

# **MSX**

## **Top Secret**

# **3**

**THE FINAL COMPILATION OF MSX INFORMATION**

## **Appendix**

**Edison Moraes [2019-2022]**



## **AUTHOR'S NOTE**

After the release of MSX Top Secret 2, in April 2004, I figured that there would be no need to update it anymore, since MSX is no longer commercially manufactured by large companies. In addition, the internet has evolved and a wide range of information has become available to everyone.

However, the information is sparse, leading to the need for multiple and tiring searches not always achieving complete success. That's why I thought it convenient to write this third – and final – edition of MSX Top Secret, gathering all the information I could find in one place.

As the amount of information is very large, I divided it into two volumes and, interestingly, the Appendix ended up being ready before the main volume, which is still in progress. Because of the time involved, I thought it best to publish the Appendix, which is the tome presented here.

Good research!

Edison Antonio Pires de Moraes (author)

**Sorry for my english mistakes. I'm not fluent in english.**

**Suggestions and information about errors are welcome.**

**Send it to:**

**eapmoraes2012@gmail.com**



# Índice

|  |           |
|--|-----------|
| <b>1 – CHARACTERS AND KEYBOARD.....</b>                    | <b>15</b> |
| 1.1 – CHARACTER SETS.....                                  | 15        |
| 1.1.1 – Japanese Set.....                                  | 15        |
| 1.1.2 – Internacional Set.....                             | 16        |
| 1.1.3 – Brazilian Set 1.0 (Expert 1.0).....                | 17        |
| 1.1.4 – Brazilian Set 1.1 (Expert 1.1 and Hotbit 1.2)..... | 18        |
| 1.1.5 – Russian Set.....                                   | 19        |
| 1.1.6 – Korean Set.....                                    | 20        |
| 1.1.7 – Arabic Set (AX-170).....                           | 21        |
| 1.1.8 – Arabic Set (AX-500).....                           | 22        |
| 1.2 – KEYBOARD MATRICES.....                               | 23        |
| 1.2.1 – Japanese Matrix.....                               | 23        |
| 1.2.1.1 – Japanese Matrix with locked か な /KANA key.....   | 24        |
| 1.2.2 – PX-7 Matrix.....                                   | 25        |
| 1.2.3 – Internacional Matrix.....                          | 26        |
| 1.2.5 – Argentine / Spanish Matrix.....                    | 27        |
| 1.2.6 – United Kingdom Matrix (England).....               | 27        |
| 1.2.7 – Russian Matrix.....                                | 28        |
| 1.2.7.1 – Russian Matrix with locked PYC/CODE key.....     | 28        |
| 1.2.8 – Korean Matrix.....                                 | 29        |
| 1.2.8.1 – Korean Matrix with locked 한글 /CODE key.....      | 29        |
| 1.2.9 – Arabic Matrix.....                                 | 30        |
| 1.2.9.1 – Arabic matrix with Arabic mode activated.....    | 30        |
| 1.3 – KEYBOARD LAYOUTS.....                                | 31        |
| 1.3.1 – Internacional Layout.....                          | 31        |
| 1.3.2 – Japanese Layout (JIS).....                         | 31        |
| 1.3.3 – Japanese Layout (ANSI).....                        | 31        |
| 1.3.4 – Brazilian Layout 1.0 (Expert 1.0).....             | 32        |
| 1.3.5 – Brazilian Layout 1.1 (Hotbit / Expert 1.1).....    | 32        |
| 1.3.6 – United Kingdom Layout.....                         | 32        |
| 1.3.7 – Argentine / Spanish Layout.....                    | 33        |
| 1.3.8 – Russian Layout (Cyrillic).....                     | 33        |
| 1.3.9 – Korean Layout (CPC-400).....                       | 33        |
| 1.3.10 – Arabic Layout (AX-170).....                       | 34        |
| 1.3.11 – French Layout (ML-F80).....                       | 34        |

|   |           |
|---|-----------|
| 1.3.12 – German Layout (HB-F700D).....  | 34        |
| 1.4 – CONTROL CODES.....                | 35        |
| <b>2 – I/O PORTS MAP.....</b>           | <b>36</b> |
| <b>3 – MSX-BASIC.....</b>               | <b>41</b> |
| 3.1 – FORMAT.....                       | 41        |
| 3.1.1 – Instructions Abbreviations..... | 41        |
| 3.1.2 – Logical Operation Codes.....    | 41        |
| 3.1.3 – Code notations.....             | 42        |
| 3.1.4 – Format Notations.....           | 42        |
| 3.2 – INSTRUCTIONS DESCRIPTION.....     | 43        |
| 3.3 – EXTENDED COMMANDS.....            | 73        |
| 3.3.1 – Commands Description.....       | 79        |
| A.....                                  | 79        |
| B.....                                  | 82        |
| C.....                                  | 85        |
| D.....                                  | 96        |
| E.....                                  | 99        |
| F.....                                  | 101       |
| G.....                                  | 104       |
| H.....                                  | 105       |
| I.....                                  | 106       |
| J.....                                  | 108       |
| K.....                                  | 109       |
| L.....                                  | 113       |
| M.....                                  | 118       |
| N.....                                  | 125       |
| O.....                                  | 145       |
| P.....                                  | 147       |
| Q.....                                  | 155       |
| R.....                                  | 156       |
| S.....                                  | 162       |
| T.....                                  | 171       |
| U.....                                  | 174       |
| V.....                                  | 175       |
| W.....                                  | 177       |
| X.....                                  | 177       |
| Y.....                                  | 178       |
| 3.4 – MSX-BASIC ERROR CODES.....        | 178       |

|  |            |
|--|------------|
| <b>4 – MSXDOS.....</b>                                 | <b>181</b> |
| 4.1 – FORMAT NOTATION.....                             | 181        |
| 4.1.1 – Description of filenames extensions.....       | 182        |
| 4.2 – DESCRIPTION OF COMMANDS.....                     | 190        |
| 4.3 – BDOS CALLS.....                                  | 204        |
| 4.3.1 – I/O Handling.....                              | 204        |
| 4.3.2 – Definition and reading of parameters.....      | 206        |
| 4.3.3 – Absolute reading/writing of sectors.....       | 208        |
| 4.3.4 – Accessing files by using FCB.....              | 209        |
| 4.3.5 – Functions added by MSXDOS2.....                | 212        |
| 4.3.6 – Functions added by NEXTOR.....                 | 226        |
| 4.4 – MSXDOS ERROR CODES.....                          | 232        |
| 4.5 – MSXDOS2 ERROR CODES.....                         | 233        |
| 4.5.1 – Disk Errors.....                               | 233        |
| 4.5.2 – MSXDOS Functions Errors.....                   | 234        |
| 4.5.3 – Errors Added by Nextor.....                    | 235        |
| 4.5.4 – End Programs Errors.....                       | 235        |
| 4.5.5 – Command Errors.....                            | 235        |
| <b>5 – SYMBOLS.....</b>                                | <b>236</b> |
| 5.1 – KERNEL ROUTINES.....                             | 236        |
| 5.1.1 – Kernel Restarts.....                           | 236        |
| 5.1.2 – Kernel Commands (Multitasking Management)..... | 238        |
| 5.1.3 – Kernel Responses (Multitasking Mangement)..... | 241        |
| 5.1.4 – Kernel Functions (Memory Management).....      | 242        |
| 5.1.5 – Kernel Functions (Banking Management).....     | 244        |
| 5.1.6 – Kernel Functions (Miscellaneous).....          | 247        |
| 5.2 – DESKTOP MANAGER COMMANDS.....                    | 247        |
| 5.2.1 – Desktop Manager Responses.....                 | 253        |
| 5.2.2 – Desktop Manager Services.....                  | 256        |
| 5.2.3 – Desktop Manager Functions.....                 | 259        |
| 5.2.4 – Desktop Manager Data Records.....              | 260        |
| 5.2.4.1 – Window Data Record.....                      | 260        |
| 5.2.4.2 – Control Group Data Record.....               | 262        |
| 5.2.4.3 – Control Data Records.....                    | 262        |
| 5.2.4.4 – Calculation Rule Data Record.....            | 263        |
| 5.3 – CONTROL TYPES.....                               | 263        |
| 5.3.1 – Paint.....                                     | 263        |
| 5.3.2 – Graphics.....                                  | 266        |

|  |     |
|--|-----|
| 5.3.3 – Buttons.....                               | 268 |
| 5.3.4 – Miscellaneous.....                         | 269 |
| 5.3.5 – Textinput.....                             | 270 |
| 5.3.6 – Lists.....                                 | 273 |
| 5.3.7 – Pulldown Menus.....                        | 275 |
| 5.4 – FONTS AND GRAPHICS.....                      | 276 |
| 5.4.1 – Standard graphics.....                     | 276 |
| 5.4.2 – Graphics with extended header.....         | 277 |
| 5.4.3 – Fonts.....                                 | 278 |
| 5.5 – SYSTEM MANAGER.....                          | 279 |
| 5.5.1 – Application Management.....                | 279 |
| 5.5.2 – System Management.....                     | 282 |
| 5.5.3 – Dilogue Services.....                      | 283 |
| 5.5.4 – System Manager Functions.....              | 286 |
| 5.6 – FILE MANAGER.....                            | 289 |
| 5.6.1 – System Manager Messages.....               | 289 |
| 5.6.2 – Error Codes.....                           | 290 |
| 5.6.3 – Mass Storage Device Functions.....         | 291 |
| 5.6.4 – File Management Functions.....             | 293 |
| 5.6.5 – Directory Management Functions.....        | 297 |
| 5.6.6 – Device Manager Functions.....              | 303 |
| 5.7 – SYMSHELL TEXT TERMINAL.....                  | 306 |
| 5.8.1 – SymShell Commands and responses.....       | 306 |
| 5.7.2 – Symshell Text Terminal Control.....        | 310 |
| 5.7.3 – Extended ASCII Codes.....                  | 311 |
| 5.7.4 – Keyboard Scan Codes.....                   | 311 |
| 5.8 – SYSTEM CONFIGURATION.....                    | 312 |
| 5.8.1 – Header.....                                | 312 |
| 5.8.2 – Core Area Part.....                        | 313 |
| 5.8.2.1 – Mass storage devices.....                | 313 |
| 5.8.2.2 – Display and miscellaneous (1).....       | 313 |
| 5.8.2.3 – Keyboard (1) and mouse.....              | 314 |
| 5.8.2.4 – Miscellaneous (2) and Desktop Links..... | 314 |
| 5.8.3 – Data Area Part.....                        | 315 |
| 5.8.3.1 – Desktop Links (2).....                   | 315 |
| 5.8.3.2 – Screen Saver.....                        | 316 |
| 5.8.3.3 – Keyboard (2).....                        | 316 |
| 5.8.3.4 – Security.....                            | 316 |



|  |            |
|--|------------|
| 5.9 – SCREENSAVER APPLICATIONS.....                            | 316        |
| 5.10 – SYMBOS MEMORY MAP.....                                  | 317        |
| 5.10.1 – General Memory Usage.....                             | 317        |
| 5.10.2 – Application Memory Usage.....                         | 318        |
| 5.10.3 – Memory Configurations.....                            | 318        |
| 5.11 – SCREEN MANAGER.....                                     | 320        |
| 5.12 – NETWORK DAEMON.....                                     | 320        |
| 5.12.1 – Configuration.....                                    | 320        |
| 5.12.2 – Transportation Layer Services.....                    | 320        |
| 5.12.3 – Application Layer Services.....                       | 322        |
| 5.13 – SYMBOS CONSTANTS.....                                   | 322        |
| 5.13.1 – Process-IDs.....                                      | 322        |
| 5.13.2 – Messages.....   | 322        |
| 5.13.3 – Kernel Commands.....                                  | 322        |
| 5.13.4 – Kernel Responses.....                                 | 323        |
| 5.13.5 – System Commands.....                                  | 323        |
| 5.13.6 – System Responses.....                                 | 324        |
| 5.13.7 – Desktop Commands.....                                 | 325        |
| 5.13.8 – Desktop Responses.....                                | 326        |
| 5.13.9 – Shell Commands.....                                   | 327        |
| 5.13.10 – Shell Responses.....                                 | 327        |
| 5.13.11 – Screensaver Messages.....                            | 327        |
| 5.13.12 – Desktop Actions.....                                 | 328        |
| 5.13.13 – Desktop Services.....                                | 328        |
| 5.13.14 – Jumps.....   | 329        |
| 5.13.15 – Filemanager Functions (call via MSC_SYS_SYSFIL)..... | 330        |
| <b>6 – UZIX.....</b>   | <b>332</b> |
| 6.1 – COMMANDS.....  | 332        |
| 6.1.1 – Conventions.....                                       | 332        |
| 6.1.1.1 – Format Notations.....                                | 332        |
| 6.1.2 – Commands Description.....                              | 333        |
| 6.2 – HIERARCHICAL STRUCTURE.....                              | 348        |
| 6.3 – MEMORY MAPPING.....                                      | 349        |
| 6.4 – SYSTEM CALLS.....  | 350        |
| 6.4.1 – Direct System Calls.....                               | 350        |
| 6.4.2 – Indirect System Call.....                              | 365        |
| 6.4.3 – Calls via GETSET.....                                  | 365        |
| 6.4.4 – TCP/IP module.....                                     | 368        |

