 of array B.
120                                 !
```

```
130 ! ********************************************************
140 ! ***  Search for locations of numbers greater than 2.  ***
150 ! ********************************************************
160                                 !
170 Search: MAT SEARCH B(1,2,*),LOC(>2);C,C+1
180                                 !
190       IF C+1>6 THEN 260         ! Test: Have all locations been searched?
170                                 !       If yes, stop.
200                                 !
210       DISP C;                   ! Display most recent search results.
220       WAIT 500                  ! Wait slightly before further display.
230       IF C<5 THEN Search        ! Test: Has search reached last location?
220                                 !       If not, continue.
240                                 !
250       DATA 7,4,0,1,6,2,3,4,5,6,2,1,8,7,0,9,6,3,0,4
260     END
```

The variable C is initially set to zero. Line 170 begins the search routine. Since C was set to zero, the starting address specifier (C + 1) directs the search to begin at record one. Line 190 tests to see if the specified locations have all been searched. (Remember, if all locations have been searched, a number one greater than the last record searched is returned to the variable. In this case, that number is six). If all locations have not yet been searched, line 210 displays the contents of C (it now contains a satisfactory value). If C contains a value less than 5, the search is not finished and line 230 directs the program to search again. Due to the nature of the starting address specifier, the search begins this time at the next memory location beyond that which satisfied the condition previously. In other words, the search resumes where it left off. If you run this program, the following should be displayed:

```
 2  3  4  5
```

Searching String Arrays

Searching string arrays follows the same format as searching numeric arrays.

Syntax:

MAT SEARCH source array (location specifier), condition; variable
[, starting address]

The source array, location specifier, condition, variable, and starting address specifier serve the same functions for string arrays as for numeric arrays. If a relational operator is included, the LEXICAL ORDER IS statement determines the ordering of the string data. The order can be STANDARD (according to the ASCII table) or user-defined (see Appendix A).

For example, assume array List$ contains a list of names and dollar amounts. The program shown next inputs the data to the source array. It then searches for a particular name and outputs the corresponding dollar amount.

```
10   ! ************************************************************
20   ! ***         This program demonstrates the use of          ***
30   ! ***              the MAT SEARCH statement.                 ***
40   ! ************************************************************
50                                !
60     OPTION BASE 1              ! Select OPTION BASE.
70     DIM List$(4)[20]           ! Dimension source array.
80                                !
90       FOR I=1 TO 4             ! Establish loop for data entry.
100        READ List$(I)          ! Read data into List$.
110      NEXT I                   ! Read next data item.
120                               !
130      MAT PRINT List$          ! Output the original list.
140                               !
150  ! ************************************************************
160  ! ***          Search List$ for a particular name.          ***
170  ! ************************************************************
180                               !
190      MAT SEARCH List$(*)[1,5],LOC("HAYS ");B   ! Search the proper
200                                                ! portion of each
210                                                ! string.
220                               !
230      PRINT List$(B)[1,5];":  ";List$(B)[15,18] ! Output specified
240                                                ! name and dollar
250                                                ! amount
260                               !
270      DATA BROWN  FRANK  $100  ,SMITH  BOB    $75
280      DATA HAYS   SUE    $150  ,JONES  ED     $200
290    END
```

In this program a MAT SEARCH is used to find the string which contains the required name. Once that string is found, the portion of it containing the dollar amount is displayed. Note that the substring specifier is used in the search and display statements. If you run this program, the following results are obtained.

```
$150

BROWN  FRANK  $100
SMITH  BOB    $75
HAYS   SUE    $150
JONES  ED     $200

HAYS :  $150
```

MAX and MIN values can also be obtained from a string search as demonstrated by the program shown next.

```
10   ! ************************************************************
20   ! ***  This program demonstrates use of the MAT SEARCH   ***
30   ! ***        statement in conjunction with strings.       ***
40   ! ************************************************************
50                                          !
60      OPTION BASE 1                       ! Select the OPTION BASE.
70      DIM Stringsearch$(3)[3]             ! Dimension the string array.
80                                          !
90        MAT READ Stringsearch$            ! Read data into the array.
100         DATA "ABC","BCD","DEF"
110                                         !
120       MAT SEARCH Stringsearch$(*),MAX;A$
130                                         ! Search string array for
140                                         ! maximum string value.
150                                         !
160       MAT SEARCH Stringsearch$(*),MIN;B$
170                                         ! Search string array for
180                                         ! minimum string value.
190                                         !
200       PRINT "MAXIMUM = ";A$,"MINIMUM = ";B$
210                                         ! Output the results.
220     END
```

The array Stringsearch$ is filled with string data. MAT SEARCH statements are then used to find the maximum and minimum values of the data. Note that the search statements use the current lexical order of the data to perform the search comparisons (see LEXICAL ORDER IS statement). If this program is run, the results shown next are obtained.

```
MAXIMUM = DEF        MINIMUM = ABC
```

Chapter 3
Extended Character Sets

Statement Syntax

LEXICAL ORDER IS lexical order designator

lexical order designator: an integer array containing a local language collating table or the word **STANDARD**

LEX (string expression 1, string expression 2)

Dot Matrix – All items in dot matrix must be entered as shown.

[ ] – All items in brackets are optional.

... – Three dots indicate that successive parameters are allowed, when each is separated by a comma.

Introduction

The statements described in this chapter are used as programming aids to define and access comparison and case functions on the ASCII and Roman Extension Character Sets. Each of these statements or functions can be executed from within a program or from the keyboard. Examples of each statement are included to demonstrate its use.

LEXICAL ORDER IS Statement

The LEXICAL ORDER IS statement allows you to select the collating sequence for string sorts and determine the results of string comparisons. It also determines the results of UPC$ and LWC$ transformations for Roman Extension characters.

Syntax:

```
LEXICAL ORDER IS lexical order designator
```

The lexical order designator must be an integer array containing a local language collating table, or the word STANDARD which indicates the standard ASCII character sequence. The following local language collating tables are available on the Lexical Tables Cartridge for use with the Advanced Programming ROM.

FRENCH	SWEDSH (Swedish)
GERMAN	SPANSH (Spanish)

Refer to Appendix A for information on the lexical order collating tables.

The STANDARD lexical order designator selects the ASCII table as the collating sequence (i.e., string sorting is performed according to ASCII values). STANDARD is the default lexical order designator at power ON and after SCRATCH A. As such, you need not include a LEXICAL ORDER IS statement if you wish to use ASCII as the collating sequence. RESET does not clear the current lexical order. However, if you wish to change to STANDARD from some other collating sequence, you can do so by entering:

```
LEXICAL ORDER IS STANDARD
```

The program shown next demonstrates how a local language collating sequence is selected.

```
10  INTEGER A(1:400)
20  ASSIGN #1 TO "FRENCH"
30  MAT READ #1;A
40  LEXICAL ORDER IS A(*)
```

Line 10 dimensions an integer array to hold the specified collating table. Here, A is chosen as the array and is dimensioned to a length of 400 which accommodates any of the local language character collating tables listed previously.

Line 20 assigns #1 to the specified file on the cartridge; here, FRENCH.

Line 30 reads the contents of the file (the specified collating table) into the array.

Finally, line 40 establishes the contents of the array (which now contains the collating table) as the lexical order.

Note that when a LEXICAL ORDER IS statement is executed, the information in the array is transferred into an internal buffer. This allows you to use the array for something else. Care must be taken, however, to insure that enough memory exists for the buffer when the statement is executed. Otherwise, an error occurs.

The LEX Function

The LEX function allows you to compare two strings and determine which precedes the other according to the current lexical order. The LEX function can be executed from the keyboard or from within a program.

Syntax:

LEX(string expression 1, string expression 2)

The LEX function begins comparing string characters until a difference is found or until one or both strings terminate. The function returns the number −1 if string 1 precedes string 2. If the current lexical order cannot distinguish between the two strings, 0 is returned. The number 1 is returned if string 2 precedes string 1.

The program shown next demonstrates the LEX function. Two strings are compared and the result is displayed.

```
10  A=LEX("%EF","%BC")
20  DISP A
30  END
```

Notice that the program utilizes the default STANDARD lexical order. The LEX function compares the strings up to their second characters since these are the first respective characters that differ. Upon execution, the number 1 is displayed because B (string 2) precedes E (string 1) according to the current lexical order (ASCII).

The previous example program compared strings of equal length. The example program shown next demonstrates a comparison of strings of unequal length.

```
10  A=LEX("A","A ")
20  B=LEX("A","AB")
30  DISP A,B
40  END
```

In this case, both A and B are −1. In both comparisons, the first string terminates before the second.

Uppercase and Lowercase Functions

The Advanced Programming ROM extends the capabilities of the mainframe uppercase (UPC$) and lowercase (LWC$) functions to correctly handle the upper and lowercase transformations of the various local language characters.

The results of the UPC$ and LWC$ functions are determined by the entries in the upper and lowercase sections of the lexical order array for Roman Extension characters (refer to the Reference Tables). The upper and lower cases of characters in the main ASCII set are fixed.

Descriptions of the French, German, Spanish and Swedish upper and lowercase transformations which are provided in the respective files on the cartridge are explained in Appendix A.

The UPC$ function can be used to obtain a proper LEX comparison if it is not known if the two strings are in the same case (i.e., if "ABC" should equal "abc"). Example:

```
X = LEX (UPC$ (A$),UPC$ (B$) )
```

NOTE

If a program containing UPC$ or LWC$ statements is STORE'd in your computer when an Advanced Programming ROM is installed, and is then LOAD'ed into your computer when an Advanced Programming ROM is not installed, a MISSING ROM error message occurs.

If a program is STORE'd in your computer when no Advanced Programming ROM is installed, and is then LOAD'ed into your computer when an Advanced Programming ROM is installed, no error message results. However, if any lexical statements are added, the current UPC$/LWC$ lines must be re-STORE'd in order for the Advanced Programming ROM to affect them.

It is suggested that you SAVE rather than STORE programs which will be switched between those computers which have an Advanced Programming ROM and those which do not.

Chapter 4
File Catalog Access with the HP 9835

Introduction

The CAT TO statement described in this chapter is provided by the Advanced Programming ROM for the HP 9835; it is a mainframe statement in the HP 9845B/C. CAT TO allows your program to read mass storage catalogs. This statement can be executed from the keyboard or from within a program. Examples are given to explain the statement and its parameters.

CAT TO Statement

The CAT TO statement is used to write a mass storage catalog into a one dimensional string array. This allows your program to have immediate access to mass storage catalogs. This feature allows you to copy files from one medium to another under program control.

Syntax:

CAT TO string array (*) [,skip count [,return variable]] [;"selective catalog specifier/msus"[,heading specifier]]

The catalog entries are read into the string array. The elements of the string array must be dimensioned to be at least 41 characters to accommodate a catalog file entry.

The skip count parameter allows you to skip the specified number of catalog file entries before you begin recording them in the string array. The default skip count is 0.

The return variable indicates whether the string array is filled before the specified catalog entries are exhausted. Upon execution, 0 is returned to the variable if the number of catalog entries to be recorded is equal to or less than the number of array strings. Any strings not filled with catalog entries are filled with the null string. If there are more catalog entries to be entered than there are array strings, the position in the catalog of the last recorded entry is returned to the variable. If you do not include a return variable, this information is lost.

The selective catalog specifier consists of a string expression or a group of from one to six ASCII characters (excluding a colon) which identify particular file entries to be recorded. Spaces within these characters are deleted. Only those file entries whose names begin with the (non-blank) characters of the specifier are recorded in the string array. If no selective catalog specifier is included, the CAT TO process attempts to record all catalog entries. If a selective catalog specifier is given, the skip count and return variable refer to the selective catalog entries.

The mass storage unit specifier (msus) indicates the mass storage unit in which the catalog is located. Typically, this specifier is used to select a mass storage unit which is different from the one selected by a MASS STORAGE IS statement.

The heading specifier is a numeric expression which, when its value rounds to anything other than 1, causes the second line of the catalog (the heading) to be recorded in the first array element (string). The heading indicates the mass storage unit and the number of available tracks. The default value for the heading specifier is 1. (Note that this default is opposite that of the CAT statement.)

Note that within the CAT TO syntax there are two groups of optional parameters which are separated by a semi-colon. Within either group the first parameter must be specified if you wish to use the second. The groups themselves, however, can be included independently.

The following example program demonstrates the CAT TO statement. Assume that the data shown next represent the catalog of mass storage unit T15 (tape drive).

```
NAME   PRO TYPE REC/FILE  BYTES/REC   ADDRESS
T15          2

GPAS       DATA       7        256         5
SERCH      DATA       4        256        12
DATABS     DATA       7        256        16
TEMPS      DATA       6        256        23
DOLARS     DATA       4        256        29
MAXMIN     DATA       4        256        33
MTPRT1     DATA       2        256        37
MTPRT2     DATA       2        256        39
LEXPRO     DATA       1        256        41
LXFUNC     DATA       1        256        42
```

The following program records the file entries in the string array List$. The contents of List$ are then output.

```
10   OPTION BASE 1                      ! Select option base.
20   DIM List$(12)[41]                  ! Dimension string array.
30   !
40   CAT TO List$(*)                    ! Record catalog entries in string array.
50   !
60   PRINT "STRING ARRAY ="%LIN(2)      ! Print heading.
70   MAT PRINT List$                    ! Print the string array.
80   END
```

Notice that the CAT TO statement assumes its optional default parameters since none are specified.

If the program is run, the results shown next are obtained.

