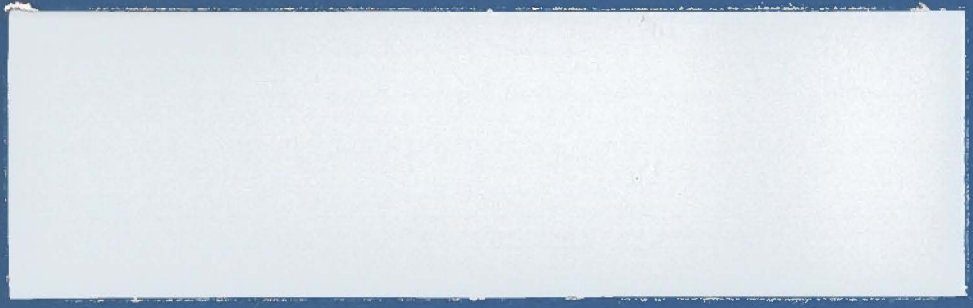
MCM COMPUTERS LIMITED



MCM

M C M S Y S T E M / 9 0 0

U T I L I T I E S M A N U A L

MCM COMPUTERS LTD.
6700 FINCH AVE. W.
SUITE 600
REXDALE, ONTARIO
M9W 5P5.

MCM COMPUTERS LTD.
133 DALTON STREET
KINGSTON, ONTARIO
K7L 4W2

MANUAL NO: 018-0059

REVISION NO: AA

DATE: February 1979

TABLE OF CONTENTS

NOTE The numbers appearing in parentheses after the subsections represent the group number in which they reside on diskette.

To get a directory of any group, simply select the group you are interested in and type '*DIR*'. To get a directory of all the groups and their descriptions, select group 0 and type '*DIR*'.

SECTION ONE

MCM/APL UTILITY FUNCTIONS

DATA INPUT AND EDITING (Group 1)

AIP

Function: Prompt for keyboard input

Syntax: *RESULT ← PROMPT AIP DEFAULT*

Subroutines: None

Description: This function prompts the user for literal input and provides a *DEFAULT* answer. The maximum length of the input plus the prompt must be less than 86 characters.

EDT

Function: Edit a data matrix 1 row at a time

Syntax: *RESULT ← EDT TEXT*

Subroutines: *CKΔ*

Description: The text to be edited is a matrix of either alpha-numeric or numeric data that has a maximum column width (when formatted if numeric) of 81 characters.

Text is displayed on the screen one row at a time with the row number being the leftmost part of the display. Any character may be deleted or altered, spaces may be inserted anywhere on the line and characters may be altered in any order by using the screen editing facilities. When editing numeric text, the edited line is checked for numeric validity and that the number of elements in the row has not been changed. Rows of the matrix are displayed sequentially each time the execute key is depressed. A row may be selected by the user by altering the current row number to the required row number, then depressing the execute key.

If the row number entered is zero or greater than the total number of rows in the matrix *TEXT*, the edit function terminates and the edited result is stored in *RESULT*.

ENI

Function: Prompt for numeric input, provide a default answer and check for numeric validity.

Syntax: *RESULT ← PROMPT ENI DEFAULT*

Subroutines: *CKΔ*

Description: This function prompts the user with *PROMPT* and checks the keyboard input for numeric validity. The prompt together with the default answer must contain less than 85 characters. The *DEFAULT* is displayed to the right of the prompt and will be used as the answer if nothing is entered.

TAK

Function: Perform the same operation as the *APL* primitive ↑, with the addition that if you take a matrix of larger size than the data item you are operating on, padding will be done automatically (blanks for characters and 0's for numerics).

Syntax: *RESULT ← SIZE TAK MATRIX*

Subroutines: *TKΔ*

Description: *SIZE* is a numeric vector specifying the dimensions of the matrix to be taken from data item *MATRIX*.

Example:

```
A←4 4
B←3 3⍴1 2 3

R←A TAK B
```

R is now the matrix:

```
1 2 3 0
1 2 3 0
1 2 3 0
1 2 3 0
```

LISTING OF DATA AND FUNCTIONS (Group 2)

ΔCR

Function: Format and print on the MCP-132 printer: functions, data or entire groups

Syntax: *ΔCR ARGUMENT*

Subroutines: *LM WID PAG BKΔ NPG SOR FNΔ DXΔ*

Description: The *ARGUMENT* should be either a numeric vector of group numbers or a character vector specifying the names of objects. If a numeric vector is specified, the function will print the contents of all the specified groups from the volume mounted on drive 1. If the argument is a character vector containing one or more names separated by spaces, or an array of names, one per row, only these items are printed.

This function can be used to print anything from a single function to the entire contents of a tape or disk. Note: *ΔCR* must reside in group 0 of the AVS drive. It assumes that the user does not have an object with the same name as the right argument, otherwise there will be a name conflict.

XRF

Function: Print function and cross reference

Syntax: *XRF FUNCTION*

Subroutines: *BKΔ DXΔ FNΔ LM NPG PAG SCΔ*

Description: This function lists a function with line numbers and also prints a cross reference of all the object names used in the program. The cross reference produced on the printer is an alphabetical list of all objects used in the function along with its type (local or label) and the line numbers where the object is used.

TRC

Function: Trace the execution of a function.

Syntax: *LINE-NUMBERS TRC FUNCTION*

Subroutines: *FN∆*

Description: This function executes the function(right argument) and traces the lines given in the left argument. The value of the lines are printed on the printer each time they are executed.

Restrictions:

- -traced functions must be niladic
- -user must insure no conflicts exist between locals and labels in *TRC* and the traced function
- -each traced line must have a result
- -subfunctions are not traced
- -this function may not properly execute imbedded branches

COPY/BACKUP (Group 3)

COP

Function: To copy groups to another drive.

Syntax: *DRIVE-NUMBER COP GROUPS*

Subroutines: None

Description: This function copies the groups from the AVS drive (drive 1), to another drive. The right argument is either a numeric list of valid group numbers that reside on the volume on drive 1, or, an empty vector (ie. ''), in which case all the groups on drive 1 are copied to the drive specified by the left argument.
NOTE: Copy must reside in group 0 on drive 1.