|  |            |
|--|------------|
| 6.4.5 – Error codes.....                               | 371        |
| 6.5 – VT-5 TERMINAL CODES.....                         | 372        |
| <b>7 – SYSTEM VARIABLES.....</b>                       | <b>374</b> |
| 7.1 – SYSTEM AREA FOR MSXDOS1.....                     | 374        |
| 7.1.1 – Hooks called by disk routines.....             | 376        |
| 7.1.2 – Other DOS data.....                            | 377        |
| 7.1.3 – Hooks for the 'COM:' port.....                 | 379        |
| 7.1.4 – Keyboard.....                                  | 379        |
| 7.1.5 – MSXDOS Variables.....                          | 379        |
| 7.1.6 – DPB addresses.....                             | 380        |
| 7.1.7 – Routines used by MSXDOS.....                   | 381        |
| 7.1.8 – Inter-slot movement routines.....              | 381        |
| 7.2 – SYSTEM AREA FOR MSXDOS2.....                     | 382        |
| 7.2.1 – Physical information about disks.....          | 382        |
| 7.2.2 – Hooks called by disk routines (1).....         | 382        |
| 7.2.3 – Logical information about disks.....           | 384        |
| 7.2.4 – Hooks called by disk routines.....             | 384        |
| 7.2.5 – MSXDOS2 variables.....                         | 385        |
| 7.2.6 – Pointers and buffers (FAT, DTA, FCB, DPB)..... | 388        |
| 7.2.7 – System jumps.....                              | 389        |
| 7.3 – INTER-SLOT SUBROUTINES.....                      | 389        |
| 7.4 – USR FUNCTION AND TEXT MODES.....                 | 390        |
| 7.5 – AREA USED BY THE SCREEN.....                     | 391        |
| 7.5.1 – Screen 0.....                                  | 391        |
| 7.5.2 – Screen 1.....                                  | 392        |
| 7.5.3 – Screen 2.....                                  | 392        |
| 7.5.4 – Screen 3.....                                  | 393        |
| 7.5.4 – Other Screen Values.....                       | 393        |
| 7.6 – VDP REGISTERS AREA.....                          | 394        |
| 7.6.1 – Area used for the V9938.....                   | 395        |
| 7.6.2 – Area used for the V9958.....                   | 396        |
| 7.7 – MISCELLANEOUS.....                               | 396        |
| 7.8 – AREA USED BY PLAY COMMAND.....                   | 397        |
| 7.24 – AREA USED BY THE PLAY COMMAND.....              | 398        |
| 7.8.1 – Offset for PLAY buffer parameter control.....  | 399        |
| 7.8.2 – Data area for the parameter buffer.....        | 400        |
| 7.9 – KEYBOARD AREA.....                               | 400        |
| 7.10 – AREA USED BY CASSETTE.....                      | 401        |

|  |            |
|--|------------|
| 7.11 – AREA USED BY CIRCLE COMMAND.....                          | 402        |
| 7.12 – AREA INTERNALLY USED BY BASIC.....                        | 403        |
| 7.12.1 – BASIC text buffers.....                                 | 404        |
| 7.12.2 – General data.....                                       | 405        |
| 7.12.3 – BASIC lines control at runtime.....                     | 407        |
| 7.12.4 – BASIC text storage adresses.....                        | 408        |
| 7.12.5 – Area for user functions.....                            | 409        |
| 7.12.6 – Interpreter data area.....                              | 410        |
| 7.13 – MATH-PACK AREA.....                                       | 411        |
| 7.14 – DISK SYSTEM DATA AREA.....                                | 412        |
| 7.15 – AREA USED BY PAINT COMMAND.....                           | 414        |
| 7.16 – ADDED AREA FOR MSX2.....                                  | 415        |
| 7.17 – AREA USED BY RS232C.....                                  | 417        |
| 7.18 – GENERAL DATA AREA.....                                    | 419        |
| 7.19 – BIOS EXPANSION ROUTINES.....                              | 424        |
| 7.20 – DATA AREA FOR SLOTS AND PAGES.....                        | 424        |
| 7.20.1 – Main-ROM slot.....                                      | 426        |
| 7.20.2 – Secondary slot register.....                            | 427        |
| 7.21 – HOOKS DESCRIPTION.....                                    | 427        |
| <b>8 – BIOS ROUTINES.....</b>                                    | <b>440</b> |
| 8.1 – Main-ROM ROUTINES.....                                     | 440        |
| 8.1.1 – RST Routines.....  | 440        |
| 8.1.2 – Routines for I/O initialization.....                     | 443        |
| 8.1.3 – Routines for accessing the VDP.....                      | 443        |
| 8.1.4 – Routines for access to PSG.....                          | 449        |
| 8.1.5 – Routines for accessing keyboard, screen and printer..... | 450        |
| 8.1.6 – I/O access routines for games.....                       | 453        |
| 8.1.7 – I/O access routines for cassette register.....           | 455        |
| 8.1.8 – Routines for the PSG queue.....                          | 456        |
| 8.1.9 – Routines for MSX1 graphics screens.....                  | 457        |
| 8.1.10 – Miscellaneous.....                                      | 460        |
| 8.1.11 – Routines for accessing the disk system.....             | 462        |
| 8.1.12 – Routines added for MSX2.....                            | 463        |
| 8.1.13 – Routines added for MSX2+.....                           | 465        |
| 8.1.14 – Routines added for the MSX turbo R.....                 | 466        |
| 8.1.15 – Inter-slot work area routines.....                      | 467        |
| 8.2 – SubROM ROUTINES.....                                       | 468        |
| 8.2.1 – Routines for BASIC graphical functions.....              | 468        |

|   |     |
|---|-----|
| 8.2.2 – Routines for graphical functions.....             | 471 |
| 8.2.3 – Duplicate routines (same as MainROM).....         | 475 |
| 8.2.4 – Various routines for MSX2 or higher.....          | 477 |
| 8.2.5 – Color palette handling routines.....              | 482 |
| 8.2.6 – Various routines used by BASIC.....               | 482 |
| 8.2.7 – Block transfer routines (bit-blit).....           | 484 |
| 8.3 – MATH-PACK ROUTINES.....                             | 487 |
| 8.3.1 – Floating point mathematical functions.....        | 487 |
| 8.3.2 – Operations with integer numbers.....              | 487 |
| 8.3.3 – Special functions.....                            | 488 |
| 8.3.4 – Movement.....                                     | 488 |
| 8.3.5 – Conversions.....                                  | 489 |
| 8.4 – BASIC INTERPRETER ROUTINES.....                     | 491 |
| 8.4.1 – Execution routines.....                           | 491 |
| 8.4.2 – Command and function routines.....                | 494 |
| 8.5 – EXTENDED BIOS ROUTINES.....                         | 498 |
| 8.5.1 – Extended BIOS Entry.....                          | 498 |
| 8.5.2 – Internal commands (broadcast commands).....       | 499 |
| 8.5.3 –Memory Mapper.....                                 | 500 |
| 8.5.3.1 – Memory Mapper Manipulation Routines.....        | 502 |
| 8.5.4 – RS232C Serial Port and MSX Modem.....             | 507 |
| 8.5.4.1 – Parameter Bytes.....                            | 508 |
| 8.5.4.2 – RS232C serial port manipulation routines.....   | 509 |
| 8.5.4.3 – MSX Modem manipulation routines.....            | 512 |
| 8.5.5 – MSX-AUDIO.....                                    | 518 |
| 8.5.5.1 – Startup routines.....                           | 519 |
| 8.5.5.2 – PCM/ADPCM Routines.....                         | 521 |
| 8.5.5.3 – Musical keyboard routines.....                  | 524 |
| 8.5.5.4 – FM synthesizer routines.....                    | 525 |
| 8.5.5.5 – MBIOS routines (Music BIOS).....                | 527 |
| 8.5.6 – MSX-JE.....                                       | 554 |
| 8.5.6.1 – Calling MSX-JE functions.....                   | 555 |
| 8.5.6.2 – MSX-JE dictionary interface.....                | 557 |
| 8.5.7 – MSX UNAPI.....                                    | 562 |
| 8.5.7.1 – RAM Helper.....                                 | 562 |
| 8.5.7.2 – API for Ethernet cartridges.....                | 564 |
| 8.5.8 – MemMan.....                                       | 568 |
| 8.5.8.1 – Fast Calls (Preferred alternative entries)..... | 568 |

|  |            |
|--|------------|
| 8.5.8.2 – MemMan Functions.....                              | 570        |
| 8.5.9 – System commands.....                                 | 575        |
| 8.6 – DISC INTERFACE ROUTINES.....                           | 576        |
| 8.6.1 – Interface Initialization.....                        | 576        |
| 8.6.2 – Standard interface routines.....                     | 577        |
| 8.6.3 – Routines for accessing standard IDE Hard-Disks.....  | 582        |
| 8.6.4 – Routines for accessing standard SCSI Hard-Disks..... | 584        |
| 8.7 – MSX-MUSIC ROUTINES (FM/OPLL).....                      | 595        |
| <b>9 – MSX-HID (Human Interface Device).....</b>             | <b>599</b> |
| 9.1 – FINGERPRINTS OF MSX DEVICES.....                       | 599        |
| 9.2 – FINGERPRINTS OF SEGA COMPATIBLE DEVICES.....           | 599        |
| 9.3 – FINGERPRINTS OF DEVICES THAT CONFLICT.....             | 599        |
| 9.4 – HOMEBREW DEVICES.....                                  | 600        |
| 9.5 – RESERVED FINGERPRINTS (DO NOT USE).....                | 600        |
| <b>10 – Z80/R800 MNEMONICS.....</b>                          | <b>601</b> |
| 10.1 – 8-BIT LOAD GROUP.....                                 | 601        |
| 10.2 – 16-BIT LOAD GROUP.....                                | 603        |
| 10.3 – 8-BIT ARITHMETIC GROUP.....                           | 605        |
| 10.4 – 16-BIT ARITHMETIC GROUP.....                          | 608        |
| 10.5 – EXCHANGE GROUP.....                                   | 609        |
| 12.6 – BLOCK TRANSFER GROUP.....                             | 610        |
| 10.7 – SEARCH GROUP.....                                     | 611        |
| 10.8 – COMPARISON GROUP.....                                 | 612        |
| 10.9 – LOGICAL GROUP.....                                    | 613        |
| 10.10 – ROTATE AND SHIFT GROUP.....                          | 615        |
| 10.11 – BIT SET, RESET AND TEST GROUP.....                   | 618        |
| 10.12 – JUMP GROUP.....                                      | 620        |
| 10.13 – CALL AND RETURN GROUP.....                           | 621        |
| 10.14 – INPUT AND OUTPUT GROUP.....                          | 622        |
| 10.15 – GENERAL PURPOSE AND CONTROL GROUPS.....              | 624        |
| <b>11 – STANDARD CHIPS REGISTERS MAPS.....</b>               | <b>625</b> |
| 11.1 – MAP OF THE REGISTERS OF THE V9918/38/58.....          | 625        |
| 11.1.1 – Access ports for VDPs V9918/38/38.....              | 630        |
| 11.1.2 – Standard color chart.....                           | 631        |
| 11.2 – MAP OF THE V9990 REGISTERS.....                       | 632        |
| 11.2.1 – Access ports to V9990.....                          | 637        |
| 11.3 – MAP OF PSG REGISTERS (AY-3-8910).....                 | 639        |
| 11.3.1 – Access ports to PSG.....                            | 640        |

|  |            |
|--|------------|
| 11.4 – MAP OF FM-OPLL REGISTERS (YM2413).....    | 641        |
| 11.4.1 – Access ports to OPLL.....               | 642        |
| 11.5 – MSX-AUDIO REGISTERS MAP (Y8950).....      | 643        |
| 11.5.1 – MSX-Audio access ports.....             | 646        |
| MSX-Audio a.....                                 | 646        |
| 11.6 – MAP OF THE OPL4 REGISTERS (YMF278).....   | 647        |
| 11.6.1 – Register Array #0.....                  | 647        |
| 11.6.2 – Register Array #1.....                  | 649        |
| 11.6.3 – Wave synthesis.....                     | 652        |
| 11.6.4 – OLP4 access ports.....                  | 654        |
| 11.6.5 – Wave table synthesis header.....        | 655        |
| 11.6.6 – Wave data lenght.....                   | 656        |
| 11.7 – MAP OF THE SCC REGISTERS (2212/2312)..... | 657        |
| 11.7.1 – Acess adresses for SCC.....             | 658        |
| <b>BIBLIOGRAPHIC REFERENCES.....</b>             | <b>660</b> |
| <b>OTHER BOOKS BY THE AUTHOR.....</b>            | <b>663</b> |