```
STRING ARRAY =

GPAS        DATA        7        256          5
SERCH       DATA        4        256         12
DATABS      DATA        7        256         16
TEMPS       DATA        6        256         23
DOLARS      DATA        4        256         29
MAXMIN      DATA        4        256         33
MTPRT1      DATA        2        256         37
MTPRT2      DATA        2        256         39
LEXPRO      DATA        1        256         41
LXFUNC      DATA        1        256         42
```

Notice that the heading does not occupy the first element of the array. This is because the heading specifier assumes a default value of 1. Line one of the catalog is never recorded in the string arrray because it is the same in every catalog.

The effects of the optional parameters can be demonstrated by modifying the CAT TO statement in the original program. For example, by setting the skip count to 2, the CAT TO process begins recording entries at the third file entry instead of the first as shown next.

```
40 CAT TO List$(*),2
```

This causes the process to skip down two file entries before it begins recording them in the string array. DATABS is the first file entry in the string array when this statement is executed as shown next.

```
STRING ARRAY =


DATABS    DATA      7      256        16
TEMPS     DATA      6      256        23
DOLARS    DATA      4      256        29
MAXMIN    DATA      4      256        33
MTPRT1    DATA      2      256        37
MTPRT2    DATA      2      256        39
LEXPRO    DATA      1      256        41
LXFUNC    DATA      1      256        42
```

By including a return variable you can easily learn whether all specified catalog entries are recorded in the string array. The original program is modified to include a return variable as shown next. The string array is shortened to exclude a few catalog entries.

```
10   OPTION BASE 1                ! Select option base.
20   DIM List$(5)[41]             ! Dimension string array (too small).
30   !
40   CAT TO List$(*),0,B          ! CAT TO using skip count and return variable.
50   !
60   PRINT "B = "&B&LIN(2)        ! Print content of return variable.
70   MAT PRINT List$              ! Print the string array.
80   END
```

In this program, the skip count parameter is set to 0 so as to start recording file entries from the first one. Upon execution, the results shown next are obtained.

```
B = 5


GPAS      DATA      7      256         5
SEARCH    DATA      4      256        12
DATABS    DATA      7      256        16
TEMPS     DATA      6      256        23
DOLARS    DATA      4      256        29
```

This indicates that the fifth file entry was the last one recorded in the string array. Since the string array was shortened, it cannot accommodate all program entries. Notice that DOLARS is the last file entry in the string array.

Although the selective catalog specifier and the msus are grouped together within a parameter, they can be used separately. The selective catalog specifier is used to record selected file entries in the string array. For example, to record only those files whose names begin with the letter S, the CAT TO statement in the original program is modified as follows.

```
40 CAT TO List$(*);"S"
```

The semi-colon indicates that the S belongs to the second group of parameters. Notice that the selective catalog specifier is entered within quotes. If this CAT TO statement is executed within the original program, the following results are otained.

```
SERCH     DATA      4      256      12
```

Notice that List$ contains only that file entry whose name begins with the letter S. The selective catalog specifier used in this example could also be written as a string expression. For example, the program lines

```
30 Z$="S"
40 CAT TO List$(*);Z$
```

would yield the same results.

The msus is typically used to override the current mass storage unit default. The mass storage unit used in the original program can be changed to a disk drive by modifying the CAT TO statement as follows.

```
40 CAT TO List$(*);":F8"
```

A colon precedes the msus to differentiate it from the selective catalog specifier.

The last parameter in the CAT TO syntax is the heading specifier. The original example did not record the heading in the string array. The heading is recorded in the first element of the string array if the heading specifier rounds to anything but 1. Example:

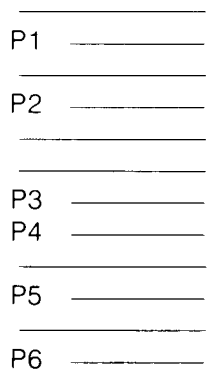
```
40 CAT TO List$(*);" ",0
```

In order to include a heading specifier, a selective catalog specifier or a msus must precede it. Here, a space between quotes (used as a selective catalog specifier) fulfills this syntax requirement without actually specifying any particular file entries. Spaces are deleted within a selective catalog specifier. A null string (two quotes without a space) does not fulfill this requirement.

The heading specifier 0 causes the heading to be recorded in the first string of the array. If this line is executed within the original program, the following results are obtained.

```
T15          2
GPAS       DATA      7      256          5
SERCH      DATA      4      256         12
DATABS     DATA      7      256         16
TEMPS      DATA      6      256         23
DOLARS     DATA      4      256         29
MAXMIN     DATA      4      256         33
MTPRT1     DATA      2      256         37
MTPRT2     DATA      2      256         39
LEXPRO     DATA      1      256         41
LXFUNC     DATA      1      256         42
```

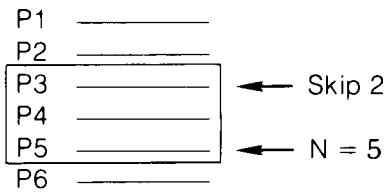
The parameters of the CAT TO statement can be used to create a "window" around a specified portion of the catalog. For example, assume that the figure shown next represents a catalog containing several file entries whose names begin with the letter P.



If the string array A$ is dimensioned three elements, then a statement such as

```
CAT TO A$(*),2,N;"P"
```

would record only three of the P file entries. The selective catalog specifier selects only the P file entries to be recorded. The skip count further limits the number of recorded P file entries by skipping two of them. Finally, the size of the string array limits the total number of file entries that can be input. The figure shown next indictes the "window" containing the recorded file entries.



Upon execution, A$ contains only file entries P3, P4, and P5. The variable N equals 5 because the fifth P file entry was the last one recorded.

As another example, notice how the CAT TO statement is used in the following program to copy files from one mass storage unit to another.

```
10   OPTION BASE 1                    ! Select option base.
20   DIM Cat$(20)[41]                 ! Dimension array to hold catalog.
30   N=0                              ! Set skip count/return variable counter to
40   !                                  zero.
50   INPUT "FROM DEVICE",Msus1$,"TO DEVICE",Msus2$   ! Enter mass storage
60   !                                                  unit specifiers.
70   !
80 Loop:  CAT TO Cat$(*),N,N;Msus1$  ! Read catalog entries into Cat$.
90     FOR I=1 TO 20                  ! Initiate copy loop: 20 entries per loop.
100      IF LEN(Cat$(I)) THEN Doio    ! Test for string: Entry available"
110      GOTO Done                    ! End if all entries exhausted.
120   !
130 Doio: IF Cat$(I)[15;1]="?" THEN Dont                    ! Test: 9835 file?
140      COPY Cat$(I)[1;6]&Msus1$ TO Cat$(I)[1;6]&Msus2$    ! Copy 9835 files.
150      PRINT Cat$(I)[1;6]&" COPIED FROM "&Msus1$&" TO "&Msus2$  ! Tell what
160   !                                                             was copied.
170      GOTO Next                                         ! Skip "Dont".
180   !
190 Dont:  PRINT "CAN'T COPY "&Cat$(I)[1;6]                 ! List non-9835 files.
200   !
210 Next:  NEXT I                  ! Do next entry.
220   !
230   IF N THEN Loop               ! Test return variable: More entries to be
240 Done:  END                     !   copied?
```

Line 50 allows you to enter the mass storage devices you are using. Line 140 contains the copy statement for files based on the string array contents from the CAT TO statement (line 80). You can easily modify this program to copy selective files which suit your needs.

Appendix A
User-Defined Lexical Order

It is recommended that you use the existing collating tables as they appear on the tape cartridge. However, an experienced programmer can create a lexical order table to accommodate his particular application. The following information describes the procedures involved. A knowledge of binary and octal numbers is necessary to utilize these procedures.

To modify a local language collating table, it is first input to an integer array. Once in the array, the specified table can be accessed for modifications.

Along with the local language tables, a special ASCII table ("ASCII") is included on the cartridge for the purpose of modifying the ASCII collating sequence. The LEXICAL ORDER IS STANDARD statement automatically uses the standard ASCII sequence. But in order to modify it, the "ASCII" file on the cartridge may be input to an integer array in the same manner as is a local language table. Example:

```
10   ! ******************************************************
20   ! ***     This program segment illustrates the     ***
30   ! ***          LEXICAL ORDER IS statement.         ***
40   ! ******************************************************
50                                  !
60     OPTION BASE 1                ! Select OPTION BASE.
70     INTEGER B(400)               ! Dimension integer array.
80       ASSIGN #1 TO "ASCII"       ! Select ASCII file.
90       MAT READ #1;B              ! Read in the ASCII table.
100      !
110      ! User modifications.
120      !
130      LEXICAL ORDER IS B(*)      ! Establish contents of
140                                 ! array B as lexical order.
150    END
```

Line 90 inputs the "ASCII" file contents to the integer array B. Lines 100 through 120 represent the user modifications to the ASCII table, and line 130 establishes the modified table as the lexical order. Notice that in order for the table in the given array to dictate lexical comparison results, a LEXICAL ORDER IS statement must be included after any modifications to the array have been made.

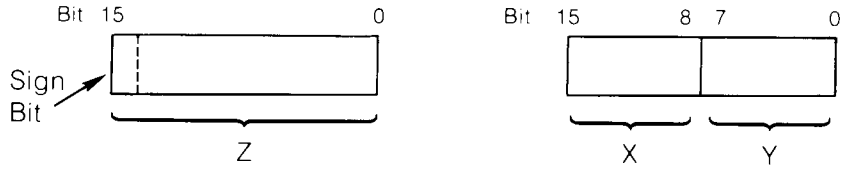
Once a LEXICAL ORDER IS statement has been executed, the integer array can be used for other purposes. Be sure, however, to save all necessary modifications on the tape cartridge before using the array for other purposes. The program lines shown next can be used to maintain a copy of the lexical table modifications for the previous example.