The function *COP* must reside in group 0 and group 0 must be selected to run it. The routine cannot copy an object called '*COP*'.

DISK RECOVERY (Group 4)

RED

Function: To recover information from a diskette where the directory has been destroyed.

Syntax: *RED BLOCK-NUMBER*

Subroutines: *QXΔ*

Description: Before starting, initialize a disk to receive the contents of the disk being recovered. Write *RED* and *QXΔ* into group 255 of this new disk and close it. Clear the workspace.

Mount the dead disk on drive 1, initialized disk on drive 2. Read *RED* and *QXΔ* from drive 2 and execute *RED* 1. The program will read the contents from drive 1 and write it onto drive 2. A printer is used to show the user the object name just recovered, the block number where it was found, and the group the object was assigned to.

The disk recovery may fail if it encounters a damaged block. In this case the system will freeze. Pull the plug from the back of the computer, plug it in again and restart it. Read *RED* and *QXΔ* from drive 2 and execute *RED N* where *N* is the last block number reported on the printer plus 1. If it fails immediately, increment this number by one and repeat.

SYSTEM DIAGNOSTICS (Group 5)

ROM

Function: Check the read only memory

Syntax: *ROM*

Subroutines: None

Description: This function checks the ROM (read only memory) of the computer. The output is a list of checksum numbers and their total. If you do not know what they should be, call your field service representative with your serial number and model number. Supplied with this information, you will be able to tell is there is any error in the Read Only Memory.

RAM

Function: Check random access memory.

Syntax: *RAM*

Subroutines: None

Description: This function checks the random access memory of the computer. The output from this routine is the number where the first RAM test failure is. Normally it should be at the end of the memory. Check with your field service representative as to the numbers that should appear for your computer.

STA

Function: Display status of omniport device.

Syntax: *STA UNIT-ADDRESS*

Subroutines: None

Description: This function displays on the screen, the status of the device specified by the right argument. The device can be any attached omniport peripheral (ie. printer, disk drive, etc..). The right argument can be the device unit address (ie. *STA* 1) or the status from the answer back code (ie. *STA* □*OU* ι0).To check the status of the file subsystem, the argument is the unit number of the drive plus 200 (ie. AVS drive is 201).

SPA

Function: Gives the available space on a diskette.

Syntax: *SPA*

Subroutines: *FSΔ*

Description: This routine asks you for the unit that the disk you want to analyse resides on. It will tell you how many blocks are being used, the last block number that is being used, and the percentage of space used. If the number of blocks being used is significantly smaller than the last block number, it is advisable to make a copy of the disk and compress the space.

MATRIX INVERSION AND DOMINO FUNCTIONS (Group 6)

INV

Function: Matrix Inversion

Syntax: *RESULT ← INV MATRIX*

Subroutines: None

Description: This function uses the Gauss-Jordan method to find the inverse of the non-singular square matrix.

MMD
DMD

Function: Matrix inverse and divide.

Syntax: *RESULT ← MMD MATRIX*
RESULT ← MATRIX-1 DMD MATRIX-2

Subroutines: *LS*

Description: These two functions are to substitute for the primitive operator 'DOMINO'.

MCP-132 PRINTER UTILITIES (Group 7)

EHT

Function: Edit MCP-132 printer control table

Syntax: *EHT*

Subroutines: *ENI AIP CK4*

Description: This function allows the user to edit the control table for the MCP-132 printer. The parameters that can be changed and their range are:

characters per inch(1 to 60)
lines per inch(1 to 60)
print width(30 to 132)
left margin(0 to 132)
indent(for continued lines)(0 to 15)
continuation character
print lines per page(0 to 90)
skip lines end of page(0 to 10)

LM

Function: To set left margin on MCP-132 printer.

Syntax: *LM MARGIN*

Subroutines: None

Description: This function sets the left margin on the MCP-132 printer to the numeric value given by the right argument. □*OU* must be pointing to a MCP-132 printer(ie. □*OU* □*YA* 66).

NPG

Function: Ejects the paper in the MCP-132 printer to the top of a new page.

Syntax: *NPG*

Subroutines: None

Description: This function advances the paper in the MCP-132 to the top of the next page. Before this function is used, paging must be started with the *PAG* function. Once this function has been used to advance the paper to the top of a page, it is then possible to simply continue, or to issue a heading if one is desired.

PFT

Function: Prints a footing and then ejects to the top of a new page

Syntax: *PFT FOOTING*

Subroutines: None

Description: This function advances to the bottom of the current page, prints the right argument, and then advances to the top of the next page. It is useful when the page number or some other information is to be placed at the bottom of each page. Paging must have been turned on by the *PAG* function before this function is used.

PAG

Function: Turns paging on by establishing the logical and physical size of the page, and setting the line and page counters to zero.

Syntax: *RESULT ← PAG FORMAT*

Subroutines: None

Description: The right argument of *PAG* is a two-element vector. The first element of the vector specifies the number of lines to be printed on the page (the logical page size), while the second element indicates the physical length of the page in lines (the physical page size). For example, standard 11 inch paper has an area for printing which is 10 inches long. At 6 lines per inch this yields a logical page size of 10 X 6 = 60 (indicating that there are 60 lines available for actual printing). Since the paper is 11 inches from fold to fold, the physical page size is 11 X 6 = 66.

This function sets up the MCP-132 printer control tables based on the values given in the right argument and then sets the page and line counters to zero. For this reason it is important that the paper in the printer be at the top of a page when the *PAG* function is used. In most cases it is useful to issue a message on the display asking the user to position the paper.

The *RESULT* produced by the function is the previous setting for the logical and physical page size. The system function □*PC SETTINGS* is used for paging control. A right argument of ⍳0 will return the present value of the page and line counters. A right argument of a two-element vector, will set the counters to the values of the vector.

PRT

Function: Prints on the MCP-132 printer without altering the existing communications subsystem tables.