### 1.1.4 – Brazilian Set 1.1 (Expert 1.1 and Hotbit 1.2)

|   | 0         | 1         | 2         | 3         | 4         | 5         | 6         | 7         | 8   | 9         | A         | B    | C   | D   | E         | F         |
|---|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----|-----------|-----------|------|-----|-----|-----------|-----------|
| 0 | MULL      | GRPH      | CTRL<br>B | CTRL<br>C | CTRL<br>D | CTRL<br>E | CTRL<br>F | BEEP      | BS  | TAB       | LF        | HOME | CLS | RET | CTRL<br>N | CTRL<br>O |
| 1 | CTRL<br>P | CTRL<br>R | INS       | CTRL<br>S | CTRL<br>T | CTRL<br>U | CTRL<br>V | CTRL<br>W | SEL | CTRL<br>V | CTRL<br>Z | ESC  | →   | ←   | ↑         | ↓         |
| 2 |           | !         | "         | #         | \$        | %         | &         | '         | (   | )         | *         | +    | ,   | -   | .         | /         |
| 3 | 0         | 1         | 2         | 3         | 4         | 5         | 6         | 7         | 8   | 9         | :         | ;    | <   | =   | >         | ?         |
| 4 | @         | A         | B         | C         | D         | E         | F         | G         | H   | I         | J         | K    | L   | M   | N         | O         |
| 5 | P         | Q         | R         | S         | T         | U         | V         | W         | X   | Y         | Z         | [    | \   | ]   | ^         | _         |
| 6 | `         | a         | b         | c         | d         | e         | f         | g         | h   | i         | j         | k    | l   | m   | n         | o         |
| 7 | p         | q         | r         | s         | t         | u         | v         | w         | x   | y         | z         | {    |     | }   | ~         | DEL       |
| 8 | Ç         | ü         | é         | â         | À         | à         | ˆ         | ç         | ê   | í         | ó         | Ô    | Ä   | É   | Ö         | À         |
| 9 | É         | æ         | Æ         | ô         | ö         | ò         | ô         | ü         | ö   | Ü         | ¢         | £    | ¥   | Q   | f         |           |
| A | Á         | í         | ó         | ú         | ñ         | Ñ         | º         | ¿         | ¬   | ¬         | ½         | ¼    | i   | <<  | >>        |           |
| B | Ä         | ä         | ÿ         | ï         | ø         | ö         | ü         | ÿ         | ij  | ¾         | ˆ         | ◊    | ¼   | π   | δ         |           |
| C | ■         | ■         | ■         | ■         | ■         | ■         | ■         | ■         | ■   | ■         | ■         | ■    | ■   | ■   | ■         | ■         |
| D | ◀         | ▶         | ◀         | ▶         | ◀         | ▶         | ◀         | ▶         | ◀   | ▶         | ◀         | ▶    | ◀   | ▶   | ◀         | ▶         |
| E | α         | β         | Γ         | Π         | Σ         | σ         | μ         | γ         | Φ   | Θ         | Ω         | δ    | ω   | ϑ   | €         | Π         |
| F | ≡         | ±         | ≥         | ≤         | ↑         | J         | ÷         | ≈         | °   | •         | -         | √    | ˆ   | ²   | ■         | ■         |

|   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 |   | ☺ | ☹ | ♥ | ♠ | ♣ | ♠ | • | ■ | ○ | ☹ | ♂ | ♀ | ♂ | ♂ | * |
| 5 | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + |

**Obs.:** The character in the 9EH position (Cz) was “Pt” in the first HOTBIT version.



## 1.1.6 – Korean Set

|   | 0         | 1         | 2         | 3         | 4         | 5         | 6         | 7         | 8   | 9         | A         | B    | C   | D   | E         | F         |
|---|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----|-----------|-----------|------|-----|-----|-----------|-----------|
| 0 | NUL       | GRPH      | CTRL<br>B | CTRL<br>C | CTRL<br>D | CTRL<br>E | CTRL<br>F | BEEP      | BS  | TAB       | LF        | HOME | CLS | RET | CTRL<br>N | CTRL<br>O |
| 1 | CTRL<br>P | CTRL<br>R | INS       | CTRL<br>S | CTRL<br>T | CTRL<br>U | CTRL<br>V | CTRL<br>W | SEL | CTRL<br>Y | CTRL<br>Z | ESC  | →   | ←   | ↑         | ↓         |
| 2 |           | !         | "         | #         | \$        | %         | &         | '         | (   | )         | *         | +    | ,   | -   | .         | /         |
| 3 | 0         | 1         | 2         | 3         | 4         | 5         | 6         | 7         | 8   | 9         | :         | ;    | <   | =   | >         | ?         |
| 4 | @         | A         | B         | C         | D         | E         | F         | G         | H   | I         | J         | K    | L   | M   | N         | O         |
| 5 | P         | Q         | R         | S         | T         | U         | V         | W         | X   | Y         | Z         | [    | \   | ]   | ^         | _         |
| 6 | `         | a         | b         | c         | d         | e         | f         | g         | h   | i         | j         | k    | l   | m   | n         | o         |
| 7 | p         | q         | r         | s         | t         | u         | v         | w         | x   | y         | z         | {    |     | }   | ~         | DEL       |
| 8 | ☸         | ♥         | ♣         | ♠         | ○         | ●         | ㄱ         | ㅋ         | ㄴ   | ㄷ         | ㄹ         | ㅁ    | ㅂ   | ㅅ   | ㅈ         | ㅊ         |
| 9 | ㅍ         | ㅇ         | ㅊ         | ㅌ         | ㅍ         | ㅊ         | ㅋ         | ㅌ         | ㅍ   | ㅊ         | ㅌ         | ㅍ    | ㅊ   | ㅌ   | ㅍ         | ㅊ         |
| A | ㅈ         | ㅊ         | ㅌ         | ㅍ         | ㅊ         | ㅌ         | ㅍ         | ㅊ         | ㅌ   | ㅍ         | ㅊ         | ㅌ    | ㅍ   | ㅊ   | ㅌ         | ㅍ         |
| B | ㅈ         | ㅊ         | ㅌ         | ㅍ         | ㅊ         | ㅌ         | ㅍ         | ㅊ         | ㅌ   | ㅍ         | ㅊ         | ㅌ    | ㅍ   | ㅊ   | ㅌ         | ㅍ         |
| C | ㅈ         | ㅊ         | ㅌ         | ㅍ         | ㅊ         | ㅌ         | ㅍ         | ㅊ         | ㅌ   | ㅍ         | ㅊ         | ㅌ    | ㅍ   | ㅊ   | ㅌ         | ㅍ         |
| D | ㅈ         | ㅊ         | ㅌ         | ㅍ         | ㅊ         | ㅌ         | ㅍ         | ㅊ         | ㅌ   | ㅍ         | ㅊ         | ㅌ    | ㅍ   | ㅊ   | ㅌ         | ㅍ         |
| E | ㅈ         | ㅊ         | ㅌ         | ㅍ         | ㅊ         | ㅌ         | ㅍ         | ㅊ         | ㅌ   | ㅍ         | ㅊ         | ㅌ    | ㅍ   | ㅊ   | ㅌ         | ㅍ         |
| F | ㅈ         | ㅊ         | ㅌ         | ㅍ         | ㅊ         | ㅌ         | ㅍ         | ㅊ         | ㅌ   | ㅍ         | ㅊ         | ㅌ    | ㅍ   | ㅊ   | ㅌ         | ㅍ         |

|   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 |   |   | ㅍ | ㅊ | ㅌ | ㅍ | ㅊ | ㅌ | ㅍ | ㅊ | ㅌ | ㅍ | ㅊ | ㅌ | ㅍ | ㅊ |
| 5 | ㅈ | ㅊ | ㅌ | ㅍ | ㅊ | ㅌ | ㅍ | ㅊ | ㅌ | ㅍ | ㅊ | ㅌ | ㅍ | ㅊ | ㅌ | ㅍ |

## 1.1.7 – Arabic Set (AX-170)

|   | 0         | 1         | 2         | 3         | 4         | 5         | 6         | 7         | 8   | 9         | A         | B    | C   | D   | E         | F         |
|---|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----|-----------|-----------|------|-----|-----|-----------|-----------|
| 0 | NUL       | GRPH      | CTRL<br>B | CTRL<br>C | CTRL<br>D | CTRL<br>E | CTRL<br>F | BEEP      | BS  | TAB       | LF        | HOME | CLS | RET | CTRL<br>N | CTRL<br>O |
| 1 | CTRL<br>P | CTRL<br>Q | INS       | CTRL<br>S | CTRL<br>T | CTRL<br>U | CTRL<br>V | CTRL<br>W | SEL | CTRL<br>Y | CTRL<br>Z | ESC  | →   | ←   | ↑         | ↓         |
| 2 |           | !         | "         | #         | \$        | %         | &         | '         | (   | )         | *         | +    | ,   | -   | .         | /         |
| 3 | 0         | 1         | 2         | 3         | 4         | 5         | 6         | 7         | 8   | 9         | :         | ;    | <   | =   | >         | ?         |
| 4 | @         | A         | B         | C         | D         | E         | F         | G         | H   | I         | J         | K    | L   | M   | N         | O         |
| 5 | P         | Q         | R         | S         | T         | U         | V         | W         | X   | Y         | Z         | [    | \   | ]   | ^         | _         |
| 6 | `         | a         | b         | c         | d         | e         | f         | g         | h   | i         | j         | k    | l   | m   | n         | o         |
| 7 | p         | q         | r         | s         | t         | u         | v         | w         | x   | y         | z         | {    |     | }   | ~         | DEL       |
| 8 |           | !         | "         | #         | \$        | %         | &         | '         | (   | )         | *         | +    | ,   | -   | .         | /         |
| 9 | *         | 1         | 2         | 3         | 4         | 5         | 6         | 7         | 8   | 9         | :         | ;    | <   | =   | >         | ?         |
| A | ا         | ب         | ج         | د         | هـ        | و         | ز         | ح         | ط   | ظ         | ع         | ف    | ق   | ك   | ل         | م         |
| B | ن         | س         | ش         | ص         | ض         | ص         | ط         | ظ         | ع   | ف         | ق         | ك    | ل   | م   | ن         | س         |
| C | هـ        | و         | ز         | ح         | ط         | ظ         | ع         | ف         | ق   | ك         | ل         | م    | ن   | س   | ش         | ص         |
| D | ض         | ص         | ط         | ظ         | ع         | ف         | ق         | ك         | ل   | م         | ن         | س    | ش   | ص   | ض         | هـ        |
| E | و         | ز         | ح         | ط         | ظ         | ع         | ف         | ق         | ك   | ل         | م         | ن    | س   | ش   | ص         | ض         |
| F | هـ        | و         | ز         | ح         | ط         | ظ         | ع         | ف         | ق   | ك         | ل         | م    | ن   | س   | ش         | ص         |

|   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 5 | é | à | á | ç | ê | ë | è | í | î | ô | û | ü | ö | ° |   |   |



## 1.2 – KEYBOARD MATRICES

### 1.2.1 – Japanese Matrix

|         | bit 7 | bit 6 | bit 5 | bit 4 | bit 3 | bit 2 | bit 1 | bit 0 |
|---------|-------|-------|-------|-------|-------|-------|-------|-------|
| Col. 0  | 7 &   | 6 ^   | 5 %   | 4 \$  | 3 #   | 2 @   | 1 !   | 0 )   |
| Col. 1  | ; :   | ] }   | [ {   | \     | = +   | - _   | 9 (   | 8 *   |
| Col. 2  | b B   | a A   | _     | / ?   | . >   | , <   | ' ~   | ` ``  |
| Col. 3  | j J   | i I   | h H   | g G   | f F   | e E   | d D   | c C   |
| Col. 4  | r R   | q Q   | p P   | o O   | n N   | m M   | l L   | k K   |
| Col. 5  | z Z   | y Y   | x X   | w W   | v V   | u U   | t T   | s S   |
| Col. 6  | F3    | F2    | F1    | かな    | CAPS  | GRAPH | CTRL  | SHIFT |
| Col. 7  | RET   | SLCT  | BS    | STOP  | TAB   | ESC   | F5    | F4    |
| Col. 8  | →     | ↓     | ↑     | ←     | DEL   | INS   | HOME  | SPACE |
| Col. 9  | Num4  | Num3  | Num2  | Num1  | Num0  | Num/  | Num+  | Num*  |
| Col. 10 | Num.  | Num,  | Num-  | Num9  | Num8  | Num7  | Num6  | Num5  |
| Col. 11 |       |       |       |       | 実行    |       | 取消    |       |

**Obs.1:** Column 11 is used only by Panasonic models FS-A1WX, FS-A1WSX and turbo R, for access to the internal software in ROM. “実行” means “select” and “取消” means “cancel”.