```
100  ASSIGN #3 TO "NEW"
110  MAT PRINT #3;B
```

The lexical order tables give you the capability to define the collating sequence (collating section), define the uppercase/lowercase transformation of Roman Extension characters (UPC/LWC section), and prescribe "special handling" of certain characters (mode section). The lexical table with its individual sections is an integer array organized as shown in the following illustration.

1	Total Length
2	Mode Section Length
3 ⋮ 258	Collating Section (Length = 256)
259 ⋮ 354	UPC/LWC Section (Length = 96)
355 ⋮ 354 + N	Mode Section (Length = N)

Each element of the lexical table consists of 16 bits. This arrangement can accommodate one number or an integer combination of two (unsigned) numbers as shown next.



If one number is stored, a binary two's complement storage format is used. In this manner, each element has the ability to contain two distinct values.

The first element of the array contains the complete length of the lexical table. This length is 354 plus the number of elements (N) needed in the user-defined mode section (see explanation of mode section).

The second element of the array contains N, the length of the mode section of the table. Once the length of the mode section is defined, the first two elements of the array table can be filled with the proper information.

Elements 3 – 258 of the table contain the actual collating sequence and the appropriate pointers into the mode section, if one exists.

Elements 259-354 define the uppercase/lowercase transformations for characters in the Roman Extension set (refer to the ASCII and Roman Extension charts in the Reference Tables).

The mode section consists of elements 355 to (354 + N) and provides facilities for handling certain special case characters.

Collating Sequences

When the lexical order is STANDARD, the ASCII codes define the collating sequence. Thus, A lexically precedes B (i.e., LEX("A", "B") is −1) because the ASCII code for A is 65, the ASCII code for B is 66, and 65 is less than 66.

For computer processing, each character has been assigned an ASCII code. The ASCII codes are fixed, however, and cannot be changed to reflect a new user-defined lexical order. The lexical order tables give you the capability of redefining the lexical order by assigning each character a sequence number. Characters are still represented internally by their ASCII codes, but a lexical comparison compares sequence numbers rather than ASCII codes. Now, LEX ("A", "B") is −1 if the sequence number of A is less than the sequence number of B. For example, the strings "ABC" and "XYZ" are stored in the computer as strings of ASCII codes:

ASCII code for A	ASCII code for B	ASCII code for C

ASCII code for X	ASCII code for Y	ASCII code for Z

The calculation of LEX ("ABC","XYZ") involves a comparison of sequence numbers:

Sequence # for A	Sequence # for X
Sequence # for B	Sequence # for Y
Sequence # for C	Sequence # for Z

By properly assigning each character a sequence number, you can define any lexical ordering of the characters.

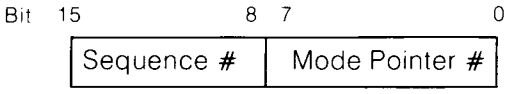
Collating Section

The collating section of the lexical order table (elements 3-258) defines the actual collating sequence by assigning a sequence number to each character. There are 256 possible 8-bit character codes (refer to the ASCII and Roman Extension charts in the Reference Tables). The user-defined sequence number for the character with ASCII code 0 goes in element 3 of the lexical table (element 1 of the collating section), sequence number for character with ASCII code 1 goes in element 4 of the lexical table (element 2 of the collating section), and so on.

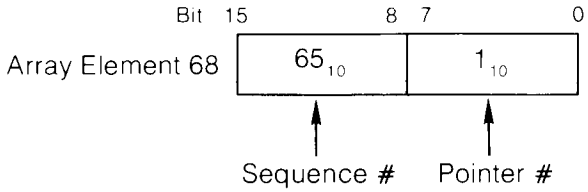
For example, the letter A has ASCII code 65_{10}. When an A is encountered, the manipulation routine goes to the 66th position of the collating section to fetch a sequence number. Note that the sequence number is fetched from the collating section element number equal to the ASCII value of the character +1. This is because the first character in the collating section is null, and its ASCII value is zero. Usually, the sequence number equals the ASCII value. However, if a different collating order is selected, the sequence number may differ from the ASCII value.

In addition to the sequence number, each entry of the collating section may contain a pointer into the mode section of the lexical table. If no pointer is specified, 0 is entered. The mode section contains instructions governing special case letters. The pointer number indicates in which position of the mode section a special case is handled.

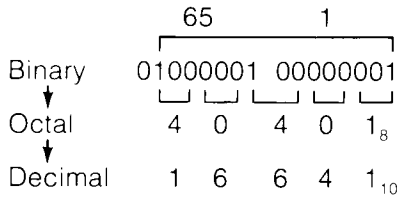
The sequence number and mode pointer number for each character are combined to form a single integer entry in the lexical table:



Assume that A is a special case letter with the sequence number 65. Also, assume that the special case is handled in position 1 of the mode section. The collating section entry containing this information is represented by:



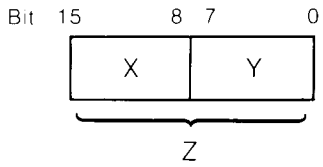
The actual content of this entry is $16\ 641_{10}$. The number is obtained as shown next.



Both the sequence number (65) and the pointer number (1) are combined to form one integer value within the table element, in this case, $16\ 641_{10}$.

An alternate method of converting the binary and octal elements of the lexical table entries to integer form is given below. For example, if the sequence number you are using is greater than 127, you may find it more convenient to use the formulas shown next.

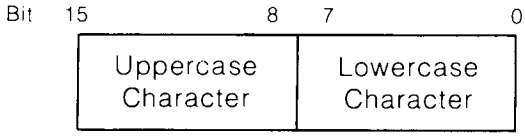
X and Y are each 8-bit unsigned integers.
Z is a 16-bit signed integer formed by concatenating the bit patterns of X and Y.



1. Given X and Y, calculate Z:
   Z = X*256 + Y − (X>127)*65 536
2. Given Z, calculate X and Y:
   X = INT(Z/256) + (Z<0)*256
   Y = Z MOD 256

Uppercase/Lowercase Section

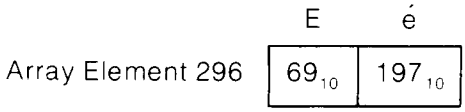
Elements 259-354 of the lexical table are used for uppercase/lowercase information for characters residing in the Roman Extension Set. Data contained in these elements is in the form:



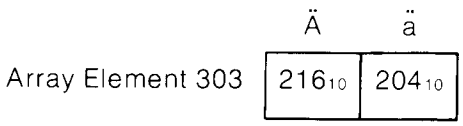
The UPC/LWC section contains an element for each character with an 8-bit code of 160 through 255. The first element of the UPC/LWC section (element 259 of the lexical table) contains the upper/lowercase for the character with 8-bit code 160, the next element contains the transformations for the character with 8-bit code 161, and so on.

The purpose of this section of the lexical table is to allow you to define the upper and lowercase versions of each letter in the Roman Extension Set for use by the UPC$ and LWC$ functions. All characters in the main ASCII table have fixed upper and lowercases, and as such, are not changed. If both the uppercase and lowercase 8-bit codes in the table are 0 for a given character, the character is unchanged by the UPC$ and LWC$ functions.

For example, if a word containing a lowercase é (code 197_{10}) is to be converted by French conventions, the accent is omitted when the letter is converted to uppercase. That is, é→E. But under German conventions, an umlaute in the letter remains so that ä→Ä. Therefore, for French, the 38th entry of the uppercase/lowercase section (296th entry of the table) is



while the 45th entry in the German uppercase/lowercase section is



Mode Section

The mode section (starting at element 355) of the lexical table handles special case characters. The three special cases shown next are possible.

- Accent Priority
- 1 For 2 Character Replacement
- 2 For 1 Character Replacement

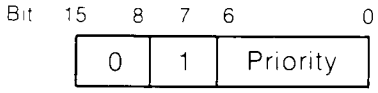
Accent Priority

You may specify the priority of letters with accent marks. Example:

$$e<\hat{e}<\ddot{e}<\acute{e}<\grave{e}$$

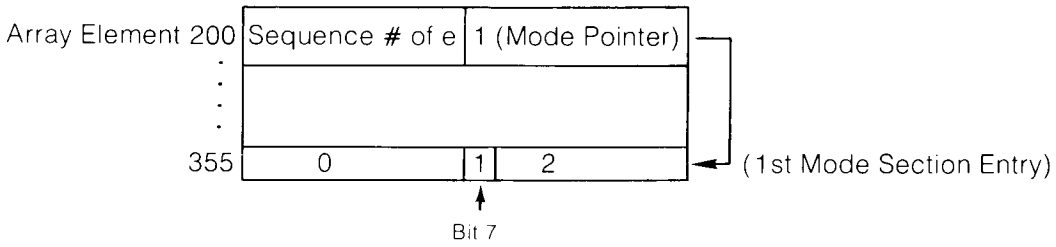
The symbol (<) means "precedes alphabetically". In this case, these letters have the same sequence number in the collating section, but different pointer numbers. That is, each has an individual entry in the mode section.

An accent priority requires only one entry in the mode section (some special cases require more than one). The format of the mode section entry for this special case is:

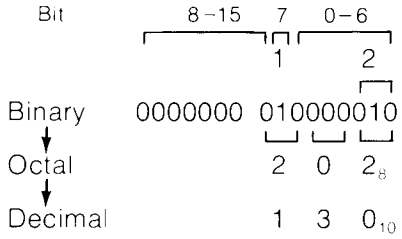


The zero indicates that the eight bits of this special case mode entry are not used, and must contain zero (0) so as not to be confused with one of the other special cases. A 1 must be present in bit seven to indicate that this special case is an accent priority. The remaining bits describe the user-defined priority number.

For example, assume that the letter é in the collating section has a pointer number which specifies the first entry of the mode section (or element 355 of the table). Also, assume that this mode entry assigns a priority of 2 to the letter é. The collating and mode sections contain the information shown next.



The following scheme is used to find the correct mode section entry.



130_{10} is the proper entry for the mode section element. Bits 6 through 0 are used to describe the priority number. Bit 7 is always 1 for an accent priority special case. Bits 15 through 8 are not used, but must contain zeros to insure proper results.

Note that the value of an accent priority entry in the mode section is always 128 plus the assigned priority. This means that the maximum accent priority is 127.

Accent priority mode section entries assign a relative priority to characters having the same sequence number. The accent priority is used only to distinguish two otherwise identical strings which differ only in their accent marks.

For example, assume ê and ë have the same sequence number and each has been assigned an accent priority; ê has priority 5, and ë has priority 2. The routine to calculate LEX ("ê"," ë") first compares the sequence numbers of ê and ë.

Since the sequence numbers are the same (and each character has been assigned an accent priority), the accent priorities are next compared. Five is greater than 2, so LEX (" ê","ë") is 1 (i.e., ê> ë, or ê comes after ë alphabetically).

A character which is not assigned an accent priority in the mode section lexically precedes any other character with the same sequence number and which has an accent priority. For example, assume e and ê have the same sequence number, but that ê has been assigned an accent priority in the mode section while the mode pointer of e is 0. Since e has not been assigned an accent priority, LEX ("e","ê") is −1, or e precedes ê alphabetically.

In the following example, the two strings have the letter e (but with different accent marks) as the first character. Since the last characters of each string differ, a mismatch is encountered without using any accent priority information from the mode section.

LEX ("êz"," ëa") = −1

In the next example, both letters in one string match those in the other; the only difference being the accent marks on the first letters. In this case, the strings match and accent priority information is used to make a distinction. If the accent priority for ë has been specified to be less than the accent priority for ê, the results would be:

LEX ("êb"," ëb") = 1

1 For 2 Character Replacement

Another special case that you can specify is a 1 for 2 character replacement. This specifies that for collating purposes, a given combination of two characters is to be treated as a single character. That is, two characters are assigned a single sequence number.