Syntax: *PRT OUTPUT*

Subroutines: None

Description: This function is used to print on the MCP-132 when some other output unit is already selected and set-up. This function saves the current communications subsystem output tables, sets up the MCP-132 printer for standard output, prints the right argument and then restores the output tables that were in effect when it was called.

TIT

Function: Prints a title in dot matrix form on the MCP-132 printer.

Syntax: *OFFSET TIT TITLE*

Subroutines: *POS PA PTA PIT ROL BGA*

Description: This function positions the print head to the position given by the left argument and then prints the right argument in large dot matrix form. The left argument should be a two element numeric vector. It is used to specify the offset, in inches, from the current position of the print head to the upper left corner of the title to be printed. The right argument can be any character vector but it should not be so long as to cause the printer to run past the right margin.

The exact size of the printed title is controlled by the setting of the variables *HEI* and *CPI* in line 6 of the function. These can be changed to provide different size titles. The default values are for a height of one quarter inch (.25) and 5 characters per inch.

WID

Function: Alters the print width, the overflow character (optional), and the number of spaces the overflow line is to be indented (optional).

Syntax: *RESULT←WID FORMAT*

Subroutines: None

Description: The right argument can consist of 1 to 3 integers. The last two are optional. The first argument specifies the width of the ouput in characters. The second argument is the ⎕IO value of the overflow character displayed at the end of the line (the value must be between 128 and 130). The third argument is the number of spaces you want to indent on a continuation line (value must be between 0 and 15).

PLOTTING ON THE MCP-132 (Group 8)

BOX

Function: Prints a box of arbitrary dimensions.

Syntax: *OFFSET BOX SIZE*

Subroutines: *PΔ PTΔ POS*

Description: Note the left and right arguments must be two-element numeric vectors. This function draws a box of arbitrary size at an arbitrary location.

The left argument should be a two-element numeric vector. It is used to specify the offset, in inches, from the current position of the print head to the lower left corner of the box to be printed. The first element of the vector gives the number of inches that the print head should be moved left or right, while the second element indicates how many inches the print head should be moved up or down before printing the box (positive directions are left and up).

The right argument is used to specify the size of the box in inches. The first element specifies the width while the second element specifies the height.

DOW

Function: Rolls paper on MCP-132 printer down.

Syntax: *DOW LINES*

Subroutines: None

Description: This function rolls the paper in the MCP-132 down by the number of lines specified in the right argument and adjusts the line counter to reflect the current position on the page. This function will not move past the start of the current page. If the argument is large enough that it would move past the beginning of the page, then this function will position the print head on the first line of the page.

EQU

Function: Evaluates an equation for plotting.

Syntax: *RESULT ← EQU*

Subroutines: None

Description: This function requests that the user to input the equation for evaluation. After asking the user to define the independent variable, the function then proceeds to evaluate the expression and shapes it into an N by 2 matrix suitable for use by the function *PLOT*. The function must be entered as a valid *APL* statement in the form *'Y ← EXPRESSION X'*. The independent variable *X* must be defined as a vector, of points(ie. *X ← ⍳25* which generates 25 points of values).

HLI

Function: Prints a horizontal line of arbitrary length.

Syntax: *OFFSET HLI LENGTH*

Subroutines: *POS PΔ PTΔ*

Description: The right argument is a number which specifies the length of the line in inches. The left argument is a two-element numeric vector. It is used to specify the offset, in inches from the current position of the print head. The first element gives the number of inches that the print head should be moved left or right, while the second element indicates how many inches the print head should be moved up or down, before the printing of the line begins.

PIT

Function: To change the number of characters printed horizontally per inch on the MCP-132 printer.

Syntax: *RESULT ← PIT CHARACTERS-PER-INCH*

Subroutines: None

Description: The right argument is either a number in the range of 1 to 60 or an empty vector (⍳0). If it is a number in the range of 1 to 60, then the number of characters to be printed per inch is set to that number. If it is an empty vector, then the pitch setting is not changed. In either case the *RESULT* is the pitch setting that was in effect when this function was called.

Since the printer moves horizontally in 1/120 inch steps, it is not possible to produce printing at all of the character spacing between 1 and 60 characters per inch. Only those which require spacing that is a multiple or 1/60 inches can be produced exactly. Those which can be produced exactly are: 1 2 3 4 5 6 10 12 15 20 30 and 60 characters per inch. If pitch receives a request for a character spacing other than one of the above, it selects the next lower possible spacing.

Although not all spacings are possible, the most common ones, PICA (10) and ELITE (12), are available. The default value is 10 characters per inch.

PLO

Function: Scales and plots data on the MCP-132.

Syntax: *PLO CO-ORDINATES*

Subroutines: *POS PΔ PTΔ CEN VLI HLI*

Description: This function accepts as the right argument, an N by 2 array of co-ordinates to correspond to the height and width of the plot. In addition to the right argument, this function will request that the user input the plot character, the width and the height (in inches) of the plot, and titles for the X and Y axis.

Note: The function *EQU* is very useful for producing the values to be plotted.

POS

Function: Moves the print head an arbitrary number of inches from its current position.

Syntax: *POS OFFSET*

Subroutines: *PΔ PTΔ*

Description: This function simply moves the print head to the position specified in the right argument, which must be a two-element numeric vector. It is used to specify the offset in inches from the current position of the print head. The first element gives the number of inches that the print head should be moved left or right, while the second element indicates how many inches the print head should be moved up or down.

PΔ

Function: Prints any number of points at arbitrary positions. This is a high-speed routine that has most of the abilities of direct control.

Syntax: *PLOT-CHARACTER PΔ CO-ORDINATES*

Subroutines: *PTΔ*

Description: The left argument is the plot character and the numbers from the right argument are used in pairs to direct the motion of the print head. This function is used to move the print head around and print a plot character. The numbers from the right argument are then used in pairs to specify the X-Y locations at which the character is to be printed.