**Obs.2:** The “かな” position is the “KANJI” key and corresponds to the CODE key in the international version.

## 1.2.1.1 – Japanese Matrix with locked かな/KANA key

| JIS    | bit 7 | bit 6 | bit 5 | bit 4 | bit 3 | bit 2 | bit 1 | bit 0 |
|--------|-------|-------|-------|-------|-------|-------|-------|-------|
| Col. 0 | や     | お     | え     | う     | あ     | ふ     | め     | わ     |
| Col. 1 | れ     | °「    | ゝ     | ー     | へ     | ほ     | よ     | ゆ     |
| Col. 2 | こ     | ち     | ろ     | め・    | る。    | ね、    | む」    | け     |
| Col. 3 | ま     | に     | く     | き     | は     | い     | し     | そ     |
| Col. 4 | す     | た     | せ     | ら     | み     | も     | り     | の     |
| Col. 5 | っ     | ん     | さ     | て     | ひ     | な     | か     | と     |

| ANSI   | bit 7 | bit 6 | bit 5 | bit 4 | bit 3 | bit 2 | bit 1 | bit 0 |
|--------|-------|-------|-------|-------|-------|-------|-------|-------|
| Col. 0 | に     | な     | お     | え     | う     | い     | あ     | の     |
| Col. 1 | も     | ろ「    | れ     | る     | り     | ら     | ね     | ぬ     |
| Col. 2 | と     | さ     | ん・    | を。    | わ、    | よ     | °」    | ゝー    |
| Col. 3 | み     | ふ     | ま     | そ     | せ     | く     | す     | っ     |
| Col. 4 | け     | か     | ほ     | へ     | や     | ゆ     | め     | む     |
| Col. 5 | た     | は     | ち     | き     | て     | ひ     | こ     | し     |

| GRAPH  | bit 7 | bit 6 | bit 5 | bit 4 | bit 3 | bit 2 | bit 1 | bit 0 |
|--------|-------|-------|-------|-------|-------|-------|-------|-------|
| Col. 0 | 土     | 金     | 木     | 水     | 火     | 月     | 日     | 万     |
| Col. 1 | ♣     | ○     |       | 円     |       | ー     | 千     | 百     |
| Col. 2 | 」     |       | ◆     | ♠     | 大     | 小     | ●     | ♥     |
| Col. 3 |       |       | 時     | 十     | 十     | 「     | ト     | ㄥ     |
| Col. 4 | 〒     |       | π     |       |       | 分     | 中     |       |
| Col. 5 |       | 年     | ×     |       | ⊥     |       | ㄣ     | 秒     |



## 1.2.2 – PX-7 Matrix

|        | bit 7 | bit 6 | bit 5   | bit 4 | bit 3 | bit 2 | bit 1 | bit 0 |
|--------|-------|-------|---------|-------|-------|-------|-------|-------|
| Col. 0 | 7 &   | 6 ^   | 5 %     | 4 \$  | 3 #   | 2 @   | 1 !   | 0 )   |
| Col. 1 | ; :   | ] }   | [ {     | \     | = +   | - _   | 9 (   | 8 *   |
| Col. 2 | b B   | a A   | , " \ ^ | / ?   | . >   | , <   | ' ~   | ` ``  |
| Col. 3 | j J   | i I   | h H     | g G   | f F   | e E   | d D   | c C   |
| Col. 4 | r R   | q Q   | p P     | o O   | n N   | m M   | l L   | k K   |
| Col. 5 | z Z   | y Y   | x X     | w W   | v V   | u U   | t T   | s S   |
| Col. 6 | F3    | F2    | F1      | かな    | CAPS  | GRAPH | CTRL  | SHIFT |
| Col. 7 | RET   | SLCT  | BS      | STOP  | TAB   | ESC   | F5    | F4    |
| Col. 8 | →     | ↓     | ↑       | ←     | DEL   | INS   | HOME  | SPACE |
| Col. 9 |       |       |         |       |       | SupI  | Video | Comp  |

**SupI** → **Superimpose**

**Video** → **Video**

**Comp** → **Computer**

**Obs.:** The PX-7 does not have a separate numeric keypad.

### 1.2.3 – Internacional Matrix

|         | bit 7 | bit 6 | bit 5 | bit 4 | bit 3 | bit 2 | bit 1 | bit 0 |
|---------|-------|-------|-------|-------|-------|-------|-------|-------|
| Col. 0  | 7 &   | 6 ^   | 5 %   | 4 \$  | 3 #   | 2 @   | 1 !   | 0 )   |
| Col. 1  | ; :   | ] }   | [ {   | \     | = +   | - _   | 9 (   | 8 *   |
| Col. 2  | b B   | a A   |       | / ?   | . >   | , <   | ' ~   | ` "   |
| Col. 3  | j J   | i I   | h H   | g G   | f F   | e E   | d D   | c C   |
| Col. 4  | r R   | q Q   | p P   | o O   | n N   | m M   | l L   | k K   |
| Col. 5  | z Z   | y Y   | x X   | w W   | v V   | u U   | t T   | s S   |
| Col. 6  | F3    | F2    | F1    | CODE  | CAPS  | GRAPH | CTRL  | SHIFT |
| Col. 7  | RET   | SLCT  | BS    | STOP  | TAB   | ESC   | F5    | F4    |
| Col. 8  | →     | ↓     | ↑     | ←     | DEL   | INS   | HOME  | SPACE |
| Col. 9  | Num4  | Num3  | Num2  | Num1  | Num0  | Num/  | Num+  | Num*  |
| Col. 10 | Num.  | Num,  | Num-  | Num9  | Num8  | Num7  | Num6  | Num5  |

### 1.2.4 – Brazilian Matrix (Expert 1.1 and Hotbit)

|         | bit 7 | bit 6  | bit 5 | bit 4 | bit 3 | bit 2 | bit 1 | bit 0 |
|---------|-------|--------|-------|-------|-------|-------|-------|-------|
| Col. 0  | 7 &   | 6 "    | 5 %   | 4 \$  | 3 #   | 2 @   | 1 !   | 0 )   |
| Col. 1  | ç Ç   | .. ' / | ' \   | \ ^   | = +   | - _   | 9 (   | 8 *   |
| Col. 2  | b B   | a A    | < >   | / ?   | . :   | , ;   | [ ]   | ~ ^   |
| Col. 3  | j J   | i I    | h H   | g G   | f F   | e E   | d D   | c C   |
| Col. 4  | r R   | q Q    | p P   | o O   | n N   | m M   | l L   | k K   |
| Col. 5  | z Z   | y Y    | x X   | w W   | v V   | u U   | t T   | s S   |
| Col. 6  | F3    | F2     | F1    | CODE  | CAPS  | GRAPH | CTRL  | SHIFT |
| Col. 7  | RET   | SLCT   | BS    | STOP  | TAB   | ESC   | F5    | F4    |
| Col. 8  | →     | ↓      | ↑     | ←     | DEL   | INS   | HOME  | SPACE |
| Col. 9  | 4     | 3      | 2     | 1     | 0     | /     | +     | *     |
| Col. 10 | .     | ,      | -     | 9     | 8     | 7     | 6     | 5     |

**Obs:** The Expert 1.0 uses the international matrix.

### 1.2.5 – Argentine / Spanish Matrix

|         | bit 7 | bit 6 | bit 5 | bit 4 | bit 3 | bit 2 | bit 1 | bit 0 |
|---------|-------|-------|-------|-------|-------|-------|-------|-------|
| Col. 0  | 7 &   | 6 ^   | 5 %   | 4 \$  | 3 #   | 2 @   | 1 !   | 0 )   |
| Col. 1  | ñ Ñ   | ] }   | [ {   | \     | = +   | - _   | 9 (   | 8 *   |
| Col. 2  | b B   | a A   |       | / ?   | . >   | , <   | ; :   | ' "   |
| Col. 3  | j J   | i I   | h H   | g G   | f F   | e E   | d D   | c C   |
| Col. 4  | r R   | q Q   | p P   | o O   | n N   | m M   | l L   | k K   |
| Col. 5  | z Z   | y Y   | x X   | w W   | v V   | u U   | t T   | s S   |
| Col. 6  | F3    | F2    | F1    | CODE  | CAPS  | GRAPH | CTRL  | SHIFT |
| Col. 7  | RET   | SLCT  | BS    | STOP  | TAB   | ESC   | F5    | F4    |
| Col. 8  | →     | ↓     | ↑     | ←     | DEL   | INS   | HOME  | SPACE |
| Col. 9  | Num4  | Num3  | Num2  | Num1  | Num0  | Num/  | Num+  | Num*  |
| Col. 10 | Num.  | Num,  | Num-  | Num9  | Num8  | Num7  | Num6  | Num5  |

**Obs.:** Only columns 1 and 2 differ from the international.

### 1.2.6 – United Kingdom Matrix (England)

|         | bit 7 | bit 6 | bit 5 | bit 4 | bit 3 | bit 2 | bit 1 | bit 0 |
|---------|-------|-------|-------|-------|-------|-------|-------|-------|
| Col. 0  | 7 &   | 6 ^   | 5 %   | 4 \$  | 3 #   | 2 @   | 1 !   | 0 )   |
| Col. 1  | ; :   | ] }   | [ {   | \     | = +   | - _   | 9 (   | 8 *   |
| Col. 2  | b B   | a A   | £     | / ?   | . >   | , <   | ` ~   | ' "   |
| Col. 3  | j J   | i I   | h H   | g G   | f F   | e E   | d D   | c C   |
| Col. 4  | r R   | q Q   | p P   | o O   | n N   | m M   | l L   | k K   |
| Col. 5  | z Z   | y Y   | x X   | w W   | v V   | u U   | t T   | s S   |
| Col. 6  | F3    | F2    | F1    | CODE  | CAPS  | GRAPH | CTRL  | SHIFT |
| Col. 7  | RET   | SLCT  | BS    | STOP  | TAB   | ESC   | F5    | F4    |
| Col. 8  | →     | ↓     | ↑     | ←     | DEL   | INS   | HOME  | SPACE |
| Col. 9  | Num4  | Num3  | Num2  | Num1  | Num0  | Num/  | Num+  | Num*  |
| Col. 10 | Num.  | Num,  | Num-  | Num9  | Num8  | Num7  | Num6  | Num5  |

**Obs.:** Only column 2 differ from the international.

## 1.2.7 – Russian Matrix

|         | bit 7 | bit 6 | bit 5 | bit 4 | bit 3 | bit 2 | bit 1 | bit 0 |
|---------|-------|-------|-------|-------|-------|-------|-------|-------|
| Col. 0  | & 6   | % 5   | ¤ 4   | # 3   | " 2   | ! 1   | + ;   | ) 9   |
| Col. 1  | v V   | * :   | h H   | - ^   | = _   | \$ 0  | ( 8   | ' 7   |
| Col. 2  | i I   | f F   | ? /   | < ,   | @     | b B   | > .   | \     |
| Col. 3  | o O   | [ ]   | r R   | p P   | a A   | u U   | w W   | s S   |
| Col. 4  | k K   | j J   | z Z   | ] }   | t T   | x X   | d D   | l L   |
| Col. 5  | q Q   | n N   | ~     | c C   | m M   | g G   | e E   | y Y   |
| Col. 6  | F3    | F2    | F1    | PYC   | CAPS  | GRAPH | CTRL  | SHIFT |
| Col. 7  | RET   | SLCT  | BS    | STOP  | TAB   | ESC   | F5    | F4    |
| Col. 8  | →     | ↓     | ↑     | ←     | DEL   | INS   | HOME  | SPACE |
| Col. 9  | Num4  | Num3  | Num2  | Num1  | Num0  | Num/  | Num+  | Num*  |
| Col. 10 | Num.  | Num,  | Num-  | Num9  | Num8  | Num7  | Num6  | Num5  |

## 1.2.7.1 – Russian Matrix with locked PYC/CODE key

|        | bit 7 | bit 6 | bit 5 | bit 4 | bit 3 | bit 2 | bit 1 | bit 0 |
|--------|-------|-------|-------|-------|-------|-------|-------|-------|
| Col. 0 | & 6   | % 5   | ¤ 4   | # 3   | " 2   | ! 1   | + ;   | ) 9   |
| Col. 1 | ж Ж   | * :   | х Х   | ъ Ъ   | = _   | \$ 0  | ( 8   | ' 7   |
| Col. 2 | и И   | ф Ф   | ? /   | < ,   | ю Ю   | б Б   | > .   | э Э   |
| Col. 3 | о О   | ш Ш   | р Р   | п П   | а А   | у У   | в В   | с С   |
| Col. 4 | к К   | й Й   | э Э   | щ Щ   | т Т   | ь Ъ   | д Д   | л Л   |
| Col. 5 | я Я   | н Н   | ч Ч   | ц Ц   | м М   | г Г   | е Е   | ы Ы   |