An example of this are the letters "CH" together in the Spanish standard collating sequence. The correct alphabetical sequence in Spanish is A, B, C, CH, D... All words beginning with C followed by any letter other than H come before words beginning with C followed by H. Therefore, the two letters CH can be taken as a single letter coming between C and D in the alphabetical sequence.

The string "CHA" is to be stored as

ASCII Code for C	ASCII Code for H	ASCII Code for A

but for collating purposes, the sequence numbers fetched for this string would be:

Sequence # for CH
Sequence # for A

The mode section format is:

Seq. # of 2-character combination	Uppercase of 2nd character
Seq. # of 2-character combination	Lowercase of 2nd character
0	128_{10}

The front, or upper part of the first (and possibly second) entry, contains the sequence number of the 2-character combination. The last, or lower part of the entry, contains the ASCII value of the 2nd character.

Usually an entry for both the uppercase and lowercase versions of the 2nd character are specified. This is to catch both occurrences of the given character combination; for example, CH and Ch. However, this is not required. If the special 1 for 2 character replacement is to specify CH only (and not Ch), the lowercase entry would be omitted.

In the case that both an uppercase and lowercase entry appears, the sequence numbers in the upper part can be the same in both entries. They are the same in the Spanish table. However, different sequence numbers for each combination are acceptable also.

No matter how many entries are given, the last entry for this special case is equal to an octal 200, or a decimal 128. It is a terminator that indicates there are no further character combinations.

When a lexical calculation on a string encounters a character with a mode pointer to a mode entry of this type, the next character in the string is compared one-by-one with the possible second characters given in the mode entries. If a match is found, the two characters together are represented by the single sequence number given in the corresponding mode table entry. If no match is found, the first character is assigned the sequence number given in its collating section entry and processing continues with the next character in the string. Note that "don't care" characters in the string are not regarded as such for this type of replacement.

For example, the sequence numbers fetched for the string "CAH" (even if "A" is a "don't-care" character) would be:

Sequence # for C
Sequence # for A
Sequence # for H

Note that this special case does not cause any actual substitutions to be made in a string containing CH; CHA is still stored as:

ASCII Code for C	ASCII Code for H	ASCII Code for A

This means that for collating purposes, the character pair CH causes a single sequence number to be fetched.

Consider again the example of CH for Spanish. Assume that the letter C has sequence number 67 and D has sequence number 68. To include the combination CH, a new sequence number must be created between 67 and 68. To do this, the collating sequence is rearranged so that C has sequence number 67 and D has 69. Sequence number 68 is reserved for CH and the remaining sequence numbers are adjusted accordingly.

Assume that the 4th, 5th, and 6th elements in the mode sequence contain the information governing the CH special case. The mode sequence entries for this example are shown next.

Array Element 70	67	4
71	69	
. . .		
358	Sequence # for CH	H
359	Sequence # for CH	h
360	0	128_{10}

(4th Mode Section Entry)

Numerically, the mode section entries for this example would contain:

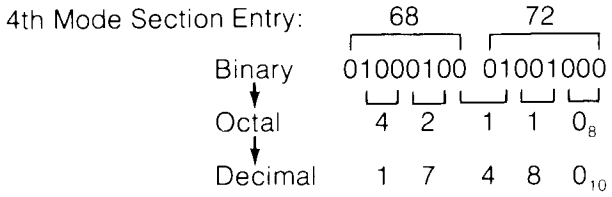
68	72
68	104
0	128

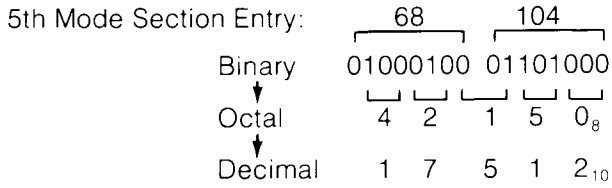
Note that 72 is the ASCII value for uppercase H and 104 is the ASCII value for lowercase h.

Whenever a C is encountered in collating and H or h immediately follows in the string, the pair of characters (CH or Ch) is assigned a single sequence number. Processing the string "CHA" would fetch the sequence numbers:

Sequence # for CH
Sequence # for A

In the original example, the actual contents of the mode sequence entries are:





Therefore, lexical table element 358 (1+1+256+96+4) contains 17 480$_{10}$, element 359 contains 17 512$_{10}$, and element 360 contains the terminator value 128$_{10}$.

It is possible to specify additional character replacements for the same character. For example, if the combination CZ were to follow CH in the previous example, the mode section would be expanded to include the additional characters as shown below.

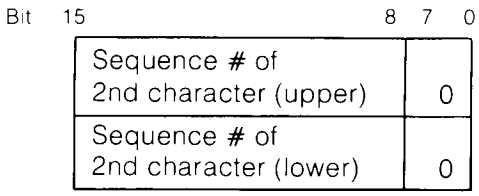
Sequence # for CH	H
Sequence # for Ch	h
Sequence # for CZ	Z
Sequence # for Cz	z
0	128

The terminator does not occur until all additional characters have been included.

2 For 1 Character Replacement

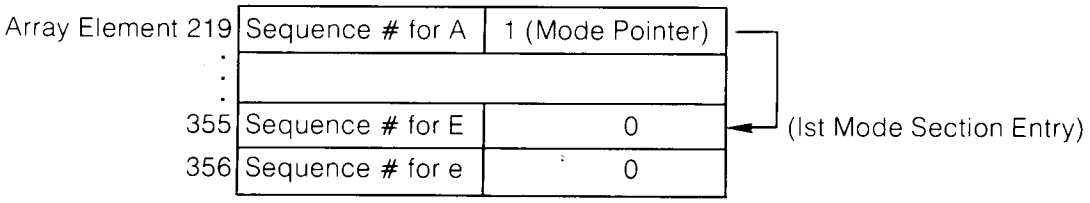
The third special case is a 2 for 1 character replacement. This type of mode entry specifies that a single character is to be assigned two sequence numbers for lexical comparisons. For example, the letter Ä in German is alphabetically equivalent to AE (or Ae). In a lexical comparison, Ä is assigned two sequence numbers; the sequence number for A and the sequence number for E (or e).

In general, if the mode entry specifies that a character is to be assigned two sequence numbers, the first sequence number is given in the normal collating section of the table. The second sequence number is given in the mode table entries. The format of the mode table entries is shown next.

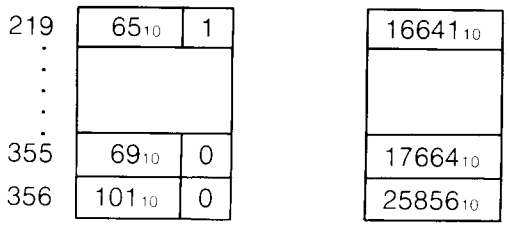


Since there is only one number required for this case, the second portion of the element (bits 7-0) is not used, and must be set to zero to avoid confusion.

Note that mode entries of this special case require no terminator element. For example, assume Ä is a special case character that is alphabetically equivalent to AE (or Ae), and that the special case for Ä is handled in the first entry of the mode section. The collating section entry for Ä (which is character code 216) and the mode section entry are shown next.



Assuming that the sequence numbers of A, E, and e are the same as their ASCII values, the following illustration represents the numerical contents of the table entries.



Note that the sequence number for Ä in the collating section is equal to the sequence number for A. Differentiation occurs in the mode section information. That is, A probably will have no mode section pointer, while Ä will. The sequence difference between A and Ä, then, is handled by Ä's entry in the mode section.

The choice between lower and uppercase for the second sequence number is determined by the case of the character immediately following the special case letter. For ÄN, A is equivalent to AE. For Än, A is equivalent to Ae. The character Ä alone is equivalent to AE.

This special case does not cause any actual substitutions to be made in a string. In the previous example, the string "Än" is stored as:

ASCII CODE for Ä	ASCII Code for n

For collating purposes, the sequence numbers fetched for the string "Än" would be:

Sequence # for A

Sequence # for e

Sequence # for n

"Don't Care" Characters

There is a fourth special case which doesn't require a mode section entry. This is the "don't care" special case where the specified character is to be ignored for collating purposes. An example of such a case is the hyphen in hyphenated words.

A character is ignored alphabetically if it has a sequence number of 255 in the collating section. That is, wherever a "don't care" condition exists in the collating section, simply enter 255 as the sequence number in that element.

Note that once a character is specified to be a "don't care" character, it is always treated as if it did not appear for collating purposes. For example, if the hyphen is designated as a "don't care" character, then "user-defined" would be collated as "userdefined", and, consequently, "−2" would be collated as "2". Therefore, consideration should be given to the desired results before a character is designated as a "don't care" condition.

It should also be noted that the null string precedes a string containing only a "don't care" character. That is, if A$ = "" and B$ = "−", then A$ precedes B$ even if the hyphen is a "don't care" character.

Advanced Programming Lexical Tables

The following listings are the Advanced Programming lexical tables. The tables are recorded on the Lexical Tables Cartridge under the names ASCII, FRENCH, GERMAN, SWEDSH and SPANSH. For your convenience, backup tables are included on the cartridge under the names BKUPAS, BKUPFR, BKUPGR, BKUPSW and BKUPSP.

ASCII TABLE

ENTRY #1 -- LENGTH OF COLLATING TABLE = 354
ENTRY #2 -- LENGTH OF MODE TABLE = 0

---------- COLLATING SECTION ----------

ENTRY #	COLL. ENTRY	CHAR CODE	CHAR	SEQ #	MODE PTR	ENTRY #	COLL. ENTRY	CHAR CODE	CHAR	SEQ #	MODE PTR
3	1	0	NUL	0	0	51	49	48	0	48	0
4	2	1	SOH	1	0	52	50	49	1	49	0
5	3	2	STX	2	0	53	51	50	2	50	0
6	4	3	ETX	3	0	54	52	51	3	51	0
7	5	4	EOT	4	0	55	53	52	4	52	0
8	6	5	ENQ	5	0	56	54	53	5	53	0
9	7	6	ACK	6	0	57	55	54	6	54	0
10	8	7	BEL	7	0	58	56	55	7	55	0
11	9	8	BS	8	0	59	57	56	8	56	0
12	10	9	HT	9	0	60	58	57	9	57	0
13	11	10	LF	10	0	61	59	58	:	58	0
14	12	11	VT	11	0	62	60	59	;	59	0
15	13	12	FF	12	0	63	61	60	<	60	0
16	14	13	CR	13	0	64	62	61	=	61	0
17	15	14	SO	14	0	65	63	62	>	62	0
18	16	15	SI	15	0	66	64	63	?	63	0
19	17	16	DLE	16	0	67	65	64	@	64	0
20	18	17	DC1	17	0	68	66	65	A	65	0
21	19	18	DC2	18	0	69	67	66	B	66	0
22	20	19	DC3	19	0	70	68	67	C	67	0
23	21	20	DC4	20	0	71	69	68	D	68	0
24	22	21	NAK	21	0	72	70	69	E	69	0
25	23	22	SYN	22	0	73	71	70	F	70	0
26	24	23	ETB	23	0	74	72	71	G	71	0
27	25	24	CAN	24	0	75	73	72	H	72	0
28	26	25	EM	25	0	76	74	73	I	73	0
29	27	26	SUB	26	0	77	75	74	J	74	0
30	28	27	ESC	27	0	78	76	75	K	75	0
31	29	28	FS	28	0	79	77	76	L	76	0
32	30	29	GS	29	0	80	78	77	M	77	0
33	31	30	RS	30	0	81	79	78	N	78	0
34	32	31	US	31	0	82	80	79	O	79	0
35	33	32	SP	32	0	83	81	80	P	80	0
36	34	33	!	33	0	84	82	81	Q	81	0
37	35	34	"	34	0	85	83	82	R	82	0
38	36	35	#	35	0	86	84	83	S	83	0
39	37	36	$	36	0	87	85	84	T	84	0
40	38	37	%	37	0	88	86	85	U	85	0
41	39	38	&	38	0	89	87	86	V	86	0
42	40	39	'	39	0	90	88	87	W	87	0
43	41	40	(	40	0	91	89	88	X	88	0
44	42	41	)	41	0	92	90	89	Y	89	0
45	43	42	*	42	0	93	91	90	Z	90	0
46	44	43	+	43	0	94	92	91	[	91	0
47	45	44	,	44	0	95	93	92	\	92	0
48	46	45	-	45	0	96	94	93	]	93	0
49	47	46	.	46	0	97	95	94	^	94	0
50	48	47	/	47	0	98	96	95	_	95	0

---------- COLLATING SECTION - CONT ----------

ENTRY #	COLL. ENTRY	CHAR CODE	CHAR	SEQ #	MODE PTR	ENTRY #	COLL. ENTRY	CHAR CODE	CHAR	SEQ #	MODE PTR
99	97	96	`	96	0	147	145	144		144	0
100	98	97	a	97	0	148	146	145		145	0
101	99	98	b	98	0	149	147	146		146	0
102	100	99	c	99	0	150	148	147		147	0
103	101	100	d	100	0	151	149	148		148	0
104	102	101	e	101	0	152	150	149		149	0
105	103	102	f	102	0	153	151	150		150	0
106	104	103	g	103	0	154	152	151		151	0
107	105	104	h	104	0	155	153	152		152	0
108	106	105	i	105	0	156	154	153		153	0
109	107	106	j	106	0	157	155	154		154	0
110	108	107	k	107	0	158	156	155		155	0
111	109	108	l	108	0	159	157	156		156	0
112	110	109	m	109	0	160	158	157		157	0
113	111	110	n	110	0	161	159	158		158	0
114	112	111	o	111	0	162	160	159		159	0
115	113	112	p	112	0	163	161	160		160	0
116	114	113	q	113	0	164	162	161	À	161	0
117	115	114	r	114	0	165	163	162	Ì	162	0
118	116	115	s	115	0	166	164	163	Ò	163	0
119	117	116	t	116	0	167	165	164	Ù	164	0
120	118	117	u	117	0	168	166	165	à	165	0
121	119	118	v	118	0	169	167	166	è	166	0