The right argument can be any shape but the pairs are taken from it in ravel order. The first number of the pair specifies the X location and the second number specifies the Y location. If, for example, the first plot character is to appear one inch to the right of and one inch above the starting (0,0) point, then the first two numbers of the right argument should be 120 96 (the horizontal increments are specified in 1/96 inch increments, however the MCP-132 printer has a vectical resolution of 1/48 inches, therefore when specifying vertical motion directly in terms of increments, it is important that the numbers should be even). Two things are important here. First note that positive numbers specify locations to the right of or above the center while negative numbers specify locations to the left of or below the center.

If the next character was to appear two inches below and two inches to the left of the center, then the next two numbers in the right argument should be -240 -192. This function prints a plot character for each pair of numbers in the right argument, except for the last pair, for which it simply moves the print head to the specified position without printing a character.

ROL

Function: To change the number of lines per inch to be printed on the MCP-132 printer.

Syntax: *RESULT ← ROL LINES-PER-INCH*

Subroutines: None

Description

The right argument is either a number in the range 1 to 48 or an empty vector (⍳0). If it is a number in the range 1 to 48, then the number of lines to be printed per inch is set to the argument. If it is an empty vector, then the roll setting is not changed. In either case the *RESULT* is the roll setting that was in effect when this function used.

Since the printer moves vertically in 1/48 inch steps, it is not possible to produce printing at all of the line spacings between 1 and 48 lines per inch. Only those which require spacing that is a multiple of 1/48 inches can be produced exactly.

Those which can be produced exactly are... 1 2 3 4 6 8 12 16 24 and 48 lines per inch. If *ROL* receives a request for a line spacing other than one of the above, it selects the next lower possible line spacing. The default value is 6 lines per inch. The most common ones are shown below:

8 compressed with no overlap

6 standard computer spacing

4 one and one half spacing

3 double spacing

2 triple spacing

UP

Function: Rolls paper up on MCP-132 printer.

Syntax: *UP LINES*

Subroutines: None

Description: This function moves the paper in the MCP-132 up by the number of lines specified in the right argument and also adjusts the line counter to reflect the actual position on the page. This function will not move the user off the current page. If the argument is large enough that it would move to the next page, then this function moves only as far as the last line on the current page.

VLI

Function: Prints a vertical line of a given length.

Syntax: *OFFSET VLI LENGTH*

Subroutines: *POS PΔ PTΔ*

Description: This function is similar to the *HLI* function, but it draws a vertical line of the length specified by the right argument starting at the offset specified by the left argument, from the current position of the print head.

RUN FUNCTION (Group 9)

RUN

Function: A table driven form of calling functions suitable for running a system.

Syntax: *RUN*

Subroutines: *ΔLΔ DNU* (*ΔLΔ* is a global variable)

Description: This function can be used as a control function, stored in group 0 to control access of groups for a large system. It accepts input from the user and then selects either a group or program according to the user's instructions. A character table (*ΔLΔ*) containing the system name, *APL* function name, and group, must be in group zero and conform to the following format:

System Name (col. 1 to 6)
APL Function Name(col. 7 to 9)
Group Number(col. 10 to 12)

The variable *ΔDT* is needed to run this function because it uses it as the last date entered. The function asks for and sets current date in two formats:

ΔDA - Month. Day Year (character)
ΔDT - YY MM DD (numeric)

The function *RUN* requires the subroutines *CKΔ* and *SCΔ*. It also uses a table called *MΔ* which is a list of abbreviated month names, an a variable called *ΔDT* which is a 3 element vector for the previous date used (format is YY MM DD).

DEVICE SUPPORT UTILITIES (Group 10)

BAU

Function: To set the baud rate of an external input/output device.

Syntax: *BAU RATE*

Subroutines: None

Description: This function sets the baud rate only if □*IN* or □*OU* are pointing to a communications interface. The rate specified must be a numeric scalar value, which is the baud rate in bits per second (ie. 110 134.5 300 600 1200 2400).

SET

Function: To setup communications interface to an RS-232C device.

Syntax: *SET TABLE*

Subroutines: (tables - *YCI*, *YCO*, *YEI*, *YEO*, *YAI*, *YAO*)

Description: This function sets up the communications control tables for interfacing to another computer. Valid transmission tables are: *C* (correspondence), *E* (EBCDIC), *A* (ASCII).

XFE

Function: To transfer data to an RS-232C device.

Syntax: *RESULT ← XFE DATA*

Subroutines: None

Description: Sets the input and output control pointing to a RS-232C device. Data is transferred to the device and *RESULT* contains the input from the device. If a soft interrupt is executed while waiting for input from the device, *RESULT* will contain all data transferred up to the interrupt.

CEP

Function: Sets up the output table for the MCP-712 or the MCP-713 and selects it as the output device.

Syntax: *RESULT ← CEP*

Subroutines: *YCP* (printer table)

Description: This function will set up the output table for the system so that all ⎕ output will be in the proper format for the 712 and 713 printers. The function also selects the printer as the output device. If you deselect the printer and direct output to another device (ie. the screen), you must run this routine again before you direct ⎕ output to the printer.

The result is the answer back code from the printer. The normal answer back is 6 70 153. A value of 0 0 0 or 6 0 0 indicates the printer is not connected. A value of 6 70 140 indicates that the select button has not been pressed.

NOTE: If the 712 or 713 have the *APL* character set, this function will load the complete character set. If you have a 712 or 713 with an ASCII character set, the numbers and letters will properly mask from *APL* to ASCII. There are two other representations of the rest of the *APL* character set on ASCII only printers besides what *CEP* produces. See *HAS* and *MNE*.

HAS

Function: Produces a print table for an ASCII only printer where the *APL* characters other than numbers and letters are represented wherever possible by the ASCII equivalent, and if there is no equivalent, it is represented by a '#'.

Syntax: *HAS*

Subroutines: *CC* (a variable).

Description: To load the hash character set, this routine should be run immediately after running *CEP*.

MNE

Function: Produces a print table for an ASCII only printer where the *APL* characters other than numbers and letters are represented by a mnemonic. The mnemonic is a $ followed by 2 descriptive letters to relate to the symbol.

Syntax: *MNE*

Subroutines: *CC* (variable)

Description: To load the mnemonic character set, this routine should be run immediately after running *CEP*.