## 1.2.8 – Korean Matrix

|         | bit 7                                   | bit 6 | bit 5 | bit 4 | bit 3 | bit 2 | bit 1 | bit 0 |
|---------|---|-------|-------|-------|-------|-------|-------|-------|
| Col. 0  | 7 ' 6 & 5 % 4 \$ 3 # 2 " 1 ! 0 0        |       |       |       |       |       |       |       |
| Col. 1  | ; + [ { @ ` ₩   ^ ~ - = 9 ) 8 (         |       |       |       |       |       |       |       |
| Col. 2  | b B a A _ / ? . > , < ] } : *           |       |       |       |       |       |       |       |
| Col. 3  | j J i I h H g G f F e E d D c C         |       |       |       |       |       |       |       |
| Col. 4  | r R q Q p P o O n N m M l L k K         |       |       |       |       |       |       |       |
| Col. 5  | z Z y Y x X w W v V u U t T s S         |       |       |       |       |       |       |       |
| Col. 6  | F3 F2 F1 한글 CAPS GRAPH CTRL SHIFT       |       |       |       |       |       |       |       |
| Col. 7  | RET SLCT BS STOP TAB ESC F5 F4          |       |       |       |       |       |       |       |
| Col. 8  | → ↓ ↑ ← DEL INS HOME SPACE              |       |       |       |       |       |       |       |
| Col. 9  | Num4 Num3 Num2 Num1 Num0 Num/ Num+ Num* |       |       |       |       |       |       |       |
| Col. 10 | Num. Num, Num- Num9 Num8 Num7 Num6 Num5 |       |       |       |       |       |       |       |

## 1.2.8.1 – Korean Matrix with locked 한글/CODE key

|        | bit 7               | bit 6 | bit 5 | bit 4 | bit 3 | bit 2 | bit 1 | bit 0 |
|--------|---------------------|-------|-------|-------|-------|-------|-------|-------|
| Col. 0 |                     |       |       |       |       |       |       |       |
| Col. 1 |                     |       |       |       |       |       |       |       |
| Col. 2 | π □                 |       |       |       |       |       |       |       |
| Col. 3 | ㅏ ㅑ ㅓ ㅕ ㅗ ㅛ ㅜ ㅝ ㅟ ㅡ |       |       |       |       |       |       |       |
| Col. 4 | ㅓ ㅕ ㅗ ㅛ ㅜ ㅝ ㅟ ㅡ     |       |       |       |       |       |       |       |
| Col. 5 | ㅓ ㅕ ㅗ ㅛ ㅜ ㅝ ㅟ ㅡ     |       |       |       |       |       |       |       |

### 1.2.9 – Arabic Matrix

|         | bit 7 | bit 6 | bit 5 | bit 4 | bit 3 | bit 2 | bit 1 | bit 0 |
|---------|-------|-------|-------|-------|-------|-------|-------|-------|
| Col. 0  | 7 &   | 6 ^   | 5 %   | 4 \$  | 3 #   | 2 @   | 1 !   | 0 )   |
| Col. 1  | ; :   | ] }   | [ {   | \     | = +   | - _   | 9 (   | 8 *   |
| Col. 2  | b B   | a A   |       | / ?   | . >   | , <   | ' ~   | ` "   |
| Col. 3  | j J   | i I   | h H   | g G   | f F   | e E   | d D   | c C   |
| Col. 4  | r R   | q Q   | p P   | o O   | n N   | m M   | l L   | k K   |
| Col. 5  | z Z   | y Y   | x X   | w W   | v V   | u U   | t T   | s S   |
| Col. 6  | F3    | F2    | F1    | CODE  | CAPS  | GRAPH | CTRL  | SHIFT |
| Col. 7  | RET   | SLCT  | BS    | STOP  | TAB   | ESC   | F5    | F4    |
| Col. 8  | →     | ↓     | ↑     | ←     | DEL   | INS   | HOME  | SPACE |
| Col. 9  | Num4  | Num3  | Num2  | Num1  | Num0  | Num/  | Num+  | Num*  |
| Col. 10 | Num.  | Num,  | Num-  | Num9  | Num8  | Num7  | Num6  | Num5  |

#### 1.2.9.1 – Arabic matrix with Arabic mode activated

|        | bit 7 | bit 6 | bit 5 | bit 4 | bit 3 | bit 2 | bit 1 | bit 0 |
|--------|-------|-------|-------|-------|-------|-------|-------|-------|
| Col. 0 | ٧     | ٦     | ٥     | ٤     | ٣     | ٢     | ١     | ٠     |
| Col. 1 | ك     |       | ج     |       |       |       | ٩     | ٨     |
| Col. 2 | لا    | آ ش   |       | ؟     |       | ،     | ؛     | ”     |
| Col. 3 | ت     | هـ °  | أ ا   | ل لا  | ب     | ث °   | نـ ي  | ذ د   |
| Col. 4 | ق °   | ض °   | ح [   | خ °   | لآ ة  | و °   | م     | ن     |
| Col. 5 | ظ ط   | غ °   | ى °   | ص °   | ز ر   | ع °   | ف °   | ش !   |

## 1.3 – KEYBOARD LAYOUTS

### 1.3.1 – Internacional Layout

|       |   |      |   |       |   |       |   |   |   |   |   |   |        |       |  |
|-------|---|------|---|-------|---|-------|---|---|---|---|---|---|--------|-------|--|
| ESC   | ! | @    | # | \$    | % | ^     | & | * | ( | ) | - | + |        | BS    |  |
| TAB   | Q | W    | E | R     | T | Y     | U | I | O | P | { | } | RETURN |       |  |
| CTRL  | A | S    | D | F     | G | H     | J | K | L | : | " | ~ | ↵      |       |  |
| SHIFT |   | Z    | X | C     | V | B     | N | M | < | > | ? | ' | ^      | SHIFT |  |
|       |   | CAPS |   | GRAPH |   | SPACE |   |   |   |   |   |   | CODE   |       |  |

### 1.3.2 – Japanese Layout (JIS)

|       |   |      |       |       |   |   |   |     |   |   |   |   |    |        |
|-------|---|------|-------|-------|---|---|---|-----|---|---|---|---|----|--------|
| ESC   | 1 | 2    | 3     | 4     | 5 | 6 | 7 | 8   | 9 | 0 | = | ~ |    | BS     |
|       | め | ふ    | あ     | う     | え | お | や | (ゆ) | よ | わ | - | ^ | ¥  |        |
| TAB   | Q | W    | E     | R     | T | Y | U | I   | O | P | @ | . | {  | RETURN |
|       | た | て    | い     | す     | か | ん | な | に   | ら | せ |   |   | [  |        |
| CTRL  | A | S    | D     | F     | G | H | J | K   | L | ; | * | } | ]  |        |
|       | ち | と    | し     | は     | き | く | ま | の   | り | : | れ | : | む  |        |
| SHIFT | Z | X    | C     | V     | B | N | M | <   | > | ? | ~ | - |    | SHIFT  |
|       | づ | ざ    | そ     | ひ     | こ | み | も | 、   | 、 | 、 | 、 | ろ |    |        |
|       |   | CAPS | GRAPH | SPACE |   |   |   |     |   |   |   |   | かな |        |

### 1.3.3 – Japanese Layout (ANSI)

|       |    |      |       |       |    |    |    |    |    |    |    |     |    |        |
|-------|----|------|-------|-------|----|----|----|----|----|----|----|-----|----|--------|
| ESC   | 1あ | 2い   | 3う    | 4え    | 5お | 6な | 7に | 8ぬ | 9ね | 0の | =ら | ~へり | ¥る | BS     |
| TAB   | Qか | Wき   | Eく    | Rけ    | Tこ | Yは | Uひ | Iふ | Oへ | Pほ | @れ | {「  | 」ろ | RETURN |
| CTRL  | Aさ | Sし   | Dす    | Fせ    | Gそ | Hま | Jみ | Kむ | Lめ | +; | *: | -`  | }」 | ←      |
| SHIFT | Zた | Xち   | Cっ    | Vて    | Bと | Nや | Mゆ | <よ | >わ | ?/ | を  | -ん  |    | SHIFT  |
|       |    | CAPS | GRAPH | SPACE |    |    |    |    |    |    |    |     | かな |        |







### 1.3.10 – Arabic Layout (AX-170)

|       |        |        |        |         |         |        |        |         |        |        |        |        |      |       |        |
|-------|--------|--------|--------|---------|---------|--------|--------|---------|--------|--------|--------|--------|------|-------|--------|
| ESC   | 1<br>! | 2<br>@ | 3<br># | 4<br>\$ | 5<br>%  | 6<br>^ | 7<br>& | 8<br>*  | 9<br>( | 0<br>) | .      | -      | +    |       | BS     |
| TAB   | Q<br>ض | W<br>ص | E<br>ث | R<br>ق  | T<br>ف  | Y<br>غ | U<br>ع | I<br>هـ | O<br>خ | P<br>ح | [<br>ج | {<br>[ | }    | ]     | RETURN |
| CTRL  | A<br>آ | S<br>ش | D<br>ذ | F<br>ب  | G<br>ل  | H<br>ا | J<br>ت | K<br>ن  | L<br>م | :      | ;      | "      | “    | ~     | ↵      |
| SHIFT | Z<br>ط | X<br>ى | C<br>د | V<br>ر  | B<br>لا | N<br>ة | M<br>و | <       | >      | ?      | !      | /      | °    | SHIFT |        |
|       | CAPS   |        | GRAPH  |         | SPACE   |        |        |         |        |        |        |        | CODE |       |        |

### 1.3.11 – French Layout (ML-F80)

|       |        |        |        |        |        |        |        |        |        |        |   |   |      |       |        |
|-------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|---|---|------|-------|--------|
| ESC   | 1<br>& | 2<br>é | 3<br>“ | 4<br>’ | 5<br>( | 6<br>§ | 7<br>è | 8<br>! | 9<br>ç | 0<br>à | ° | - | >    | <     | BS     |
| TAB   | A      | Z      | E      | R      | T      | Y      | U      | I      | O      | P      | ¨ | * | \$   |       | RETURN |
| CTRL  | Q      | S      | D      | F      | G      | H      | J      | K      | L      | M      | % | ù | £    | #     | ↵      |
| SHIFT | W      | X      | C      | V      | B      | N      | M      | ?      | ,      | :      | / | + | =    | SHIFT |        |
|       | CAPS   |        | GRAPH  |        | SPACE  |        |        |        |        |        |   |   | CODE |       |        |

### 1.3.12 – German Layout (HB-F700D)

|       |        |        |        |         |        |        |        |        |        |   |        |        |        |  |
|-------|--------|--------|--------|---------|--------|--------|--------|--------|--------|---|--------|--------|--------|--|
| ESC   | 1<br>! | 2<br>" | 3<br>§ | 4<br>\$ | 5<br>% | 6<br>& | 7<br>/ | 8<br>( | 9<br>) | = | ?<br>ß | `<br>´ | BS     |  |
| TAB   | Q      | W      | E      | R       | T      | Y      | U      | I      | O      | P | Ü      | *<br>+ | RETURN |  |
| CTRL  | A      | S      | D      | F       | G      | H      | J      | K      | L      | Ö | Ä      | ^<br># | ↵      |  |
| SHIFT | ><br>< | Z      | X      | C       | V      | B      | N      | M      | ;      | : | -      | SHIFT  |        |  |
|       |        | CAPS   | GRAPH  | SPACE   |        |        |        |        |        |   |        |        | CODE   |  |