122	120	119	w	119	0	170	168	167	ò	167	0
123	121	120	x	120	0	171	169	168	´	168	0
124	122	121	y	121	0	172	170	169	`	169	0
125	123	122	z	122	0	173	171	170	^	170	0
126	124	123	{	123	0	174	172	171	¨	171	0
127	125	124	|	124	0	175	173	172	˜	172	0
128	126	125	}	125	0	176	174	173	é	173	0
129	127	126	~	126	0	177	175	174	ù	174	0
130	128	127	DEL	127	0	178	176	175	£	175	0
131	129	128	CLR	128	0	179	177	176	_	176	0
132	130	129	IV	129	0	180	178	177	Ā	177	0
133	131	130	BL	130	0	181	179	178	ā	178	0
134	132	131	IV-B	131	0	182	180	179	°	179	0
135	133	132	UL	132	0	183	181	180	Ç	180	0
136	134	133	IV-U	133	0	184	182	181	ç	181	0
137	135	134	BL-U	134	0	185	183	182	Ñ	182	0
138	136	135	I-B-U	135	0	186	184	183	ñ	183	0
139	137	136		136	0	187	185	184	¡	184	0
140	138	137		137	0	188	186	185	¿	185	0
141	139	138		138	0	189	187	186	¤	186	0
142	140	139		139	0	190	188	187	£	187	0
143	141	140		140	0	191	189	188	¤	188	0
144	142	141		141	0	192	190	189	§	189	0
145	143	142		142	0	193	191	190	²	190	0
146	144	143		143	0	194	192	191	¹	191	0

---------- COLLATING SECTION - CONT ----------

ENTRY #	COLL. ENTRY	CHAR CODE	CHAR	SEQ #	MODE PTR	ENTRY #	COLL. ENTRY	CHAR CODE	CHAR	SEQ #	MODE PTR
195	193	192	à	192	0	243	241	240		240	0
196	194	193	é	193	0	244	242	241		241	0
197	195	194	ô	194	0	245	243	242		242	0
198	196	195	û	195	0	246	244	243		243	0
199	197	196	à	196	0	247	245	244		244	0
200	198	197	è	197	0	248	246	245		245	0
201	199	198	ò	198	0	249	247	246		246	0
202	200	199	ù	199	0	250	248	247		247	0
203	201	200	ä	200	0	251	249	248		248	0
204	202	201	ë	201	0	252	250	249		249	0
205	203	202	ö	202	0	253	251	250		250	0
206	204	203	ü	203	0	254	252	251		251	0
207	205	204	ä	204	0	255	253	252		252	0
208	206	205	ë	205	0	256	254	253		253	0
209	207	206	ö	206	0	257	255	254		254	0
210	208	207	ü	207	0	258	256	255		255	0
211	209	208	Å	208	0						
212	210	209	î	209	0						
213	211	210	Ø	210	0						
214	212	211	Æ	211	0						
215	213	212	å	212	0						
216	214	213	í	213	0						
217	215	214	ø	214	0						
218	216	215	æ	215	0						
219	217	216	Ä	216	0						
220	218	217	ï	217	0						
221	219	218	Ö	218	0						
222	220	219	Ü	219	0						
223	221	220	É	220	0						
224	222	221	ï	221	0						
225	223	222	ß	222	0						
226	224	223		223	0						
227	225	224		224	0						
228	226	225		225	0						
229	227	226		226	0						
230	228	227		227	0						
231	229	228		228	0						
232	230	229		229	0						
233	231	230		230	0						
234	232	231		231	0						
235	233	232		232	0						
236	234	233		233	0						
237	235	234		234	0						
238	236	235		235	0						
239	237	236		236	0						
240	238	237		237	0						
241	239	238		238	0						
242	240	239		239	0						

---------- UPPERCASE/LOWERCASE SECTION ----------

ENTRY #	U/L ENTRY	CHAR CODE	CHAR	UPPER CASE	LOWER CASE	ENTRY #	U/L ENTRY	CHAR CODE	CHAR	UPPER CASE	LOWER CASE
259	1	160				307	49	208	Á	Á	à
260	2	161	Á	Á	à	308	50	209	î	I	î
261	3	162	Í	Í	í	309	51	210	Ø	Ø	ø
262	4	163	Ó	Ó	ó	310	52	211	Æ	Æ	æ
263	5	164	Ú	Ú	ú	311	53	212	à	Á	à
264	6	165	Á	Á	à	312	54	213	í	Í	í
265	7	166	É	É	é	313	55	214	ø	Ø	ø
266	8	167	Ó	Ó	o	314	56	215	æ	Æ	æ
267	9	168	´	´	´	315	57	216	Ä	Ä	à
268	10	169	`	`	`	316	58	217	ï	I	ï
269	11	170	^	^	^	317	59	218	Ö	Ö	ö
270	12	171	¨	¨	¨	318	60	219	Ü	Ü	ü
271	13	172	~	~	~	319	61	220	É	É	é
272	14	173	É	É	é	320	62	221	ï	I	ï
273	15	174	Ô	Ô	ô	321	63	222	ß	ß	ß
274	16	175	£	£	£	322	64	223			
275	17	176				323	65	224			
276	18	177	Ã	Ã	ã	324	66	225			
277	19	178	ã	Ã	ã	325	67	226			
278	20	179	°	°	°	326	68	227			
279	21	180	Ç	Ç	ç	327	69	228			
280	22	181	ç	Ç	ç	328	70	229			
281	23	182	Ñ	Ñ	ñ	329	71	230			
282	24	183	ñ	Ñ	ñ	330	72	231			
283	25	184	¡	¡	¡	331	73	232			
284	26	185	¿	¿	¿	332	74	233			
285	27	186	¤	¤	¤	333	75	234			
286	28	187	£	£	£	334	76	235			
287	29	188	¤	¤	¤	335	77	236			
288	30	189	§	§	§	336	78	237			
289	31	190	2	2	2	337	79	238			
290	32	191	+	+	+	338	80	239			
291	33	192	à	A	à	339	81	240			
292	34	193	é	É	é	340	82	241			
293	35	194	ô	Ô	ô	341	83	242			
294	36	195	Ú	U	Ú	342	84	243			
295	37	196	à	Á	à	343	85	244			
296	38	197	é	É	é	344	86	245			
297	39	198	ò	Ò	ò	345	87	246			
298	40	199	ú	Ú	ú	346	88	247			
299	41	200	à	Á	à	347	89	248			
300	42	201	è	É	è	348	90	249			
301	43	202	ò	Ò	ò	349	91	250			
302	44	203	ù	U	ù	350	92	251			
303	45	204	ä	Ä	ä	351	93	252			
304	46	205	ë	E	ë	352	94	253			
305	47	206	ö	Ö	ö	353	95	254			
306	48	207	ü	Ü	ü	354	96	255			

---------- MODE SECTION ----------

ENTRY #	MODE ENTRY	TYPE DESCRIPTION

|||||||||| TYPE DETAILS ||||||||||

-- DON'T CARES --

0 ASCII - 255 = Don't care

-- ACCENT PRIORITIES --

-- TWO FOR ONE REPLACEMENTS --

-- ONE FOR TWO REPLACEMENTS --

FRENCH TABLE

ENTRY #1 -- LENGTH OF COLLATING TABLE = 374
ENTRY #2 -- LENGTH OF MODE TABLE = 20

---------- COLLATING SECTION ----------

ENTRY #	COLL. ENTRY	CHAR CODE	CHAR	SEQ #	MODE PTR	ENTRY #	COLL. ENTRY	CHAR CODE	CHAR	SEQ #	MODE PTR
3	1	0	NUL	0	0	51	49	48	0	48	0
4	2	1	SOH	1	0	52	50	49	1	49	0
5	3	2	STX	2	0	53	51	50	2	50	0
6	4	3	ETX	3	0	54	52	51	3	51	0
7	5	4	EOT	4	0	55	53	52	4	52	0
8	6	5	ENQ	5	0	56	54	53	5	53	0
9	7	6	ACK	6	0	57	55	54	6	54	0
10	8	7	BEL	7	0	58	56	55	7	55	0
11	9	8	BS	8	0	59	57	56	8	56	0
12	10	9	HT	9	0	60	58	57	9	57	0
13	11	10	LF	10	0	61	59	58	:	58	0
14	12	11	VT	11	0	62	60	59	;	59	0
15	13	12	FF	12	0	63	61	60	<	60	0
16	14	13	CR	13	0	64	62	61	=	61	0
17	15	14	SO	14	0	65	63	62	>	62	0
18	16	15	SI	15	0	66	64	63	?	63	0
19	17	16	DLE	16	0	67	65	64	@	64	0
20	18	17	DC1	17	0	68	66	65	A	65	0
21	19	18	DC2	18	0	69	67	66	B	68	0
22	20	19	DC3	19	0	70	68	67	C	69	0
23	21	20	DC4	20	0	71	69	68	D	70	0
24	22	21	NAK	21	0	72	70	69	E	71	0
25	23	22	SYN	22	0	73	71	70	F	72	0
26	24	23	ETB	23	0	74	72	71	G	73	0
27	25	24	CAN	24	0	75	73	72	H	74	0
28	26	25	EM	25	0	76	74	73	I	75	0
29	27	26	SUB	26	0	77	75	74	J	76	0
30	28	27	ESC	27	0	78	76	75	K	77	0
31	29	28	FS	28	0	79	77	76	L	78	0
32	30	29	GS	29	0	80	78	77	M	79	0
33	31	30	RS	30	0	81	79	78	N	80	0
34	32	31	US	31	0	82	80	79	O	82	0
35	33	32	SP	32	0	83	81	80	P	84	0
36	34	33	!	33	0	84	82	81	Q	85	0
37	35	34	"	34	0	85	83	82	R	86	0
38	36	35	#	35	0	86	84	83	S	87	0
39	37	36	$	36	0	87	85	84	T	88	0
40	38	37	%	37	0	88	86	85	U	89	0
41	39	38	&	38	0	89	87	86	V	91	0
42	40	39	'	39	0	90	88	87	W	92	0
43	41	40	(	40	0	91	89	88	X	93	0
44	42	41	)	41	0	92	90	89	Y	94	0
45	43	42	*	42	0	93	91	90	Z	95	0
46	44	43	+	43	0	94	92	91	[	96	0
47	45	44	,	44	0	95	93	92	\	97	0
48	46	45	-	255	0	96	94	93	]	98	0
49	47	46	.	46	0	97	95	94	^	99	0
50	48	47	/	47	0	98	96	95	_	100	0

---------- COLLATING SECTION - CONT ----------

ENTRY #	COLL. ENTRY	CHAR CODE	CHAR	SEQ #	MODE PTR	ENTRY #	COLL. ENTRY	CHAR CODE	CHAR	SEQ #	MODE PTR
99	97	96	`	101	0	147	145	144		0	0
100	98	97	a	102	0	148	146	145		0	0
101	99	98	b	103	0	149	147	146		0	0
102	100	99	c	104	0	150	148	147		0	0
103	101	100	d	106	0	151	149	148		0	0
104	102	101	e	107	0	152	150	149		0	0
105	103	102	f	108	0	153	151	150		0	0
106	104	103	g	109	0	154	152	151		0	0
107	105	104	h	110	0	155	153	152		0	0
108	106	105	i	111	0	156	154	153		0	0
109	107	106	j	112	0	157	155	154		0	0
110	108	107	k	113	0	158	156	155		0	0
111	109	108	l	114	0	159	157	156		0	0
112	110	109	m	115	0	160	158	157		0	0
113	111	110	n	116	0	161	159	158		0	0
114	112	111	o	118	0	162	160	159		0	0
115	113	112	p	119	0	163	161	160		0	0
116	114	113	q	120	0	164	162	161	Á	65	0
117	115	114	r	121	0	165	163	162	Í	75	0
118	116	115	s	122	0	166	164	163	Ó	82	0
119	117	116	t	123	0	167	165	164	Ú	89	0
120	118	117	u	124	0	168	166	165	À	65	0
121	119	118	v	125	0	169	167	166	È	71	0
122	120	119	w	126	0	170	168	167	Ò	82	0
123	121	120	x	127	0	171	169	168	´	137	0
124	122	121	y	128	0	172	170	169	`	138	0
125	123	122	z	129	0	173	171	170	^	139	0
126	124	123	{	133	0	174	172	171	¨	140	0
127	125	124	|	134	0	175	173	172	~	141	0
128	126	125	}	135	0	176	174	173	Ë	71	0
129	127	126	~	136	0	177	175	174	Ö	82	0
130	128	127	DEL	0	0	178	176	175	£	142	0
131	129	128	CLR	0	0	179	177	176		143	0
132	130	129	IV	0	0	180	178	177	Ã	65	0
133	131	130	BL	0	0	181	179	178	ã	102	0
134	132	131	IV-B	0	0	182	180	179	°	144	0
135	133	132	UL	0	0	183	181	180	Ç	69	0
136	134	133	IV-U	0	0	184	182	181	ç	105	0
137	135	134	BL-U	0	0	185	183	182	Ñ	81	0
138	136	135	I-B-U	0	0	186	184	183	ñ	117	0
139	137	136		0	0	187	185	184	¡	145	0
140	138	137		0	0	188	186	185	¿	146	0
141	139	138		0	0	189	187	186	¤	147	0
142	140	139		0	0	190	188	187	£	148	0
143	141	140		0	0	191	189	188	¥	149	0
144	142	141		0	0	192	190	189	§	150	0
145	143	142		0	0	193	191	190	¿	151	0
146	144	143		0	0	194	192	191	·	152	0

---------- COLLATING SECTION - CONT ----------

ENTRY #	COLL. ENTRY	CHAR CODE	CHAR	SEQ #	MODE PTR	ENTRY #	COLL. ENTRY	CHAR CODE	CHAR	SEQ #	MODE PTR
195	193	192	å	102	0	243	241	240		0	0
196	194	193	é	107	0	244	242	241		0	0
197	195	194	ô	118	0	245	243	242		0	0
198	196	195	û	124	0	246	244	243		0	0
199	197	196	à	102	0	247	245	244		0	0
200	198	197	é	107	0	248	246	245		0	0
201	199	198	ó	118	0	249	247	246		0	0
202	200	199	ú	124	0	250	248	247		0	0
203	201	200	à	102	0	251	249	248		0	0
204	202	201	é	107	0	252	250	249		0	0
205	203	202	ó	118	0	253	251	250		0	0
206	204	203	ú	124	0	254	252	251		0	0
207	205	204	ä	102	0	255	253	252		0	0
208	206	205	ë	107	0	256	254	253		0	0
209	207	206	ö	118	0	257	255	254		0	0
210	208	207	ü	124	0	258	256	255		0	0
211	209	208	Å	65	0						
212	210	209	î	111	0						
213	211	210	Ø	82	0						
214	212	211	Æ	65	0						
215	213	212	à	102	0						
216	214	213	î	111	0						
217	215	214	ø	118	0						
218	216	215	æ	102	0						
219	217	216	Å	65	0						
220	218	217	î	111	0						
221	219	218	Ö	82	0						
222	220	219	Ü	89	0						
223	221	220	É	71	0						
224	222	221	ï	111	0						
225	223	222	ß	122	1						
226	224	223		0	0						
227	225	224		0	0						
228	226	225		0	0						
229	227	226		0	0						
230	228	227		0	0						
231	229	228		0	0						
232	230	229		0	0						
233	231	230		0	0						
234	232	231		0	0						
235	233	232		0	0						
236	234	233		0	0						
237	235	234		0	0						
238	236	235		0	0						
239	237	236		0	0						
240	238	237		0	0						
241	239	238		0	0						
242	240	239		0	0						

---------- UPPERCASE/LOWERCASE SECTION ----------

ENTRY #	U/L ENTRY	CHAR CODE	CHAR	UPPER CASE	LOWER CASE	ENTRY #	U/L ENTRY	CHAR CODE	CHAR	UPPER CASE	LOWER CASE
259	1	160				307	49	208	Â	Â	â
260	2	161	À	À	à	308	50	209	Î	I	î
261	3	162	Í	Í	í	309	51	210	Ø	Ø	ø
262	4	163	Ó	Ó	ó	310	52	211	Æ	Æ	æ
263	5	164	Ú	Ú	ú	311	53	212	à	À	à
264	6	165	Á	Á	á	312	54	213	í	I	í
265	7	166	É	É	é	313	55	214	ø	Ø	ø
266	8	167	Ó	Ó	o	314	56	215	æ	Æ	æ
267	9	168	´	´	´	315	57	216	Ä	Ä	ä
268	10	169	`	`	`	316	58	217	î	I	î
269	11	170	^	^	^	317	59	218	Ö	Ö	ö
270	12	171	¨	¨	¨	318	60	219	Ü	Ü	ü
271	13	172	~	~	~	319	61	220	É	É	é
272	14	173	É	É	é	320	62	221	ï	I	ï
273	15	174	Ô	Ô	ô	321	63	222	ß	ß	ß
274	16	175	£	£	£	322	64	223			
275	17	176				323	65	224			
276	18	177	Ã	Ã	ã	324	66	225			
277	19	178	ã	Ã	ã	325	67	226			
278	20	179	°	°	°	326	68	227			
279	21	180	Ç	Ç	ç	327	69	228			
280	22	181	ç	Ç	ç	328	70	229			
281	23	182	Ñ	Ñ	ñ	329	71	230			
282	24	183	ñ	Ñ	ñ	330	72	231			
283	25	184	¡	¡	¡	331	73	232			
284	26	185	¿	¿	¿	332	74	233			
285	27	186	¤	¤	¤	333	75	234			
286	28	187	£	£	£	334	76	235			
287	29	188	º	º	º	335	77	236			
288	30	189	§	§	§	336	78	237			
289	31	190	2	2	2	337	79	238			
290	32	191	·	·	·	338	80	239			
291	33	192	å	A	å	339	81	240			
292	34	193	ê	E	ê	340	82	241			
293	35	194	ô	O	ô	341	83	242			
294	36	195	û	U	û	342	84	243			
295	37	196	à	A	à	343	85	244			
296	38	197	è	E	è	344	86	245			
297	39	198	ò	O	ò	345	87	246			
298	40	199	ù	U	ù	346	88	247			
299	41	200	ä	A	ä	347	89	248			
300	42	201	ë	E	ë	348	90	249			
301	43	202	ö	O	ö	349	91	250			
302	44	203	ü	U	ü	350	92	251			
303	45	204	ä	A	ä	351	93	252			
304	46	205	ë	E	ë	352	94	253			
305	47	206	ö	O	ö	353	95	254			
306	48	207	ü	U	ü	354	96	255			

---------- MODE SECTION ----------