VIDEO SCREEN SUPPORT (Group 11)

ASC
BIL

Table:

ASC is the ASCII standard North American typewriter keyboard mask table.

BIL is the standard North American Bilingual keyboard mask table.

BIG

Function: Output large letters to the screen.

Syntax: *MASK,CHAR BIG CHARACTER-STRING*

Subroutines: *BC∆*

Description: The left argument is a numeric vector. The first element specifies which character set to use (0 for the standard APL set; 1 for the alternate set).

The second element (optional) gives the ⎕Y value of the character to be used to display the string. If this element is not included, the default for the display character will be the character it is displaying.

The right argument is the character string to be displayed. It has a maximum length of 20 characters.

CPL

Function: Plot character at a position on screen.

Syntax: *CHARACTER CPL X-Y-POSITION*

Subroutines: None

Description: The left argument gives the character that is to be plotted. The right argument is a numeric vector which gives the X Y position on the screen that the character is to be displayed at.

HIL
REV
RVD

Function: All three functions change the alternate character set to a new kind.

Syntax:
HIL - highlighted *APL* set
REV - reverse *APL* set
RVD - reverse alternate set

Subroutines: None

Description: All three of the functions create a new character set and place it in the alternate character set in memory. HIL gives a highlighted version of the standard *APL* character set. *REV* gives a reverse video (black characters on a white background) version of the standard *APL* character set. *RVD* gives a reverse video version of the current alternate character set.

HIS

Function: Display a histogram of numeric data

Syntax: *HIS DATA*

Subroutines: None

Description: This function draws an X and Y axis and plots a histogram on them. There are 250 units vertically (Y) and 90 units horizontally (X). Therefore, you can give as the right argument a vector of up to 90 elements, each with values from 0 to 250.

PR

Function: Output data starting at X-Y on screen.

Syntax: *X-Y POSITION PR DATA*

Subroutines: None

Description: This function will output character or numeric data using the left argument as the starting position on the screen. The function checks that there is enough room to display the data, and if there is not, it will return an error message.

EDI

Function: To edit a character in the alternate character set.

Syntax: *EDI CHARACTER*

Subroutines: None

Description: This routine allows you to edit the shape of a character on the screen in the alternate character set. This is done by taking a copy of what the character looks like in the standard character set and putting it in a matrix in the upper right of the screen. You can then move the cursor entering '.' where you do not want dots and 'X' where you do want them. Your character shape is defined by the 'X's. When you are done, hit execute, and a row of the character you have formed will be displayed under the matrix you were editing.

SAV

Function: Return a copy of the current alternate character set.

Syntax: *RESULT ← SAV*

Subroutines: None

Description: This function saves the alternate character table by assigning it to *RESULT*. It can then be reloaded by using *LOA*.

LOA

Function: Load a new alternate character set.

Syntax: *LOA ALTERNATE-CHARACTER-SET*

Subroutines: None

Description: This function takes the alternate set given in the right argument and loads it into the alternate character set in memory.

RES

Function: Restore alternate character set to the standard *APL* character set.

Syntax: *RES*

Subroutines: None

Description: Executing this function will restore the alternate character set to the standard *APL* character set.

SORT FUNCTIONS (Group 12)

SOR

Function: Sort an alpha matrix in ascending order.

Syntax: *RESULT ← SOR MATRIX*

Subroutines: *GAΔ*

Description: This function sorts the alpha-numeric matrix into ascending sequence. Note that blanks sort high and that any dimension is limited to 255.

STN

Function: To sort an alpha matrix in ascending sequence beginning at a specified column.

Syntax: *RESULT ← BEGINNING-COLUMN STN MATRIX*

Subroutines: *GAΔ*

Description: This function sorts the alpha-numeric matrix into ascending sequence beginning at the specified column number. All characters to the left of the specified column are ignored. Note that blanks sort high and any dimension is limited to 255.

SECTION TWO

SYSTEM FUNCTIONS

Group 13

General Description

MCM/APL has facilities to execute system (intrinsic) functions. The purpose of these functions is to provide routines that are impractical or impossible to implement in *APL* code. For example, special input/control drivers, character table search or test, and data checking are useful functions which fall into this category.

System functions are written in machine code, therefore they execute much faster than an equivalent *APL* function. Because of the complexity of writing, debugging and interfacing them into the system, these functions can only be implemented by MCM Computers Ltd. staff.

System functions are executed using the *MCM/APL* quad function ⎕ZZ. The general call structure syntax is:

RESULT ← *ARGUMENT* ⎕ZZ[*INDEX*] '*SYSTEM-FUNCTION*'

RESULT, *ARGUMENT* and *INDEX* are optional for some system functions. If the system function is not a valid system function, a *SYNTAX ERROR* is issued.

SYSTEM FUNCTIONS:

*BK*Δ - converts vector with delimiters to a table
*UB*Δ - converts a table to a vector with delimiters
*SC*Δ - do an alpha search on a matrix
*CK*Δ - edit check on an alpha string
*FN*Δ - get a line or total length of a function
*SZ*Δ - give size of an object in bytes
*FS*Δ - report file space used
*QX*Δ - recover objects from a volume with a bad directory
*CP*Δ - compress data for storage
*XP*Δ - expand data for retrieval
*PT*Δ - plot a curve from a set of coordinates
*BG*Δ - produce big letters
*GA*Δ - generate index vector for sorting alpha matrix
*DT*Δ - converts Day-Month-Year from characters to numbers and vice versa
*GS*Δ - pack objects together for storage on disk and give them a key name
*DN*Δ - converts date to a number
*ND*Δ - converts a number to a date

BK∆

Function: Builds a table from a character vector by using a specified break character to separate the rows.

Syntax: *RESULT ← VECTOR ⎕ZZ[⎕Y'CHAR'] BK∆*

Description: The left argument is the character vector to be broken into a table. The *INDEX* is the break character which separates the elements in the left argument. The result will have the number of columns needed to accomodate the largest element in the left argument. The number of rows in the result will correspond to the number of elements in the left argument.