## 1.4 - CONTROL CODES

| Shortcut | DEC | HEX | Function  |
|----------|-----|-----|---|
| Ctrl+A   | 001 | 01H | Determines graphic character.                     |
| Ctrl+B   | 002 | 02H | Deflects cursor to start of the previous word.    |
| Ctrl+C   | 003 | 03H | Closes the entry condition.                       |
| Ctrl+D   | 004 | 04H |   |
| Ctrl+E   | 005 | 05H | Cancel character from cursor to end of line.      |
| Ctrl+F   | 006 | 06H | Deflect cursor to start the next word.            |
| Ctrl+G   | 007 | 07H | Generates a beep.                                 |
| Ctrl+H   | 008 | 08H | Deletes the letter before the cursor (BS).        |
| Ctrl+I   | 009 | 09H | Move cursor to the next TAB position (TAB).       |
| Ctrl+J   | 010 | 0AH | Line change (Linefeed).                           |
| Ctrl+K   | 011 | 0BH | Returns cursor to position 1.1 (HOME).            |
| Ctrl+L   | 012 | 0CH | Clears the screen and puts the cursor in 1.1 pos. |
| Ctrl+M   | 013 | 0DH | Carriage Return (RETURN).                         |
| Ctrl+N   | 014 | 0EH | Moves the cursor to the end of the line.          |
| Ctrl+O   | 015 | 0FH |   |
| Ctrl+P   | 016 | 10H |   |
| Ctrl+Q   | 017 | 11H |   |
| Ctrl+R   | 018 | 12H | Turn on/off insertion mode (INS).                 |
| Ctrl+S   | 019 | 13H |   |
| Ctrl+T   | 020 | 14H |   |
| Ctrl+U   | 021 | 15H | Delete the entire line on which the cursor is.    |
| Ctrl+V   | 022 | 16H |   |
| Ctrl+W   | 023 | 17H |   |
| Ctrl+X   | 024 | 18H | (SELECT).   |
| Ctrl+Y   | 025 | 19H |   |
| Ctrl+Z   | 026 | 1AH | (EOF) – End of File.                              |
| Ctrl+[   | 027 | 1BH | (ESC) – Escape.                                   |
| Ctrl+\   | 028 | 1CH | Moves the cursor to the right.                    |
| Ctrl+]   | 029 | 1DH | Moves the cursor to the left.                     |
| Ctrl+^   | 030 | 1EH | Move the cursor up.                               |
| Ctrl+_   | 031 | 1FH | Moves the cursor down.                            |
| Delete   | 127 | 7FH | Deletes the character under the cursor (DEL).     |

## 2 – I/O PORTS MAP

|         |  |
|---------|--|
| 00H~01H | Music Module MIDI (output) port (do not use at the same time with Sony Sensor Kid Cartridge).  |
| 00H~01H | Sony Sensor Kid Cartridge (do not use with Music Module).  |
| 02H~03H | FAC MIDI Interface (mirrored at 00H~07H).  |
| 04H~05H | Music Module MIDI (input).   |
| 00H~07H | MD Telcom modem.   |
| 08H~09H | No known use.  |
| 0AH     | DAC of the Music Module.   |
| 08H~0EH | No known use.  |
| 0FH     | MegaRAM Zemina.  |
| 10H~11H | PSG emulation for MegaflashROM in FPGA.  |
| 12H~13H | No known use.  |
| 14H~17H | YM2608 OPNA Cartridge.   |
| 18H~19H | Philips NMS 1170/20 barcode reader.  |
| 1AH~1FH | No known use.  |
| 20H~28H | Philips Modem NMS1251 (config. 30H~38H via jumper).<br>Miniware M4000 modem (config. 30H~38H via jumper).  |
| 21H~27H | Sunrise MP3 player.  |
| 27H~2FH | Philips NMS serial interface 1210/1211/1212.<br>(configurable in 37H~3FH via jumper).  |
| 28H~29H | DenYoNet ethernet interface.   |
| 2AH~2BH | PlaySoniq cartridge (setting registers).   |
| 30H~38H | Philips Modem NMS1251 (config. 20H~28H via jumper).<br>Miniware M4000 modem (config. 20H~28H via jumper).<br>Green-Mak SCSI interface.<br>Philips NMS 0210 CD-ROM interface. |
| 37H~3FH | Philips NMS serial interface 1210/1211/1212<br>(configurable in 27H~2FH via jumper).   |
| 3CH     | Musical Memory Mapper control register.  |
| 3FH     | Register of the SN76489 of the Musical Memory Mapper.  |
| 40H~4FH | Access to switchable I/O ports.  |
| 40H     | (R/W) Device ID.   |
| 41H~4FH | (R/W) access to the device.  |
| 48H~49H | Franky Cartridge (SN76489 and VDP).  |
| 50H~5DH | No known use.  |
| 5EH~5FH | GR8NET interface (Ethernet).   |

|           |   |
|-----------|---|
| 60H~6FH   | VDP V9990:                                  |
| 60H (R/W) | Access to VRAM.                             |
| 61H (R/W) | Access to the color palette.                |
| 62H (R/W) | Access to hardware commands.                |
| 63H (R/W) | Access to registers.                        |
| 64H (W)   | Selection of registers.                     |
| 65H (R)   | Status port.                                |
| 66H (W)   | Interrupt flag.                             |
| 67H (W)   | System control.                             |
| 68H (W)   | Address of Kanji-ROM (low) – 1.             |
| 69H (R/W) | Kanji-ROM address (high) and data – 1.      |
| 6AH (W)   | Address of the Kanji-ROM (low) – 2.         |
| 6BH (R/W) | Kanji-ROM (high) address and data – 2.      |
| 6CH~6FH   | Not used.                                   |
| 70H~73H   | Saurus MIDI Cartridge.                      |
| 74H~76H   | No known use.                               |
| 77H       | Super Game 90.                              |
| 78H~7BH   | No known use.                               |
| 7CH~7DH   | MSX-MUSIC (YM2413):                         |
| 7CH (W)   | Selects registers.                          |
| 7DH (W)   | Data port.                                  |
| 7EH~7FH   | Moonsound Cartridge (OPL4) – PCM synthesis: |
| 7EH       | PCM registers (wave).                       |
| 7FH       | PCM data (wave).                            |
| 80H~87H   | Standard RS232C serial interface:           |
| 80H (R/W) | USART 8251 – Data logger.                   |
| 81H (R/W) | USART 8251 – Status and command log.        |
| 82H (R/W) | USART 8251 – Status / communication.        |
| 83H (R/W) | Interrupt mask.                             |
| 84H (R/W) | 8253 – Counter 1.                           |
| 85H (R/W) | 8253 – Counter 2.                           |
| 86H (R/W) | 8253 – Counter 3.                           |
| 87H (W)   | Meter control.                              |
| 88H~8BH   | Access to external V9938.                   |
| 8CH~8DH   | MSX Modem.                                  |
| 8EH~8FH   | Megaram:                                    |
| 8EH       | Page selection.                             |
| 8FH       | Megaram-Disk.                               |

|         |   |
|---------|---|
| 90H~91H | Printer:<br>90H (R) Status.<br>91H (W) Data.  |
| 92H~93H | No known use.   |
| 94H     | Direction to printer port (non-standard).   |
| 95H~97H | No known use.   |
| 98H~9BH | VDP TMS9918 / V9938 / V9958:<br>98H (R/W) Read /Write data in VRAM.<br>99H (R/W) Read status register;<br>Write to the control register.<br>9AH (W) Writes to the palette registers.<br>9BH (W) Write in indirectly specified register. |
| 9CH~9FH | No known use.   |
| A0H~A2H | PSG AY-3-8910:<br>A0H (W) Address port.<br>A1H (W) Data writing port.<br>A2H (R) Data readout port.   |
| A3H     | No known use.   |
| A4H~A5H | PCM (Turbo R):<br>A4H (R/W) Data port.<br>A5H (R/W) Control port.   |
| A6H     | No known use.   |
| A7H     | Controls panel lights on the MSX turbo R:<br>bit 1 = LED Pause.<br>bit 7 = turbo LED.   |
| A8H~ABH | PPI 8255:<br>A8H (R/W) PPI port A (slot selection).<br>A9H (R/W) PPI port B (keyboard reading).<br>AAH (R/W) PPI port C (keyboard line / click keys).<br>ABH (W) PPI control port.  |
| ACH~AFH | MSX-Engine (1chipMSX control).  |
| B0H~B3H | Memory expansion (SONY 8255 specification):<br>B0H Address lines A0~A7.<br>B1H Address lines A8~A10, A13~A15, control, R/W.<br>B2H Address lines A11~A12 and data D0~D7.  |
| B4H~B5H | Clock IC (RP-5C01):<br>B4H Address of the registers.<br>B5H Read/write data.  |

|         |  |
|---------|--|
| B6H~B7H | Card reader?   |
| B8H~BBH | Lightpen control (SANYO specification).  |
| BCH~BFH | VHD Control (JVC 8255 specification).  |
| C0H~C1H | MSX-Audio Y8950:<br>C0H (R/W) Selects regs and reads reg. status.<br>C1H (R/W) Write or read reg. specified.   |
| C0H~C3H | Alternative ports for Moonsound / OPL4.  |
| C4H~C7H | Moonsound Cartridge (OPL4) – FM synthesis:<br>C4H FM register array 0 (bank 1) and reg. status.<br>C5H FM (data).<br>C6H FM register array 1 (bank 2).<br>C7H Mirror of (access via C5H is preferred). |
| C8H~CCH | Asynchronous serial interface.   |
| CDH~CFH | No known use.  |
| D0H~D7H | Reserved for disk interface.   |
| D8H~D9H | Kanji-ROM Jis 1:<br>D8H (W) Address lines A0~A5.<br>D9H (R/W) Address lines A6~A11 and data D0~D7.   |
| DAH~DBH | Kanji-ROM Jis 2:<br>DAH (W) Address lines A0~A5.<br>DBH (R/W) Address lines A6~A11 and data D0~D7.   |
| DCH~DDH | Playsoniq Cartridge (Sega Gamepad support).  |
| DEH~DFH | No known use.  |
| E0H~E2H | MSX-MIDI external:<br>E0H Data transmission / reception.<br>E1H Control port.<br>E2H Selection port.   |
| E3H     | No known use.  |
| E4H~E7H | Access to the S1990 (MSX turbo R):<br>E4H Registers addresses.<br>E5H Data.<br>E6H 16-bit counter (LSB) and counter reset.<br>E7H 16-bit counter (MSB).  |
| E8H~EFH | MSX-MIDI:<br>E8H Data transmission / reception.<br>E9H Control port.<br>EAH Latch of signals (written only).<br>EBH Mirror from EAH.   |

|         |                     |  |
|---------|---------------------|--|
|         | ECH                 | Counter 0.   |
|         | EDH                 | Counter 1.   |
|         | EEH                 | Counter 2.   |
|         | EFH                 | Control of counters (write only).  |
| F0H~F2H |                     | No known use.  |
| F3H     |                     | Current screen mode (MSX2+ only):  |
|         | bit 0 = M3          | bit 4 = M1   |
|         | bit 1 = M4          | bit 5 = TP   |
|         | bit 2 = M5          | bit 6 = YUV  |
|         | bit 3 = M2          | bit 7 = YAE  |
| F4H     |                     | RESET status for MSX2+ and MSX turbo R:  |
|         | bit 5               | Flag to indicate that the system is already initialized.   |
|         | bit 7 - 0           | = hard reset; 1 = Soft reset.  |
|         |                     | Note: in some MSX2+, the read data must be inverted to obtain the correct value.   |
| F5H     |                     | System control (setting the bit to 1 enables):   |
|         | b0 - Kanji-ROM      | b4 - MSX-Interface   |
|         | b1 - Reserved Kanji | b5 - Serial RS232C   |
|         | b2 - MSX-Audio      | b6 - Lightpen  |
|         | b3 - Superimpose    | b7 - Clock IC  |
| F6H     |                     | Color I/O bus (Color Bus).   |
| F7H     |                     | AV Control (setting the bit to 1 enables):   |
|         | b0                  | - Simultaneous right and left audio.   |
|         | b1                  | - Audio L (left) only.   |
|         | b2                  | - Select video input (RGB21).  |
|         | b3                  | - Flag indicating whether there is a video input or not.   |
|         | b4                  | - AV Control (RGB21).  |
|         | b5                  | - Ym control (RGB21).  |
|         | b6                  | - Inverse of bit 4 of reg. # 9 of the VDP.   |
|         | b7                  | - Inverse of bit 5 of the VDP reg. # 9.  |
| F8H     |                     | Access to PAL A/V control register.  |
| F8H~FBH |                     | Access to the 8-bit MSB of the 16-bit register of the Playsoniq cartridge (they are the same ports used in some MSX that are disabled by default). |
| FCH~FFH |                     | Memory Mapper:   |
|         | FCH (R/W)           | Physical page 0 (0000H~3FFFFH).  |
|         | FDH (R/W)           | Physical page 1 (4000H~7FFFFH).  |
|         | FEH (R/W)           | Physical page 2 (8000H~BFFFFH).  |
|         | FFH (R/W)           | Physical page 3 (C000H~FFFFH).   |



## 3 – MSX-BASIC

### 3.1 – FORMAT

INSTRUCTION NAME (instruction type, BASIC version)

Format: Valid formats for the instruction.

Function: Form of operation of the instruction.

There are five types of instructions, namely: declarations, commands, functions, system variables and logical operators.

The BASIC version indicates the version for which the instruction is implemented. Values separated by “-” indicate that there are differences in syntax or behavior for different versions.