ENTRY #	MODE ENTRY	TYPE DESCRIPTION
355	1	TWO FOR ONE CHARACTER REPLACEMENT
356	2	TWO FOR ONE CHARACTER REPLACEMENT

|||||||||| TYPE DETAILS ||||||||||

-- DON'T CARES --

0 - ASCII - 45 = Don't care

-- ACCENT PRIORITIES --

-- TWO FOR ONE REPLACEMENTS --

ENTRY #	REPLACEMENT	ASCII
1	ß = ss = ss	222

-- ONE FOR TWO REPLACEMENTS --

GERMAN TABLE

ENTRY #1 -- LENGTH OF COLLATING TABLE = 374
ENTRY #2 -- LENGTH OF MODE TABLE = 20

---------- COLLATING SECTION ----------

ENTRY #	COLL. ENTRY	CHAR CODE	CHAR	SEQ #	MODE PTR	ENTRY #	COLL. ENTRY	CHAR CODE	CHAR	SEQ #	MODE PTR
3	1	0	NUL	0	0	51	49	48	0	48	0
4	2	1	SOH	1	0	52	50	49	1	49	0
5	3	2	STX	2	0	53	51	50	2	50	0
6	4	3	ETX	3	0	54	52	51	3	51	0
7	5	4	EOT	4	0	55	53	52	4	52	0
8	6	5	ENQ	5	0	56	54	53	5	53	0
9	7	6	ACK	6	0	57	55	54	6	54	0
10	8	7	BEL	7	0	58	56	55	7	55	0
11	9	8	BS	8	0	59	57	56	8	56	0
12	10	9	HT	9	0	60	58	57	9	57	0
13	11	10	LF	10	0	61	59	58	:	58	0
14	12	11	VT	11	0	62	60	59	;	59	0
15	13	12	FF	12	0	63	61	60	<	60	0
16	14	13	CR	13	0	64	62	61	=	61	0
17	15	14	SO	14	0	65	63	62	>	62	0
18	16	15	SI	15	0	66	64	63	?	63	0
19	17	16	DLE	16	0	67	65	64	@	64	0
20	18	17	DC1	17	0	68	66	65	A	65	0
21	19	18	DC2	18	0	69	67	66	B	71	0
22	20	19	DC3	19	0	70	68	67	C	72	0
23	21	20	DC4	20	0	71	69	68	D	74	0
24	22	21	NAK	21	0	72	70	69	E	75	0
25	23	22	SYN	22	0	73	71	70	F	79	0
26	24	23	ETB	23	0	74	72	71	G	80	0
27	25	24	CAN	24	0	75	73	72	H	81	0
28	26	25	EM	25	0	76	74	73	I	82	0
29	27	26	SUB	26	0	77	75	74	J	84	0
30	28	27	ESC	27	0	78	76	75	K	85	0
31	29	28	FS	28	0	79	77	76	L	86	0
32	30	29	GS	29	0	80	78	77	M	87	0
33	31	30	RS	30	0	81	79	78	N	88	0
34	32	31	US	31	0	82	80	79	O	90	0
35	33	32	SP	32	0	83	81	80	P	95	0
36	34	33	!	33	0	84	82	81	Q	96	0
37	35	34	"	34	0	85	83	82	R	97	0
38	36	35	#	35	0	86	84	83	S	98	0
39	37	36	$	36	0	87	85	84	T	99	0
40	38	37	%	37	0	88	86	85	U	100	0
41	39	38	&	38	0	89	87	86	V	102	0
42	40	39	'	39	0	90	88	87	W	103	0
43	41	40	(	40	0	91	89	88	X	104	0
44	42	41	)	41	0	92	90	89	Y	105	0
45	43	42	*	42	0	93	91	90	Z	106	0
46	44	43	+	43	0	94	92	91	[	107	0
47	45	44	,	44	0	95	93	92	\	108	0
48	46	45	-	45	0	96	94	93	]	109	0
49	47	46	.	46	0	97	95	94	^	110	0
50	48	47	/	47	0	98	96	95	_	111	0

---------- COLLATING SECTION - CONT ----------

ENTRY #	COLL. ENTRY	CHAR CODE	CHAR	SEQ #	MODE PTR	ENTRY #	COLL. ENTRY	CHAR CODE	CHAR	SEQ #	MODE PTR
99	97	96	`	112	0	147	145	144		0	0
100	98	97	a	113	0	148	146	145		0	0
101	99	98	b	120	0	149	147	146		0	0
102	100	99	c	121	0	150	148	147		0	0
103	101	100	d	123	0	151	149	148		0	0
104	102	101	e	124	0	152	150	149		0	0
105	103	102	f	129	0	153	151	150		0	0
106	104	103	g	130	0	154	152	151		0	0
107	105	104	h	131	0	155	153	152		0	0
108	106	105	i	132	0	156	154	153		0	0
109	107	106	j	137	0	157	155	154		0	0
110	108	107	k	138	0	158	156	155		0	0
111	109	108	l	139	0	159	157	156		0	0
112	110	109	m	140	0	160	158	157		0	0
113	111	110	n	141	0	161	159	158		0	0
114	112	111	o	143	0	162	160	159		0	0
115	113	112	p	148	0	163	161	160		0	0
116	114	113	q	149	0	164	162	161	Ä	68	0
117	115	114	r	150	0	165	163	162	Ï	83	0
118	116	115	s	151	0	166	164	163	Ö	91	0
119	117	116	t	152	0	167	165	164	Ü	101	0
120	118	117	u	153	0	168	166	165	Ä	69	0
121	119	118	v	157	0	169	167	166	Ë	77	0
122	120	119	w	158	0	170	168	167	Ö	92	0
123	121	120	x	159	0	171	169	168	´	166	0
124	122	121	y	160	0	172	170	169	`	167	0
125	123	122	z	161	0	173	171	170	^	168	0
126	124	123	{	162	0	174	172	171	¨	169	0
127	125	124	|	163	0	175	173	172	~	170	0
128	126	125	}	164	0	176	174	173	Ë	78	0
129	127	126	~	165	0	177	175	174	Ö	93	0
130	128	127	DEL	0	0	178	176	175	£	171	0
131	129	128	CLR	0	0	179	177	176		172	0
132	130	129	IV	0	0	180	178	177	Ä	70	0
133	131	130	BL	0	0	181	179	178	ä	119	0
134	132	131	IV-B	0	0	182	180	179	°	173	0
135	133	132	UL	0	0	183	181	180	Ç	73	0
136	134	133	IV-U	0	0	184	182	181	ç	122	0
137	135	134	BL-U	0	0	185	183	182	Ñ	89	0
138	136	135	I-B-U	0	0	186	184	183	ñ	142	0
139	137	136		0	0	187	185	184	¡	174	0
140	138	137		0	0	188	186	185	¿	175	0
141	139	138		0	0	189	187	186	¤	176	0
142	140	139		0	0	190	188	187	£	177	0
143	141	140		0	0	191	189	188	°	178	0
144	142	141		0	0	192	190	189	§	179	0
145	143	142		0	0	193	191	190	²	180	0
146	144	143		0	0	194	192	191	'	181	0

---------- COLLATING SECTION - CONT ----------

ENTRY #	COLL. ENTRY	CHAR CODE	CHAR	SEQ #	MODE PTR	ENTRY #	COLL. ENTRY	CHAR CODE	CHAR	SEQ #	MODE PTR
195	193	192	à	118	0	243	241	240	´	0	0
196	194	193	é	127	0	244	242	241		0	0
197	195	194	ò	146	0	245	243	242		0	0
198	196	195	ù	156	0	246	244	243		0	0
199	197	196	á	116	0	247	245	244		0	0
200	198	197	é	125	0	248	246	245		0	0
201	199	198	ó	144	0	249	247	246		0	0
202	200	199	ú	154	0	250	248	247		0	0
203	201	200	à	117	0	251	249	248		0	0
204	202	201	è	126	0	252	250	249		0	0
205	203	202	ò	145	0	253	251	250		0	0
206	204	203	ù	155	0	254	252	251		0	0
207	205	204	ä	113	1	255	253	252		0	0
208	206	205	ë	128	0	256	254	253		0	0
209	207	206	ö	143	3	257	255	254		0	0
210	208	207	ü	153	5	258	256	255		0	0
211	209	208	Á	67	0						
212	210	209	î	135	0						
213	211	210	Ø	94	0						
214	212	211	Æ	66	0						
215	213	212	å	115	0						
216	214	213	í	133	0						
217	215	214	ø	147	0						
218	216	215	æ	114	0						
219	217	216	Ä	65	7						
220	218	217	ì	134	0						
221	219	218	Ö	90	9						
222	220	219	Ü	100	11						
223	221	220	É	76	0						
224	222	221	ï	136	0						
225	223	222	ß	151	13						
226	224	223		0	0						
227	225	224		0	0						
228	226	225		0	0						
229	227	226		0	0						
230	228	227		0	0						
231	229	228		0	0						
232	230	229		0	0						
233	231	230		0	0						
234	232	231		0	0						
235	233	232		0	0						
236	234	233		0	0						
237	235	234		0	0						
238	236	235		0	0						
239	237	236		0	0						
240	238	237		0	0						
241	239	238		0	0						
242	240	239		0	0						

---------- UPPERCASE/LOWERCASE SECTION ----------

ENTRY #	U/L ENTRY	CHAR CODE	CHAR	UPPER CASE	LOWER CASE	ENTRY #	U/L ENTRY	CHAR CODE	CHAR	UPPER CASE	LOWER CASE
259	1	160				307	49	208	Á	Á	á
260	2	161	Á	Á	á	308	50	209	Î	I	î
261	3	162	Í	Í	í	309	51	210	Ø	Ø	ø
262	4	163	Ó	Ó	ó	310	52	211	Æ	Æ	æ
263	5	164	Ú	Ú	ú	311	53	212	à	Á	à
264	6	165	Á	Á	à	312	54	213	í	Í	í
265	7	166	É	É	é	313	55	214	ø	Ø	ø
266	8	167	Ó	Ó	o	314	56	215	æ	Æ	æ
267	9	168	´	´	´	315	57	216	Ã	Ã	ã
268	10	169	`	`	`	316	58	217	î	I	î
269	11	170	^	^	^	317	59	218	Õ	Õ	õ
270	12	171	¨	¨	¨	318	60	219	Ü	Ü	ü
271	13	172	~	~	~	319	61	220	É	É	è
272	14	173	É	É	è	320	62	221	ï	I	ï
273	15	174	Ô	Ô	ô	321	63	222	ß	ß	ß
274	16	175	£	£	£	322	64	223			
275	17	176				323	65	224			
276	18	177	Ã	Ã	ã	324	66	225			
277	19	178	ã	Ã	ã	325	67	226			
278	20	179	°	°	°	326	68	227			
279	21	180	Ç	Ç	ç	327	69	228			
280	22	181	ç	C	ç	328	70	229			
281	23	182	Ñ	Ñ	ñ	329	71	230			
282	24	183	ñ	Ñ	ñ	330	72	231			
283	25	184	¡	¡	¡	331	73	232			
284	26	185	¿	¿	¿	332	74	233			
285	27	186	¤	¤	¤	333	75	234			
286	28	187	£	£	£	334	76	235			
287	29	188	ª	ª	ª	335	77	236			
288	30	189	§	§	§	336	78	237			
289	31	190	²	²	²	337	79	238			
290	32	191	¹	¹	¹	338	80	239			
291	33	192	à	Á	à	339	81	240			
292	34	193	é	É	é	340	82	241			
293	35	194	ô	Ô	ô	341	83	242			
294	36	195	ú	U	ú	342	84	243			
295	37	196	á	Á	á	343	85	244			
296	38	197	é	É	é	344	86	245			
297	39	198	ó	Ó	ó	345	87	246			
298	40	199	ú	Ú	ú	346	88	247			
299	41	200	à	Á	à	347	89	248			
300	42	201	è	É	è	348	90	249			
301	43	202	ò	Ó	ò	349	91	250			
302	44	203	ù	U	ù	350	92	251			
303	45	204	ä	Ä	ä	351	93	252			
304	46	205	ë	E	ë	352	94	253			
305	47	206	ö	Ö	ö	353	95	254			
306	48	207	ü	Ü	ü	354	96	255			

---------- MODE SECTION ----------

ENTRY #	MODE ENTRY	TYPE DESCRIPTION
355	1	TWO FOR ONE CHARACTER REPLACEMENT
356	2	TWO FOR ONE CHARACTER REPLACEMENT
357	3	TWO FOR ONE CHARACTER REPLACEMENT
358	4	TWO FOR ONE CHARACTER REPLACEMENT
359	5	TWO FOR ONE CHARACTER REPLACEMENT
360	6	TWO FOR ONE CHARACTER REPLACEMENT
361	7	TWO FOR ONE CHARACTER REPLACEMENT
362	8	TWO FOR ONE CHARACTER REPLACEMENT
363	9	TWO FOR ONE CHARACTER REPLACEMENT
364	10	TWO FOR ONE CHARACTER REPLACEMENT
365	11	TWO FOR ONE CHARACTER REPLACEMENT
366	12	TWO FOR ONE CHARACTER REPLACEMENT
367	13	TWO FOR ONE CHARACTER REPLACEMENT
368	14	TWO FOR ONE CHARACTER REPLACEMENT

|||||||||| TYPE DETAILS ||||||||||

-- DON'T CARES --

-- ACCENT PRIORITIES --

-- TWO FOR ONE REPLACEMENTS --

ENTRY #	REPLACEMENT	ASCII
1	ä = ae = ae	204
3	ö = oe = oe	206
5	ü = ue = ue	207
7	Ä = AE = Ae	216
9	Ö = OE = Oe	218
11	Ü = UE = Ue	219
13	ß = ss = ss	222

-- ONE FOR TWO REPLACEMENTS --

SPANISH TABLE

ENTRY #1 -- LENGTH OF COLLATING TABLE = 374
ENTRY #2 -- LENGTH OF MODE TABLE = 20

---------- COLLATING SECTION ----------

ENTRY #	COLL. ENTRY	CHAR CODE	CHAR	SEQ #	MODE PTR	ENTRY #	COLL. ENTRY	CHAR CODE	CHAR	SEQ #	MODE PTR
3	1	0	NUL	0	0	51	49	48	0	48	0
4	2	1	SOH	1	0	52	50	49	1	49	0
5	3	2	STX	2	0	53	51	50	2	50	0
6	4	3	ETX	3	0	54	52	51	3	51	0
7	5	4	EOT	4	0	55	53	52	4	52	0
8	6	5	ENQ	5	0	56	54	53	5	53	0
9	7	6	ACK	6	0	57	55	54	6	54	0
10	8	7	BEL	7	0	58	56	55	7	55	0
11	9	8	BS	8	0	59	57	56	8	56	0
12	10	9	HT	9	0	60	58	57	9	57	0
13	11	10	LF	10	0	61	59	58	:	58	0
14	12	11	VT	11	0	62	60	59	;	59	0
15	13	12	FF	12	0	63	61	60	<	60	0
16	14	13	CR	13	0	64	62	61	=	61	0
17	15	14	SO	14	0	65	63	62	>	62	0
18	16	15	SI	15	0	66	64	63	?	63	0
19	17	16	DLE	16	0	67	65	64	@	64	0
20	18	17	DC1	17	0	68	66	65	A	65	0
21	19	18	DC2	18	0	69	67	66	B	66	0
22	20	19	DC3	19	0	70	68	67	C	67	1
23	21	20	DC4	20	0	71	69	68	D	69	0
24	22	21	NAK	21	0	72	70	69	E	70	0
25	23	22	SYN	22	0	73	71	70	F	71	0
26	24	23	ETB	23	0	74	72	71	G	72	0
27	25	24	CAN	24	0	75	73	72	H	73	0
28	26	25	EM	25	0	76	74	73	I	74	0
29	27	26	SUB	26	0	77	75	74	J	75	0
30	28	27	ESC	27	0	78	76	75	K	76	0
31	29	28	FS	28	0	79	77	76	L	77	4
32	30	29	GS	29	0	80	78	77	M	79	0
33	31	30	RS	30	0	81	79	78	N	80	0
34	32	31	US	31	0	82	80	79	O	82	0
35	33	32	SP	32	0	83	81	80	P	83	0
36	34	33	!	33	0	84	82	81	Q	84	0
37	35	34	"	34	0	85	83	82	R	85	0
38	36	35	#	35	0	86	84	83	S	86	0
39	37	36	$	36	0	87	85	84	T	87	0
40	38	37	%	37	0	88	86	85	U	88	0
41	39	38	&	38	0	89	87	86	V	89	0
42	40	39	'	39	0	90	88	87	W	90	0
43	41	40	(	40	0	91	89	88	X	91	0
44	42	41	)	41	0	92	90	89	Y	92	0
45	43	42	*	42	0	93	91	90	Z	93	0
46	44	43	+	43	0	94	92	91	[	98	0
47	45	44	,	44	0	95	93	92	\	99	0
48	46	45	-	45	0	96	94	93	]	100	0
49	47	46	.	46	0	97	95	94	^	101	0
50	48	47	/	47	0	98	96	95	_	102	0

---------- COLLATING SECTION - CONT ----------

ENTRY #	COLL. ENTRY	CHAR CODE	CHAR	SEQ #	MODE PTR	ENTRY #	COLL. ENTRY	CHAR CODE	CHAR	SEQ #	MODE PTR
99	97	96	`	103	0	147	145	144		0	0
100	98	97	a	104	0	148	146	145		0	0
101	99	98	b	105	0	149	147	146		0	0
102	100	99	c	106	7	150	148	147		0	0
103	101	100	d	108	0	151	149	148		0	0
104	102	101	e	109	0	152	150	149		0	0
105	103	102	f	110	0	153	151	150		0	0
106	104	103	g	111	0	154	152	151		0	0
107	105	104	h	112	0	155	153	152		0	0
108	106	105	i	113	0	156	154	153		0	0
109	107	106	j	114	0	157	155	154		0	0
110	108	107	k	115	0	158	156	155		0	0
111	109	108	l	116	10	159	157	156		0	0
112	110	109	m	118	0	160	158	157		0	0
113	111	110	n	119	0	161	159	158		0	0
114	112	111	o	121	0	162	160	159		0	0
115	113	112	p	122	0	163	161	160		0	0
116	114	113	q	123	0	164	162	161	Ä	65	0
117	115	114	r	124	0	165	163	162	Ï	74	0
118	116	115	s	125	0	166	164	163	Ö	82	0
119	117	116	t	126	0	167	165	164	Ü	88	0
120	118	117	u	127	0	168	166	165	Ä	65	0
121	119	118	v	128	0	169	167	166	Ë	70	0
122	120	119	w	129	0	170	168	167	Ö	82	0
123	121	120	x	130	0	171	169	168	´	137	0
124	122	121	y	131	0	172	170	169	`	138	0
125	123	122	z	132	0	173	171	170	^	139	0
126	124	123	{	133	0	174	172	171	¨	140	0
127	125	124	|	134	0	175	173	172	~	141	0
128	126	125	}	135	0	176	174	173	É	70	0
129	127	126	~	136	0	177	175	174	Ö	82	0
130	128	127	DEL	0	0	178	176	175	£	142	0
131	129	128	CLR	0	0	179	177	176	_	143	0
132	130	129	IV	0	0	180	178	177	Ā	65	0
133	131	130	BL	0	0	181	179	178	ā	104	0
134	132	131	IV-B	0	0	182	180	179	°	144	0