There are two modes of operation for this function. The first mode uses an index of [⎕*Y 'BREAK CHARACTER'*]. In this mode each occurrence of a break character forces a new row to be added to the result. Therefore, the appearance of two break characters together will result in a blank line being added. The second mode uses an index of [128+⎕*Y 'BREAK CHARACTER'*]. In this mode, contiguous break characters are treated as a single break character. This mode is especially useful in creating words from character strings where there may be one or more spaces separating the words. Note: If no index is used, this is equivalent to [128+⎕Y' '] (using a space or multiple spaces as break characters).

Examples:

EG. 1 *'A,BC,,G,DEF'* ⎕ZZ[⎕Y',',] *BK*Δ

Returns the 5 by 3 array:

```
A
BC

G
DEF
```

EG. 2 *'A/BC////LOB'* ⎕ZZ[128+⎕Y'/'] *BK*Δ

Returns the 3 by 3 array:

```
A
BC
LOB
```

EG. 3 *'BCDEF SMITH AND JONES'* ⎕ZZ *BK*Δ

Returns the 4 by 5 array:

```
JONES
AND
SMITH
BCDEF
```

UB∆

Function: Break up a table into strings using a user defined character as a delimiter.

Syntax: *RESULT ← MATRIX ⎕ZZ[DELIMITER] UB∆*

Description: The character matrix is transformed into a vector with the trailing blanks from each row dropped. A user defined delimiter separates the rows in the vector returned in *RESULT*.

Example:

MAT is the 3 by 4 character matrix:

```
BELL
HILL
HIT
```

```
R←MAT ⎕ZZ['/'] UB∆
```

R is now '*BELL/HILL/HIT*'

Note: This function is the inverse of *BK∆*.

SC∆

Function: To apply a scan control argument to each row of an alpha matrix and return an index vector indicating rows that meet the conditions specified in the control argument.

Syntax: *RESULT ← ALPHA-MATRIX ⎕ZZ SC∆*

Description: The scan control vector (*SCV*) must be in the logical workspace when this function is executed or an *SCV ERROR* will result. The scan control vector can contain an arbitrary number of control fields. The syntax of each control field is as follows:

<LEN>(<RO><STRING>[<LO><RO><STRING>..]..)

where

<LEN> is a decimal number indicating the logical length of the control field, and must be in the range 1 - 127

<RO> is any one of the relational operators < ≤ = ≥ > ≠

<STRING> is any string of characters, but must be equal in length to *<LEN>*

<LO> is one of the logical operators ∨ ∧

and square brackets indicate portions of the control field which may be omitted.

Each control field specifies the conditions that must be met for the next *<LEN>* characters. The control fields are processed from left to right, and each one is passed against the next *<LEN>* characters from each row of the argument. The sum of the logical lengths of all the control fields must be less than or equal to the number of columns in the argument or an *SCV FORMAT ERROR* will result.

Example: EG. 1 *SCV*←'(3)=*ABC*(5)≠00000'

This control vector will return the index of each row of an N by 8 (or wider) matrix in which the first three characters are *ABC* and the next five characters are not equal to 00000. An arbitrary number of conditions may be given in each control field. If, in the above example, we had wanted the first three characters to be *ABC* or *XYZ* or *WOW*, then the appropriate control vector would be....

SCV←'(3)=*ABC*∨=*XYS*∨=*WOW*(5)≠00000'

EG. 2 It is possible to do no tests on a control field, which has the effect of causing that many characters in the matrix to be ignored. Following the example, if we had wanted to ignore the first three characters then

SCV←'(3)(5)≠00000'

would do. A control field length of zero is valid but generally meaningless. *SC*Δ treats vectors as an N by 1 matrix.

CK∆ Function: To check if alphabetic string is executeable.

Syntax: *RESULT ← CHAR-VECTOR ⎕ZZ[INDEX] CK∆*

Descripton: The character vector may be up to 85 characters in length. The result is a boolean scalar indicating the presence or absence of errors in the string.

0 means no errors found
1 means at least one error found

The *INDEX* specifies the type of check to be done and is outlined below. If the index is omitted, a default index of ⎕*IO*+3 is assumed.

⎕*IO*+0 - no check: any string accepted
⎕*IO*+1 - A to Z, 0 to 9 & blank accepted
⎕*IO*+2 - A to Z and blank accepted
⎕*IO*+3 - any *APL* scalar or vector
⎕*IO*+4 - integer numeric and blanks
⎕*IO*+5 - Date: If the field is 6 long, it must be of the form DDMMYY. If it is 8 long, it must be DD/MM/YY. The year can be any two digits. The month must be number from 01 to 12. The day must be from 01 to the maximum days for the month. A leap year is any year divisible by four.

To meet the criteria for an index of ⎕*IO*+3 the argument must be a scalar or a vector. It must be less than or equal to 85 characters long and contain at least one number. Each number must contain between 1 and 75 digits, not including the exponent. For each number the value of the exponent plus the number of digits in the integer part must not exceed 75. Each number must conform to the *APL* syntax for numbers.

FN∆

Function: Get a line or its length from a function.

Syntax:
1. *RESULT* ← *FUNCTION* ⎕ZZ *FN∆*
2. *RESULT* ← *FUNCTION* ⎕ZZ[*LINE*] *FN∆*

Description: *FUNCTION* must be a character string or character variable containing the name of a function.

1. The *FUNCTION* is interrogated to determine the number of characters in each line. In this case, the *RESULT* is a numeric vector representing the character count in each function line.

2. Line number *LINE* of the *FUNCTION* is returned as the result. The format of the result is ...

[LINE-NUMBER] LINE TEXT

The brackets, line number, and separating spaces always occupy six locations. This function is useful in listing functions or in determining the size of a function.

SZ∆

Function: Return the size in bytes of an object.

Syntax: *RESULT* ← *OBJECT* ⎕ZZ *SZ∆*

Description: Given the name of an object, *SZ∆* will return the exact size of it in bytes.

FS∆

Function: Reports file space used.

Syntax: *RESULT* ← ⎕ZZ[*UNIT-NO*] *FS∆*

Description: The result is a two-element vector representing the number of blocks used on the file, and the highest block number used respectively. If the second element of the *RESULT* is much greater than the first one, the file space is fragmented and a file backup or copy should be performed.