1~4 MSX-BASIC version  
 M MSX-MUSIC BASIC  
 K Kanji-ROM required  
 D Disk-BASIC 1.0  
 D2 Disk-BASIC 2.0

#### 3.1.1 – Instructions Abbreviations

REM ‘  
 PRINT ?  
 CALL \_

#### 3.1.2 – Logical Operation Codes

|        |         |  |
|--------|---------|--|
| PSET   | TPSET   | Uses the specified color (default)           |
| PRESET | TPRESET | Makes “NOT (color specified)”                |
| XOR    | TXOR    | Makes “(target color) XOR (specified color)” |
| OR     | TOR     | Makes “(target color) OR (specified color)”  |
| AND    | TAND    | Makes “(target color) AND (specified color)” |

Note: when the operation is preceded by "T", no operation will be performed when the color is transparent.

### 3.1.3 – Code notations

|    |  |
|----|--|
| &B | Precedes a constant in binary form       |
| &O | Precedes a constant in octal form        |
| &H | Precedes a constant in hexadecimal form  |
| %  | Marks variable as integer                |
| !  | Marks variable as simple precision       |
| #  | Marks variable as double precision       |
| \$ | Marks variable as alphanumeric           |
| -  | Mathematical operator for subtraction    |
| +  | Mathematical operator for addition       |
| /  | Mathematical operator for division       |
| *  | Mathematical operator for multiplication |
| ^  | Mathematical operator for potentiation   |
| =  | Denotes equality and assigns values      |
| <> | Denotes difference                       |

### 3.1.4 – Format Notations

|          |   |
|----------|---|
| <exprA>  | variable, constant, or string or numeric expression.                                  |
| <exprN>  | variable, constant or numeric expression.   |
| <expr\$> | variable, constant, or string expression.   |
| <n>      | is a defined number. When in parentheses it can be an expression or numeric variable. |
| [ ]      | delimits optional parameter.  |
|          | it means that only one of the items can be used.                                      |
| { }      | delimits option.  |
| X        | any variable.   |
| X%       | any integer variable.   |
| X!       | any single precision variable.  |
| X#       | any double precision variable.  |
| X\$      | any alphanumeric variable.  |

Characters in parentheses after multiple formats for an instruction indicate the version of BASIC in which that instruction format is available.

### 3.2 – INSTRUCTIONS DESCRIPTION

**ABS** (function, 1)

Format: X = ABS (<exprN>)

Function: Returns in X the absolute value (module) of <exprN>.

**AND** (logical operator, 1)

Format: <exprA1> AND <exprA2>

Function: Performs logical AND operation between <exprA1> and <exprA2>.

0 and 0 → 0      1 and 0 → 0  
0 and 1 → 0      1 and 1 → 1

**ASC** (function, 1)

Format: X = ASC (<expr\$>)

Function: Returns the ASCII code of the first character of expr\$ in X.

**ATN** (function, 1)

Format: X = ATN (<exprN>)

Function: Returns in X the arcotangent of exprN (exprN must be expressed in radians).

**AUTO** (command, 1)

Format: AUTO [numlline, [increment]]

Function: Automatically generates line numbers, starting with [numline] and incremented with the value of [increment].

**BASE** (system variable, 1-2-3)

Format: X = BASE (<n>)

BASE (<n>) = <exprN>

Function: Returns in X or sets the starting addresses of the tables in VRAM for each screen mode. <n> is an integer that follows the following table:

|            | SCREEN MODES |   |    |    |    |    |    |    |    |    |    |    | Table of ...       |
|------------|--------------|---|----|----|----|----|----|----|----|----|----|----|--------------------|
|            | 0            | 1 | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 10 | 11 | 12 |                    |
| BASE VALUE | 0            | 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 50 | 55 | 60 | pattern names      |
|            | -            | 6 | 11 | 16 | 21 | 26 | 31 | 36 | 41 | 51 | 56 | 61 | colors             |
|            | 2            | 7 | 12 | 17 | 22 | 27 | 32 | 37 | 42 | 52 | 57 | 62 | pattern generator  |
|            | -            | 8 | 13 | 18 | 23 | 28 | 33 | 38 | 43 | 53 | 58 | 63 | sprites attributes |
|            | -            | 9 | 14 | 19 | 24 | 29 | 34 | 39 | 44 | 54 | 59 | 64 | sprites generator  |

**BEEP** (declaration, 1)

Format: BEEP

Function: Generates a beep.

**BIN\$** (function, 1)

Format: X\$ = BIN\$ (<exprN>)

Function: Converts the value of <exprN> to a string of binary codes and returns the value obtained in X\$.

**BLOAD** (command, 1-D)

Format: BLOAD "<filename>" [,R [,<offset>]]

BLOAD "<filename>" [{,R | ,S}] [,<offset>]] (D)

Function: Load a binary block into RAM or, if specified [,S], into VRAM (Disk Basic only). If specified [,R], executes a program in machine code.

**BSAVE** (command, 1-D)

Format: BSAVE "<filename>",<endini>, <endfim> [,<endexec>]

BSAVE "<filename>",<endini>, <endfim> [,<endexec> [, S]]

Function: Saves a binary block to disk or tape. [,S] saves a VRAM block (option available only under Disk Basic).

**CALL** (declaration, 1-2-3-4-D-M-Nextor)

Format: CALL <extended command> [(<argument> [, argument>...])]

Function: Executes extended commands through ROM cartridges or routines loaded in RAM. See the section "DESCRIPTION OF EXTENDED COMMANDS".

**CDBL** (function, 1)

Format: X# = CDBL (<exprN>)

Function: Converts the value of <exprN> to a double precision value and returns the value obtained in the variable X#.

**CHR\$** (function, 1)

Format: X\$ = CHR\$ (<exprN>)

Function: Returns in X\$ the character whose ASCII code is expressed in <exprN>.

**CINT** (function, 1)

Format:  $X\% = \text{CINT}(\langle \text{exprN} \rangle)$

Function: Converts the value of  $\langle \text{exprN} \rangle$  to an integer number and loads it in the variable  $X\%$ .

**CIRCLE** (statement, 1-2)

Format:  $\text{CIRCLE} \{(X, Y) \mid \text{STEP}(X, Y)\}, \langle \text{radio} \rangle [, \langle \text{color} \rangle$   
 $[, \langle \text{start angle} \rangle [, \langle \text{end angle} \rangle [, \langle \text{aspect ratio} \rangle ]]]]$

Function: Draws a circle with a central point at (X, Y). If STEP is specified, the coordinates will be calculated from the current one.  $\langle \text{start angle} \rangle$  and  $\langle \text{end angle} \rangle$  must be specified in radians.  $\langle \text{proportion} \rangle$  is the relation for ellipse;  $\langle 1 \rangle$  being perfect circumference.

**CLEAR** (statement, 1)

Format:  $\text{CLEAR} [\langle \text{string area size} \rangle [, \text{upper memory limit}]]$

Function: Initializes the BASIC variables and sets the size of the string area and the upper limit of memory used by BASIC.

**CLOAD** (command, 1)

Format:  $\text{CLOAD} [\langle \text{filename} \rangle]$

Function: Loads a BASIC program from the tape.

**CLOAD?** (command, 1)

Format:  $\text{CLOAD?} [\langle \text{filename} \rangle]$

Function: Compares a BASIC program on the cassette with the one that is in the memory.

**CLOSE** (command, 1-D)

Format:  $\text{CLOSE} [[\#] \langle \text{file number} \rangle [, [\#] \langle \text{file number} \rangle \dots]]$

Function: Closes the specified files. If no file is specified, close all opened files.

**CLS** (declaration, 1)

Format: CLS

Function: Clears the screen.

**CMD** (command, 1)

Format: No format defined.

Function: Reserved for implementing new commands.

**COLOR** (statement, 1-2)

Format: COLOR [<front color> [,<background color>  
[,<border color>]] (1-2)

Function: Specifies the colors of the screen.

**COLOR** = (declaration, 2)

Format: COLOR = (<palette number>, <red level>, <green level>, <blue level>)

Function: Specifies the colors of the palette. The level can vary from 0 to 7 for each color.

**COLOR = NEW** (declaration, 2)

Format: COLOR [= NEW]

Function: Initializes the color palette.

**COLOR = RESTORE** (declaration, 2)

Format: COLOR = RESTORE

Function: Copies the contents of the color palette stored in VRAM to the VDP palette registers.

**COLOR SPRITE** (statement, 1-2)

Format: COLOR SPRITE (<sprite plane number>) = <color>

Function: Specifies the color of the sprites.

## COLOR SPRITES (declaration, 2)

Format: COLOR SPRITE\$ (<sprite plan number>) = <expr\$>

Function: Specifies the color of each line of the sprites.  
 <expr\$> = CHR\$ (1st line color) + CHR\$ (2nd line color) ...

**CONT** (command, 1)

Format: CONT

**Function:** Continues the execution of a program that was interrupted.

**COPY** (declaration, 1-2-D)

Format: COPY <filename1> [TO <filename2>] (1-D)

Function: Copy the contents of <filename1> to <filename2>.

Format: COPY (X1, X2) – (Y1, Y2) [, <source page>] TO (X3, Y3) [, <target page> [, <logical operation>]] (2)

Function: Copies a rectangular area of the screen to another.

**Format:** COPY (X1, X2) – (Y1, Y2) [*<source page>*] TO  
                                  *<matrix variable | <filename>*} (2-D)

Function: Copy the contents of a rectangular area of the screen to a matrix variable or to a file on disk.

Format: COPY {<matrix variable> | <filename>} [,<direction>] TO  
(X3, Y3) [,<dest page> [,<logical operation>]] (2-D)

Function: Copies the contents of a matrix variable or a disk file to a rectangular area on the screen.

Format: COPY <filename> TO <matrix variable> (2-D)

Function: Copies the contents of a file to an array variable.

Format: COPY <matrix variable> TO <filename> (2-D)

Function: Copies the contents of an array variable to a file.

**COPY SCREEN** (declaration, 2, optional)

Format: COPY SCREEN [<mode>]

Function: Writes the Color Bus data on VRAM.

<mode> can be:

0 – Scans the current video page.

1 – Scans two pages, the first on the page before the active one and the second on the active page (interlaced mode).

Note: requires a digitizer or superimposer.

**COS** (function, 1)

Format: X = COS (<exprN>)

Function: Returns in X the cosine value of <exprN> (exprN must be expressed in radians).

**CSAVE** (command, 1)

Format: CSAVE <filename> [,<baud rate>]

Function: Saves a BASIC program to the cassette.

## CSNG (function, 1)

Format: X! = CSNG (<exprN>)

Function: Converts the value of <exprN> to a simple precision value and store it in X!.

**CSRLIN** (system variable, 1)

Format: X = CSRLIN

Function: Contains the vertical position of the cursor.

**CVD** (function, D)

Format: X# = CVD (<8-byte string>)

Function: Converts the string to a double precision value and store it in X#.

**CVI** (function, D)

Format: X% = CVI (<2-byte string>)

Function: Convert the string to an integer value and store it in X%.

**CVS** (function, D)

Format: X! = CVS (<4-byte string>)

Function: Converts the string to a simple precision value and store it in X!.

**DATA** (declaration, 1)

Format: DATA <constant> [,<constant> ...]

Function: Stores a list of data for the READ command.

**DEF FN** (statement, 1)

**Format:** DEF FN<name> [(*<argument>* [*<argument>* ...])] =  
                    *<user function defining expression>*

Function: Defines a user function.

**DEFDBL** (statement, 1)

Format: DEFDBL <character range> [,<character range> ...]

Function: Declares the specified variables as double precision.

**DEFINT** (statement, 1)

Format: **DEFINT** <character range> [, <character range> ...]

Function: Declares the specified variables as integers.

**DEFSNG** (statement, 1)

Format: DEFSNG <character range> [,<character range> ...]

Function: Declares the specified variables as simple precision.

**DEFSTR** (statement, 1)

Format: DEFSTR <character range> [, <character range> ...]

Function: Declares the specified variables as strings.



**DEFUSR** (statement, 1)

Format: DEFUSR [<number>] = <address>

Function: Defines an initial address for executing an assembly program to be called by the USR function. <number> can vary from 0 to 9.

**DELETE** (command, 1)

Format: DELETE {<start line> – <end line> | <line> | – <end line>}

Function: Deletes the specified lines from the BASIC text.

**DIM** (declaration, 1)

Format: DIM <variable> (<max index> [,<max index> ...])

Function: Defines an variable array and allocates space in memory.