135	133	132	UL	0	0	183	181	180	Ç	67	0
136	134	133	IV-U	0	0	184	182	181	ç	106	0
137	135	134	BL-U	0	0	185	183	182	Ñ	81	0
138	136	135	I-B-U	0	0	186	184	183	ñ	120	0
139	137	136		0	0	187	185	184	¡	145	0
140	138	137		0	0	188	186	185	¿	146	0
141	139	138		0	0	189	187	186	ß	147	0
142	140	139		0	0	190	188	187	£	148	0
143	141	140		0	0	191	189	188	¤	149	0
144	142	141		0	0	192	190	189	§	150	0
145	143	142		0	0	193	191	190	±	151	0
146	144	143		0	0	194	192	191	·	152	0

---------- COLLATING SECTION - CONT ----------

ENTRY #	COLL. ENTRY	CHAR CODE	CHAR	SEQ #	MODE PTR	ENTRY #	COLL. ENTRY	CHAR CODE	CHAR	SEQ #	MODE PTR
195	193	192	à	104	0	243	241	240		0	0
196	194	193	é	109	0	244	242	241		0	0
197	195	194	ó	121	0	245	243	242		0	0
198	196	195	ú	127	0	246	244	243		0	0
199	197	196	à	104	0	247	245	244		0	0
200	198	197	é	109	0	248	246	245		0	0
201	199	198	ó	121	0	249	247	246		0	0
202	200	199	ú	127	0	250	248	247		0	0
203	201	200	à	104	0	251	249	248		0	0
204	202	201	è	109	0	252	250	249		0	0
205	203	202	ò	121	0	253	251	250		0	0
206	204	203	ù	127	0	254	252	251		0	0
207	205	204	à	104	0	255	253	252		0	0
208	206	205	è	109	0	256	254	253		0	0
209	207	206	ò	121	0	257	255	254		0	0
210	208	207	ù	127	0	258	256	255		0	0
211	209	208	Á	65	0						
212	210	209	î	113	0						
213	211	210	Ø	82	0						
214	212	211	Æ	65	0						
215	213	212	à	104	0						
216	214	213	í	113	0						
217	215	214	ø	121	0						
218	216	215	æ	104	0						
219	217	216	Å	65	0						
220	218	217	ï	113	0						
221	219	218	Ö	82	0						
222	220	219	Ü	88	0						
223	221	220	É	70	0						
224	222	221	ï	113	0						
225	223	222	ß	125	13						
226	224	223		0	0						
227	225	224		0	0						
228	226	225		0	0						
229	227	226		0	0						
230	228	227		0	0						
231	229	228		0	0						
232	230	229		0	0						
233	231	230		0	0						
234	232	231		0	0						
235	233	232		0	0						
236	234	233		0	0						
237	235	234		0	0						
238	236	235		0	0						
239	237	236		0	0						
240	238	237		0	0						
241	239	238		0	0						
242	240	239		0	0						

---------- UPPERCASE/LOWERCASE SECTION ----------

ENTRY #	U/L ENTRY	CHAR CODE	CHAR	UPPER CASE	LOWER CASE	ENTRY #	U/L ENTRY	CHAR CODE	CHAR	UPPER CASE	LOWER CASE
259	1	160				307	49	208	Â	Â	â
260	2	161	Á	Á	á	308	50	209	î	I	î
261	3	162	Í	Í	í	309	51	210	Ø	Ø	ø
262	4	163	Ó	Ó	ó	310	52	211	Æ	Æ	æ
263	5	164	Ú	Ú	ú	311	53	212	à	À	a
264	6	165	Ä	Ä	à	312	54	213	ì	I	ì
265	7	166	É	É	é	313	55	214	ø	Ø	ø
266	8	167	Ô	Ô	o	314	56	215	æ	Æ	æ
267	9	168	´	´	´	315	57	216	Ä	Ä	ä
268	10	169	`	`	`	316	58	217	ì	I	ì
269	11	170	^	^	^	317	59	218	Ö	Ö	ö
270	12	171	¨	¨	¨	318	60	219	Ü	Ü	ü
271	13	172	~	~	~	319	61	220	É	É	é
272	14	173	É	É	é	320	62	221	ì	I	ì
273	15	174	Ô	Ô	ô	321	63	222	ß	ß	ß
274	16	175	£	£	£	322	64	223			
275	17	176				323	65	224			
276	18	177	Ã	Ã	ã	324	66	225			
277	19	178	ã	Ã	ã	325	67	226			
278	20	179	°	°	°	326	68	227			
279	21	180	Ç	Ç	ç	327	69	228			
280	22	181	ç	Ç	ç	328	70	229			
281	23	182	Ñ	Ñ	ñ	329	71	230			
282	24	183	ñ	Ñ	ñ	330	72	231			
283	25	184	¡	¡	¡	331	73	232			
284	26	185	¿	¿	¿	332	74	233			
285	27	186	¤	¤	¤	333	75	234			
286	28	187	£	£	£	334	76	235			
287	29	188	¤	¤	¤	335	77	236			
288	30	189	§	§	§	336	78	237			
289	31	190	¤	¤	¤	337	79	238			
290	32	191	·	·	·	338	80	239			
291	33	192	à	A	à	339	81	240			
292	34	193	é	E	é	340	82	241			
293	35	194	ô	O	ô	341	83	242			
294	36	195	ù	U	ù	342	84	243			
295	37	196	á	A	á	343	85	244			
296	38	197	é	E	é	344	86	245			
297	39	198	ó	O	ó	345	87	246			
298	40	199	ú	U	ú	346	88	247			
299	41	200	à	A	à	347	89	248			
300	42	201	è	E	è	348	90	249			
301	43	202	ò	O	ò	349	91	250			
302	44	203	ù	U	ù	350	92	251			
303	45	204	ä	Ä	ä	351	93	252			
304	46	205	ë	E	ë	352	94	253			
305	47	206	ö	Ö	ö	353	95	254			
306	48	207	ü	Ü	ü	354	96	255			

---------- MODE SECTION ----------

ENTRY #	MODE ENTRY	TYPE DESCRIPTION
355	1	ONE FOR TWO CHARACTER REPLACEMENT
356	2	ONE FOR TWO CHARACTER REPLACEMENT
357	3	* TERMINATOR WORD *
358	4	ONE FOR TWO CHARACTER REPLACEMENT
359	5	ONE FOR TWO CHARACTER REPLACEMENT
360	6	* TERMINATOR WORD *
361	7	ONE FOR TWO CHARACTER REPLACEMENT
362	8	ONE FOR TWO CHARACTER REPLACEMENT
363	9	* TERMINATOR WORD *
364	10	ONE FOR TWO CHARACTER REPLACEMENT
365	11	ONE FOR TWO CHARACTER REPLACEMENT
366	12	* TERMINATOR WORD *
367	13	TWO FOR ONE CHARACTER REPLACEMENT
368	14	TWO FOR ONE CHARACTER REPLACEMENT

|||||||||| TYPE DETAILS ||||||||||

-- DON'T CARES --

-- ACCENT PRIORITIES --

-- TWO FOR ONE REPLACEMENTS --

ENTRY #	REPLACEMENT	ASCII
13	ß = ss = ss	222

-- ONE FOR TWO REPLACEMENTS --

ENTRY #	REPLACEMENT AND SEQUENCE NUMBER
1	CH = 68
2	Ch = 68
4	LL = 78
5	Ll = 78
7	cH = 107
8	ch = 107
10	lL = 117
11	ll = 117

SWEDISH TABLE

ENTRY #1 -- LENGTH OF COLLATING TABLE = 374
ENTRY #2 -- LENGTH OF MODE TABLE = 20

---------- COLLATING SECTION ----------

ENTRY #	COLL. ENTRY	CHAR CODE	CHAR	SEQ #	MODE PTR	ENTRY #	COLL. ENTRY	CHAR CODE	CHAR	SEQ #	MODE PTR
3	1	0	NUL	0	0	51	49	48	0	48	0
4	2	1	SOH	1	0	52	50	49	1	49	0
5	3	2	STX	2	0	53	51	50	2	50	0
6	4	3	ETX	3	0	54	52	51	3	51	0
7	5	4	EOT	4	0	55	53	52	4	52	0
8	6	5	ENQ	5	0	56	54	53	5	53	0
9	7	6	ACK	6	0	57	55	54	6	54	0
10	8	7	BEL	7	0	58	56	55	7	55	0
11	9	8	BS	8	0	59	57	56	8	56	0
12	10	9	HT	9	0	60	58	57	9	57	0
13	11	10	LF	10	0	61	59	58	:	58	0
14	12	11	VT	11	0	62	60	59	;	59	0
15	13	12	FF	12	0	63	61	60	<	60	0
16	14	13	CR	13	0	64	62	61	=	61	0
17	15	14	SO	14	0	65	63	62	>	62	0
18	16	15	SI	15	0	66	64	63	?	63	0
19	17	16	DLE	16	0	67	65	64	@	64	0
20	18	17	DC1	17	0	68	66	65	A	65	0
21	19	18	DC2	18	0	69	67	66	B	66	0
22	20	19	DC3	19	0	70	68	67	C	67	0
23	21	20	DC4	20	0	71	69	68	D	68	0
24	22	21	NAK	21	0	72	70	69	E	69	0
25	23	22	SYN	22	0	73	71	70	F	70	0
26	24	23	ETB	23	0	74	72	71	G	71	0
27	25	24	CAN	24	0	75	73	72	H	72	0
28	26	25	EM	25	0	76	74	73	I	73	0
29	27	26	SUB	26	0	77	75	74	J	74	0
30	28	27	ESC	27	0	78	76	75	K	75	0
31	29	28	FS	28	0	79	77	76	L	76	0
32	30	29	GS	29	0	80	78	77	M	77	0
33	31	30	RS	30	0	81	79	78	N	78	0
34	32	31	US	31	0	82	80	79	O	80	0
35	33	32	SP	32	0	83	81	80	P	81	0
36	34	33	!	33	0	84	82	81	Q	82	0
37	35	34	"	34	0	85	83	82	R	83	0
38	36	35	#	35	0	86	84	83	S	84	0
39	37	36	$	36	0	87	85	84	T	85	0
40	38	37	%	37	0	88	86	85	U	86	0
41	39	38	&	38	0	89	87	86	V	87	0
42	40	39	'	39	0	90	88	87	W	88	0
43	41	40	(	40	0	91	89	88	X	89	0
44	42	41	)	41	0	92	90	89	Y	90	0
45	43	42	*	42	0	93	91	90	Z	91	0
46	44	43	+	43	0	94	92	91	[	111	0
47	45	44	,	44	0	95	93	92	\	112	0
48	46	45	-	45	0	96	94	93	]	113	0
49	47	46	.	46	0	97	95	94	^	114	0
50	48	47	/	47	0	98	96	95	_	115	0

---------- COLLATING SECTION - CONT ----------

ENTRY #	COLL. ENTRY	CHAR CODE	CHAR	SEQ #	MODE PTR	ENTRY #	COLL. ENTRY	CHAR CODE	CHAR	SEQ #	MODE PTR
99	97	96	`	116	0	147	145	144		0	0
100	98	97	a	117	0	148	146	145		0	0
101	99	98	b	118	0	149	147	146		0	0
102	100	99	c	119	0	150	148	147		0	0
103	101	100	d	120	0	151	149	148		0	0
104	102	101	e	122	0	152	150	149		0	0
105	103	102	f	123	0	153	151	150		0	0
106	104	103	g	124	0	154	152	151		0	0
107	105	104	h	125	0	155	153	152		0	0
108	106	105	i	126	0	156	154	153		0	0
109	107	106	j	127	0	157	155	154		0	0
110	108	107	k	128	0	158	156	155		0	0
111	109	108	l	129	0	159	157	156		0	0
112	110	109	m	130	0	160	158	157		0	0
113	111	110	n	131	0	161	159	158		0	0
114	112	111	o	132	0	162	160	159		0	0
115	113	112	p	133	0	163	161	160		0	0
116	114	113	q	134	0	164	162	161	Á	94	0
117	115	114	r	135	0	165	163	162	Í	102	0
118	116	115	s	136	0	166	164	163	Ó	103	0
119	117	116	t	137	0	167	165	164	Ú	109	0
120	118	117	u	138	0	168	166	165	Ä	95	0
121	119	118	v	139	0	169	167	166	É	100	0
122	120	119	w	140	0	170	168	167	Ö	104	0
123	121	120	x	141	0	171	169	168	´	173	0
124	122	121	y	142	0	172	170	169	`	174	0
125	123	122	z	143	0	173	171	170	^	175	0
126	124	123	{	169	0	174	172	171	¨	176	0
127	125	124	|	170	0	175	173	172	~	177	0
128	126	125	}	171	0	176	174	173	Ë	101	0
129	127	126	~	172	0	177	175	174	Ö	105	0
130	128	127	DEL	0	0	178	176	175	£	178	0
131	129	128	CLR	0	0	179	177	176		179	0
132	130	129	IV	0	0	180	178	177	Ã	97	0
133	131	130	BL	0	0	181	179	178	ã	150	0
134	132	131	IV-B	0	0	182	180	179	°	180	0
135	133	132	UL	0	0	183	181	180	Ç	98	0
136	134	133	IV-U	0	0	184	182	181	ç	151	0
137	135	134	BL-U	0	0	185	183	182	Ñ	108	0
138	136	135	I-B-U	0	0	186	184	183	ñ	164	0
139	137	136		0	0	187	185	184	¡	181	0
140	138	137		0	0	188	186	185	¿	182	0
141	139	138		0	0	189	187	186	¤	183	0
142	140	139		0	0	190	188	187	£	184	0
143	141	140		0	0	191	189	188	º	185	0
144	142	141		0	0	192	190	189	§	186	0
145	143	142		0	0	193	191	190	²	187	0
146	144	143		0	0	194	192	191	·	188	0

---------- COLLATING SECTION - CONT ----------

ENTRY #	COLL. ENTRY	CHAR CODE	CHAR	SEQ #	MODE PTR	ENTRY #	COLL. ENTRY	CHAR CODE	CHAR	SEQ #	MODE PTR
195	193	192	à	148	0	243	241	240		0	0
196	194	193	è	153	0	244	242	241		0	0
197	195	194	ò	161	0	245	243	242		0	0
198	196	195	ù	167	0	246	244	243		0	0
199	197	196	â	146	0	247	245	244		0	0
200	198	197	ê	122	0	248	246	245		0	0
201	199	198	ô	159	0	249	247	246		0	0
202	200	199	û	165	0	250	248	247		0	0
203	201	200	ä	147	0	251	249	248		0	0
204	202	201	ë	152	0	252	250	249		0	0
205	203	202	ö	160	0	253	251	250		0	0
206	204	203	ü	166	0	254	252	251		0	0
207	205	204	ä	149	0	255	253	252		0	0
208	206	205	ë	154	0	256	254	253		0	0
209	207	206	ö	162	0	257	255	254		0	0
210	208	207	ü	168	0	258	256	255		0	0
211	209	208	Ä	93	0						
212	210	209	ï	157	0						
213	211	210	Ø	107	0						
214	212	211	Æ	92	0						
215	213	212	å	145	0						
216	214	213	í	155	0						
217	215	214	ø	163	0						
218	216	215	æ	144	0						
219	217	216	Ä	96	0						
220	218	217	ï	156	0						
221	219	218	Ö	106	0						
222	220	219	Ü	110	0						
223	221	220	É	99	0						
224	222	221	ï	158	0						
225	223	222	ß	136	1						
226	224	223		0	0						
227	225	224		0	0						
228	226	225		0	0						
229	227	226		0	0						
230	228	227		0	0						
231	229	228		0	0						
232	230	229		0	0						
233	231	230		0	0						
234	232	231		0	0						
235	233	232		0	0						
236	234	233		0	0						
237	235	234		0	0						
238	236	235		0	0						
239	237	236		0	0						
240	238	237		0	0						
241	239	238		0	0						
242	240	239		0	0						

---------- UPPERCASE/LOWERCASE SECTION ----------

ENTRY #	U/L ENTRY	CHAR CODE	CHAR	UPPER CASE	LOWER CASE	ENTRY #	U/L ENTRY	CHAR CODE	CHAR	UPPER CASE	LOWER CASE
259	1	160				307	49	208	Á	Á	á
260	2	161	Á	Á	á	308	50	209	Î	I	î
261	3	162	Í	Í	í	309	51	210	Ø	Ø	ø
262	4	163	Ó	Ó	ó	310	52	211	Æ	Æ	æ
263	5	164	Ú	Ú	ú	311	53	212	à	À	à
264	6	165	À	À	à	312	54	213	í	Í	í
265	7	166	É	É	é	313	55	214	ø	Ø	ø
266	8	167	Ó	Ó	ó	314	56	215	æ	Æ	æ
267	9	168	´	´	´	315	57	216	Ã	Ã	ã
268	10	169	`	`	`	316	58	217	ì	I	ì
269	11	170	^	^	^	317	59	218	Õ	Õ	õ
270	12	171	¨	¨	¨	318	60	219	Ü	Ü	ü
271	13	172	~	~	~	319	61	220	É	É	é
272	14	173	É	É	é	320	62	221	ï	I	ï
273	15	174	Ô	Ô	ô	321	63	222	ß	ß	ß
274	16	175	£	£	£	322	64	223			
275	17	176				323	65	224			
276	18	177	Ã	Ã	ã	324	66	225			
277	19	178	ã	Ã	ã	325	67	226			
278	20	179	°	°	°	326	68	227			
279	21	180	Ç	Ç	ç	327	69	228			
280	22	181	ç	Ç	ç	328	70	229			
281	23	182	Ñ	Ñ	ñ	329	71	230			
282	24	183	ñ	Ñ	ñ	330	72	231			
283	25	184	¡	¡	¡	331	73	232			
284	26	185	¿	¿	¿	332	74	233			
285	27	186	¤	¤	¤	333	75	234			
286	28	187	£	£	£	334	76	235			
287	29	188	¤	¤	¤	335	77	236			
288	30	189	§	§	§	336	78	237			
289	31	190	Ω	Ω	Ω	337	79	238			
290	32	191	♭	♭	♭	338	80	239			
291	33	192	á	Á	á	339	81	240			
292	34	193	é	É	é	340	82	241			
293	35	194	ó	Ó	ó	341	83	242			
294	36	195	Ú	U	Ú	342	84	243			
295	37	196	à	À	à	343	85	244			
296	38	197	é	É	é	344	86	245			
297	39	198	ó	Ó	ó	345	87	246			
298	40	199	ú	Ú	ú	346	88	247			
299	41	200	à	À	à	347	89	248			
300	42	201	è	É	è	348	90	249			
301	43	202	ò	Ò	ò	349	91	250			
302	44	203	ù	U	ù	350	92	251			
303	45	204	ä	Ä	ä	351	93	252			
304	46	205	ë	E	ë	352	94	253			
305	47	206	ö	Ö	ö	353	95	254			
306	48	207	ü	Ü	ü	354	96	255			