QX∆

Function: Recover objects from a volume where the directories have been destroyed.

Syntax: *RESULT ← BLOCK NUMBER ⎕ZZ QX∆*

Description: When the directory has been destroyed, the system function *QX∆* can be used to recover objects. *BLOCK NUMBER* is a numeric that contains the block where the search is to be started.

The *RESULT* is a two-element vector where:

RESULT[1] is next block to be searched
RESULT[2] is the group number where the object was found. The object name appears in *⎕VA* or *⎕FN*. At the end of the disk, a disk error will be issued. This function is used with the utility function *RED*.

CP∆
XP∆

Function: To compact like characters from data into a format for efficient storage on disk or tape. Expand is provided to restore the data to its original format.

Syntax: *DATA ⎕ZZ CP∆*
DATA ⎕ZZ XP∆

Description: It can be used for compacting descriptive data which usually contains a large number of blanks, or numeric data where sometimes there is a large number of zeros. A good illustration of this would be storing whole screens of information. For example, data to fill a 24 by 80 column CRT requires 1924 characters. If this character array is entirely blank, it would compact to 28 bytes. If it contained 100 characters representing account number, name, and address, the compacted form would occupy approximately 136 bytes.

For character data, the function defaults to compacting spaces but an override is provided so any character, other than zero (0), can be compacted out of the data. For numeric data the default is zero. Again the user may specify any integer in the range 1 to 255.

EG. 1
```
V ← 20 20⍴ 2
V ⎕ZZ[2] CP⍙
```

The integer 2 is compacted from the data *V*. The character compacted out (2 in this example) is saved within the compacted format and subsequently restored by expand.

```
V ⎕ZZ XP⍙
```

Rules:

1. The character compacted must occur in three successive bytes before it is compacted. Maximum data compaction is 127 to 1.

2. Compaction is disabled if the result is larger than the original data (there is an overhead of 5 + Rank_of_Data_Compacted bytes).

3. The data produced by compact is not valid *APL* data. Any attempt to access this data before it is expanded results in a *DOMAIN ERROR*.

4. Compaction accepts any argument type but it only attempts to compact character or numeric data.

5. Expansion accepts any argument type but it only expands valid compacted data.

6. Zeros (0) cannot be compressed from character data.

Using the example *V* ← 50 50 ρ 0 which normally occupies 2504 bytes, the compaction/expansion process takes approximately 0.2 seconds. The compacted form of this data occupies only 32 bytes. If this compacted data were to be written and then read from disk, there would be an overall speed improvement of over 100 percent. In general, the increase in read/write speed is proportional to the decrease in data size through compaction.

Sample Syntax:

DATA ⎕ZZ *CP*Δ
- compacts blanks from data

DATA ⎕ZZ[⎕*Y*'*C*'] *CP*Δ
-compacts character *C* from data

NUMERIC ⎕ZZ *CP*Δ
-compacts zeros from numeric data

NUMERIC ⎕ZZ[1] *CP*Δ
-compacts the integer 1 from data

CDATA ⎕ZZ *XP*Δ
-expands compacted data

*PT*Δ

Function: This function is used to move the print head or to plot a curve under control of an X Y coordinate array.

Syntax: *RESULT* ← *POINTS* ⎕ZZ[*PLOT-CHARACTER*] *PT*Δ

Description: The *POINTS* can be any shape but the pairs are taken from it in ravel (,*POINTS*) order. The first number of the pair specifies the X location and the second number specifies the Y location. If, for example, the first plot character is to appear one inch to the right of and one inch above the starting (0,0) point, then the first two numbers of the points should be 120 96. If the second plot character is to appear an inch above the axis and an inch to the left of it, then the next two numbers in the argument should be -120 and 96. Three things are important here:

1. Positive numbers specify locations to the left or below the center.

2. The first number (X coorindate) is given in gradients of 120 units per inch and the second number (Y coorindate) is given in gradients of 96 units per inch.

3. The numbers must be even.

This function prints a plot character for each pair of numbers in the argument, except for the last pair, for which it simply moves the print head to the specified position without printing a character. This allows the print head to be returned to its starting position simply by including (0,0) as the last pair of numbers. It is also important to note that if the last pair of numbers is not (0,0) then the print head will be left at that location when the function returns. This feature allows the function to be used simply to move the print head some arbitrary amount.

BG∆

Function: Produces large letters.

Syntax: *RESULT ← TEXT* ⎕ZZ*[CHARACTER] BG∆*

Description: The function *BG∆*, produces large letters for decorative purposes on reports. The *TEXT* is any valid *APL* character string that the user wishes to have in large letters. The character is a single element whose value may be any single valid *APL* character in quotes or its ⎕*Y* equivalent value. If the character is not specified, the system will default to using each character itself as the character to form the large letter. The result is a N by 6 column by 7 row character array where N is the number of characters in the string.

GA∆

Function: Generate an index vector to allow sorting of an alpha matrix.

Syntax: *RESULT ← MATRIX ⎕ZZ[INDEX] GA∆*

Description: The result is an index vector which can be used to index the rows of an alpha matrix.

EG. *A[A ⎕ZZ GA∆;]*

will return *A* in ascending order.

The index is optional. If included it specifies the column of the matrix at which to begin the grade. All columns to the left of the specified one are ignored.

DT∆

Function: Converts a Day - Month - Year character date to a number, or converts a number to a character date.

Syntax: Date to number:

RESULT ← CHARACTER-DATE ⎕ZZ DT∆

Number to date:

RESULT ← NUMBER ⎕ZZ[LENGTH] DT∆

Description: If the left argument is character, conversion from date to numeric is attempted. The left argument can be of two forms:

6 long: DDMMYY
8 long: DD/MM/YY (slashes required)

In both cases leading blanks are invalid (ie. January 5, 1978 must be 05/01/78 or 050178, not 5/ 1/78 or 5178). The result is a numeric scalar giving the number of days elapsed since January 1, 1950. The system function *CK∆* can be used to insure a date is valid before attempting to convert it to numeric.

If the left argument is numeric, conversion from numeric to date is attempted. The index *LENGTH* is required and must be either 6 or 8, specifying the length and format of the result. The reslut is a character vector 6 or 8 long giving the date represented by the left argument, as DDMMYY or DD/MM/YY.

Note: This function will handle dates from January 1, 1950 to December 31, 2049.

GS∆

Function: To pack and unpack objects for efficient storage on disk and to group related variables under a common name.

Syntax: Return namelist of structure:

RESULT ← STRUCTURE-NAME □ZZ GS∆

Gather objects into structure:

RESULT ← NAME-LIST □ZZ[1] GS∆

Split objects out of structure:

RESULT ← NAME-LIST □ZZ[2] GS∆

Description: Presently, variables are stored on the disk in the format of 1 per sector. Since a sector can hold up to 256 bytes, this could result in a waste of disk space if the variables you write are small.

The GATHER/SPLIT routines allow you to store a number of variables of different characteristics and dimensions under one name. This allows you to pack more than one variable on a sector. Compression is also done on the data to further reduce space requirements.

Terms: These terms apply to the write up:

Structure:
Variables grouped together under one name.

Structure Name:
Name given to the structure.

Namelist:
List of names of variables making up the structure (the first name in the list is the structure name.

Status Vector:
A vector of numbers that is returned when a *GATHER* or a *SPLIT* is done to report on the variables involved.

RETURN Namelist:

RESULT ← *STRUCTURE-NAME* ⎕ZZ *GS*Δ

When you supply the name of the structure, the function returns the namelist associated with it in *RESULT*. The namelist is an N by 4 character array. The first entry in the namelist is always the structure name followed by one entry for each variable in the structure. This should be used to obtain the name list needed for a *GATHER* or *SPLIT* operation.

GATHER variables into a structure:
RESULT ← *NAME-LIST* ⎕ZZ[1] *GS*Δ

The index of [1] specifies this is to be a gather operation. You supply a N by 4 character matrix namelist where the first entry is the structure name and the others are the variables that are to be packed in the structure. *RESULT* is the status vector which holds the status information on the *GATHER* operation. *RESULT*[1] gives the total length of the structure. *RESULT*[*N*] gives the length in the structure of the corresponding element (*NAME-LIST*[*N*]) in the *NAME LIST*. The length will be negative if the variable has been compressed by the *GATHER* operation (variables which are larger in their compressed form than in their uncompressed form are not compressed). If a name in the namelist has no value (0=⎕*NC NAME-LIST*[*N*;]) then a status of 0 is returned in *RESULT*[*N*] and the name is not included in the structure.

If the structure exists already and you do a *GATHER*, it acts as an update. If you specify a variable in the name list that is already in the structure, it will replace it with the new value. However, if the variable has no value (0=⎕*NC NAME-LIST*[*N*;]) it will be deleted from the structure. If the variable in the namelist is not in the structure already, and does have a value, it will be added to the structure.

SPLIT up variables in a structure:

RESULT ← *NAME-LIST* ⎕ZZ[2] *GS*Δ

The index of [2] specifies this is to be a *SPLIT* operation. You supply a N by 4 character matrix namelist where the first entry is the structure name and the others are the variables you want split out of the structure. You can ask for any of the variables in the structure and in any order. *RESULT* is the status vector that returns status information on the *SPLIT* operation. *RESULT*[1] gives the total length of the variables that are split out of the structure. *RESULT*[*N*] is the status for the corresponding variable in the namelist (*NAME-LIST*[*N*;]). *RESULT*[*N*] can have a value of 0, 1 or 2. *A* 0 means the variable in the namelist is not found in the structure. *A* 1 means that it was found and it is character. *A* 2 means that it was found and it is numeric.

If the variable already exists at the time of the *SPLIT* operation, it will receive a new value. If it does not exist, it will be created.

*DN*Δ

Function: Convert Date to Number

Syntax: *R*←*A* ⎕ZZ *DN*Δ

Description: System function *DN*Δ accepts one or more dates and converts them to the number of days elapsed since January first, Zero A.D. The program extends the current calendar backwards to Zero A.D., which is not strictly speaking correct, since the current scheme was not adopted until the 1700's, however for dates from 1750 through to 2200+ it will yield correct results.

The left argument *A* can be either a three element vector, or an *N*×3 matrix. In either case, each date to be converted must consist of three numbers in the order MONTH DAY YEAR.

NOTE: This function does NOT check for valid months and days. If an invalid month or day is passed to it, the result will be meaningless.

Examples:

```
      8 28 1978 □ZZ DNΔ
722689

      (3 3ρ8 28 1978 12 25 2001 1 1 1900) □ZZ DNΔ
722689 731209 693961
```

Restrictions:

ERROR MESSAGE	POSSIBLE CAUSE
DOMAIN ERROR	Left argument is not numeric.
	The year in one or more dates may be too large or too small. This function is restricted to years from 128 to 32000.
LENGTH ERROR	The left argument does not have exactly three elements per date.
RANK ERROR	The left argument is not a vector or a matrix.

NDΔ

Function: Convert Number to Date.

Syntax: *R←A* □*ZZ NDΔ*

Description: System function *NDΔ* accepts a numeric vector representing the number of days elapsed since January first Zero A.D., and converts each number to a date in the form of MONTH DAY YEAR. The program extends the current calendar backwards to Zero A.D. (see System Function *DNΔ*).

The left argument *A* can be either a scalar or a vector. The result is an *N*×3 matrix, where *N* is the same as the number of elements in the *A* argument.

Examples:

```
      722689 □ZZ DNΔ
8 28 1978

      722689 731209 693961 □ZZ DNΔ
 8 28 1978
12 15 2001
 1  1 1900
```

Restrictions:

ERROR MESSAGE	POSSIBLE CAUSE
DOMAIN ERROR	Left argument is not numeric. The left argument may be outside the domain of years handled by this function. This function is restricted from 128 to 32000.
RANK ERROR	The left argument is not a vector or scalar.