```
------------ MODE SECTION ------------

ENTRY #  | MODE ENTRY | TYPE DESCRIPTION
---------+------------+-----------------------------------------
  355    |     1      | TWO FOR ONE CHARACTER REPLACEMENT
  356    |     2      | TWO FOR ONE CHARACTER REPLACEMENT

                    |||||||||| TYPE DETAILS ||||||||||

 -- DON'T CARES --

 -- ACCENT PRIORITIES --

 -- TWO FOR ONE REPLACEMENTS --

ENTRY #       REPLACEMENT              ASCII

 1            ß = ss = ss               222

 -- ONE FOR TWO REPLACEMENTS --
```

Your Comments, Please...

Your comments assist us in improving the usefulness of our publications; they are an important part of the inputs used in preparing updates to the publications.

In order to write this manual, we made certain assumptions about your computer background. By completing and returning the comments card on the following page you can assist us in adjusting our assumptions and improving our manuals.

Feel free to mark more than one reply to a question and to make any additional comments.

Please do not use this form for questions about technical applications of your system or requests for additional publications. Instead, direct those inquiries or requests to your nearest HP Sales and Service Office.

If the comments card is missing, please address your comments to:

HEWLETT-PACKARD COMPANY
Desktop Computer Division
3404 East Harmony Road
Fort Collins, Colorado 80525 U.S.A.

Attn. Customer Documentation
Dept. 4231

All comments and suggestions become the property of Hewlett-Packard.