**DRAW** (macro declaration, 1)

Format: DRAW <expr\$>

Function: Draw a line according to <expr\$>. The valid commands for <expr\$> are as follows:

|      |                 |    |                      |
|------|-----------------|----|----------------------|
| Un   | to up           | Dn | to down              |
| Ln   | to left         | Rn | to right             |
| En   | up and right    | Fn | down and right       |
| Gn   | low and left    | Hn | up and left          |
| B    | move no drawing | N  | back to origin       |
| Mx,y | goes to X,Y     | An | rotates n*90 degrees |
| Sn   | scale n / 4     | Cn | color n              |

Xseries runs macro in series.

Ex. A\$ = "C15U10" → DRAW "XA\$"

= <variable> – Place a parameter as an integer after the command. Ex. A\$="C15U10", S=50,  
→ DRAW "XA\$;R=S;D=S"

**DSKF** (function, D)

Format: X = DSKF (<drive number>)

Function: Returns the free space on the specified drive in clusters. If Nextor is installed, it will return the space in Kbytes.

**EOF** (function, 1-D)

Format: X = EOF (<file number>)

Function: Returns –1 if the end of the file is detected.

**ERASE** (declaration, 1)

Format: ERASE <matrix variable> [, <matrix variable> ...]

Function: Delete the specified matrix variables.

**EQV** (logical operator, 1)

Format: <exprA1> EQV <exprA2>

Function: Performs EQV logical operation between <exprA1> and <exprA2>. The result will be 1 if the two bits are equal and zero if they are different.

0 eqv 0 → 1      1 eqv 0 → 0

0 eqv 1 → 0      1 eqv 1 → 1

**ERL** (system variable, 1)

Format: X = ERL

Function: Contains the line number where the last error occurred.

**ERR** (system variable, 1)

Format: X = ERR

Function: Contains the error code of the last error occurred.

**ERROR** (statement, 1)

Format: ERROR <error code>

Function: Puts the program in error condition.

**EXP** (function, 1)

Format: X = EXP (<exprN>)

Function: Returns in X the value of the natural potentiation of <exprN>.

**FIELD** (statement, D)

Format: FIELD [#]<file number>, <field size> AS <string variable>  
[,<field size> AS <string variable> ...]

Function: Assigns <string variable> for random disk access.

**FILES** (command, D)

Format: FILES [<filename>]

Function: Displays the directory of the disc according to <filename>. If <filename> is omitted, it displays the names of all files on the disk.

**FIX** (function, 1)

Format: X = FIX (<exprN>)

Function: Returns in X the entire part of <exprN>, without rounding.

**FOR** (statement, 1)

Format: FOR <variable name> = <initial value> TO <final value>  
[STEP <increment>]

Function: Repeat the execution of the section between FOR and NEXT.

**FRE** (function, 1)

Format: FRE (0 | “”)

Function: Returns the size of the remaining memory for the BASIC text (0) or for the string variables (“”).

**GET** (declaration, D)

Format: GET [#]<file number> [,<record number>]

Function: Reads a record from a random access file.

**GET DATE** (declaration, 2)

Format: GET DATE <string variable> [,A]

Function: Returns a string with the current date in the <string variable>. If “A” is specified, returns the alarm date.

**GET TIME** (statement, 2)

Format: GET TIME <string variable> [,A]

Function: Returns a string with the current time in the <string variable>. If “A” is specified, returns the alarm time.

**GOSUB** (statement, 1)

Format: GOSUB <line number>

Function: Calls a subroutine that starts at line <line no.>.

**GOTO** (statement, 1)

Format: GOTO <line number>

Function: Jump to line <line number>.

**HEX\$** (function, 1)

Format: X\$ = HEX\$ (<exprN>)

Function: Converts the value of <exprN> to a hexadecimal string and returns it in X\$.

**IF** (statement, 1)

Format: IF <condition> THEN {<command> | <line number>}  
[ELSE {<command> | <line number>}]

IF <condition> GOTO <line number> [ELSE <line number>]

Function: Executes commands according to <condition>.

**IMP** (logical operator, 1)

Format: <exprA1> IMP <exprA2>

Function: Performs IMP logical operation between <exprA1> and <exprA2>. The result will be 0 when the first bit is true and implies that the second is false. Otherwise, it will be 1.

$$0 \text{ imp } 0 = 1$$
$$1 \text{ imp } 0 = 0$$
$$0 \text{ imp } 1 = 1$$
$$1 \text{ imp } 1 = 1$$

**INKEY\$** (function, 1)

Format: X\$ = INKEY\$

Function: Returns the character in X\$ whose key is being pressed; otherwise, returns a null string.

**INP** (function, 1)

Format: X = INP (<port number>)

Function: Reads an I/O port from the Z80 and returns its value in X.

**INPUT** (statement, 1)

Format: INPUT ["<prompt>";<variable name> [,<variable name> ...]

Function: Reads a data entry using the keyboard and stores the obtained value(s) in the respective variable(s).

**INPUT#** (declaration, 1)

Format: INPUT# <file number>, <variable name> [, <variable name> ...]

Function: Read data from the specified file and store the obtained value(s) in the respective variable(s).

**INPUT\$** (function, 1)

Format: X\$ = INPUT\$ (<number of characters> [, [#]<file number>])

Function: Reads the specified number of characters from the keyboard or a file and stores the obtained value in X\$.

**INSTR** (function, 1)

Format:  $X = \text{INSTR} ([\langle \text{exprN} \rangle,] \langle \text{expr\$1} \rangle, \langle \text{expr\$2} \rangle)$

Function: Searches for the occurrence of  $\langle \text{expr\$2} \rangle$  in  $\langle \text{expr\$1} \rangle$  from the position  $\langle \text{exprN} \rangle$  and returns the value obtained in X. If  $\langle \text{expr\$1} \rangle$  is not found, returns 0.

**INT** (function, 1)

Format:  $X = \text{INT} (\langle \text{exprN} \rangle)$

Function: Returns in X the entire part of  $\langle \text{exprN} \rangle$ , rounding off.

**INTERVAL** (statement, 1)

Format:  $\text{INTERVAL} \{ \text{ON} \mid \text{OFF} \mid \text{STOP} \}$

Function: Activate, deactivate or suspend interruption for a period of time.

**IPL** (command, 1)

Format: No defined format.

Function: Reserved for implementing new commands.

**KEY** (command / declaration, 1)

Format:  $\text{KEY} \langle \text{key number} \rangle, \langle \text{expr\$} \rangle$

Function: Assign the content of the specified function key.

Format:  $\text{KEY} (\langle \text{key number} \rangle) \{ \text{ON} \mid \text{OFF} \mid \text{STOP} \}$

Function: Enables, disables or suspends function key interruption.

Format:  $\text{KEY} \{ \text{ON} \mid \text{OFF} \}$

Function: Turns on or off the display of the content of the function keys on the bottom screen line.

**KEY LIST** (command, 1)

Format:  $\text{KEY LIST}$

Function: Lists the content of the function keys.

**KILL** (command, D)

Format:  $\text{KILL} \langle \text{expr\$} \rangle$

Function: Delete files on the disk.  $\langle \text{expr\$} \rangle$  must contain a valid filename.

**LEFT\$** (function, 1)

Format:  $X\$ = \text{LEFT\$} (\langle \text{expr\$} \rangle, \langle \text{exprN} \rangle)$

Function: Returns the  $\langle \text{exprN} \rangle$  left  $\langle \text{expr\$} \rangle$  characters in  $X\$$ .

**LEN** (function, 1)

Format: X = LEN (<expr\$>)

Function: Returns in X the number of characters in <expr\$>.

**LET** (declaration, 1)

Format: [LET]<variable name> = <exprA>

Function: Stores the value of <exprA> in the variable.

**LFILES** (command, 1)

Format: LFILES [<filename>]

Function: Lists the filenames of the disk in the printer according to <filename>. If <filename> is omitted, it lists the names of all files on the disk.

**LINE** (statement, 1-2)

Format: LINE [{(X1, Y1) | STEP (X1, Y1)}] – {(X2, Y2) | STEP (X2, Y2)}  
[,<color> [, {B | BF} [,<logical operation>]]]

Function: Draws a line, an empty rectangle (,B) or a painted rectangle (,BF). The STEP subcommand, when specified, defines the offset.

**LINE INPUT** (statement, 1)

Format: LINE INPUT [<prompt>"];<string variable>

Function: Reads a sequence of characters from the keyboard and stores the value read in the <string variable>.

**LINE INPUT#** (declaration, 1-D)

Format: LINE INPUT#<file number>, <string variable>

Function: Reads a sequence of characters from a file and stores the value read in the <string variable>.

**LIST** (command, 1)

Format: LIST [[<start line>] – [<end line>]]

Function: List on the screen the BASIC program that is in memory.

**LLIST** (command, 1)

Format: LLIST [[<start line>] – [<end line>]]

Function: Lists the BASIC program in the printer.

**LOAD** (command, 1-D)

Format: LOAD "<filename>" [,R]

Function: Load a program into memory. The [,R] parameter makes the program to be executed after loading.

**LOC** (function, D)

Format: X = LOC (<file number>)

Function: Returns in X the number of the last accessed record of the file.

**LOCATE** (declaration, 1-2)

Format: LOCATE [<X coord.> [,<Y coord.> [,<cursor type>]]]

Function: Position the cursor on the text screens. If <cursor type> is 0 (default value) the cursor will not be displayed when the computer is busy; if any other value, the cursor will always be displayed.

**LOF** (function, D)

Format: X = LOF (<file number>)

Function: Returns the specified file size in X.

**LOG** (function, 1)

Format: X = LOG (<exprN>)

Function: Returns in X the natural logarithm of <exprN>.

**LPOS** (system variable, 1)

Format: X = LPOS

Function: Stores the horizontal location of the printer head.

**LPRINT** (declaration, 1)

Format: LPRINT [<exprA> [{; | } <exprA> ...]]

Function: Send the characters in the the expressions <exprA> to the printer. ";" prints the next character immediately after, "," prints the character in the next tab stop.

**LPRINT USING** (statement, 1)

Format: LPRINT USING "<format>"; <exprA> [{; | } <exprA>...]  
LPRINT USING "<expr\$ formatted>"

**Function:** Send to the printer the characters corresponding to the expressions <exprN> or <expr\$>, formatting. ";" prints the next character immediately after, "," prints the character at the next tab stop. The characters used to format the output are as follows:

→ Numeric formatting:

- # Space for one digit
- . Includes decimal point
- + Indicates + or -; used before or after the number
- Indicates -; used after the number
- \$\$ Put\$ to the left of the number
- \*\* Replaces leading spaces with asterisks
- \*\*\$ Places a\$ to the left preceded by asterisks
- ^^^^ Displays the number in scientific notation

→ Alphanumeric formatting:

- \ \ Character space
- ! Space for a character
- & Variable spacing
- \_ Next character is printed normally
- other Print character

### **LSET** (declaration, D)

**Format:** LSET <string variable> = <expr\$>

**Function:** Stores the contents of <expr\$> on the left of the variable string defined by the FIELD statement.

### **MAXFILES** (declaration, 1-D)

**Format:** MAXFILES = <number of files>

**Function:** Defines the maximum number of files that can be opened at the same time.

### **MERGE** (command, 1-D)

**Format:** MERGE <filename>

**Function:** Merges the program in memory with a program saved in ASCII format on disk or tape.

### **MID\$** (function / declaration, 1)

**Format:** X\$ = MID\$ (<expr\$>, <exprN1> [, <exprN2>])

**Function:** Returns, in X\$, <exprN2> characters from the character <exprN1> of <expr\$>.



**Format:** MID\$ (<string variable>, <exprN1> [,<exprN2>]) = <expr\$>

**Function:** Defines <expr\$> using <exprN2> characters from the <exprN1> position of the <string variable>.

**MKD\$** (function, D)

**Format:** X\$ = MKD\$ (<double precision value>)

**Function:** Convert a double precision value to an 8-byte string and store it in X\$.

**MKI\$** (function, D)

**Format:** X\$ = MKI\$ (<integer value>)

**Function:** Convert an integer value to a 2-byte string and store it in X\$.

**MKS\$** (function, D)

**Format:** X\$ = MKS\$ (<simple precision value>)

**Function:** Converts a simple precision value to a 4-byte string and store it in X\$.

**MOTOR** (control, 1)

**Format:** MOTOR [{ON | OFF}]

**Function:** Turns the cassette motor on or off.

**NAME** (command, D)

**Format:** NAME <filename1> AS <filename2>

**Function:** Rename the file <filename1> to <filename2>.

**NEW** (command, 1)

**Format:** NEW

**Function:** Deletes the program from memory and clears the variables.

**NEXT** (declaration, 1)

**Format:** NEXT [<variable name> [,<variable name>...]]

**Function:** Indicates the end of the FOR loop.

**NOT** (logical operator, 1)

**Format:** NOT (<exprA>)

**Function:** Performs the negation of <exprA>.

not 0 → 1

not 1 → 0

**OCT\$** (function, 1)

Format: X\$ = OCT\$ (<exprN>)

Function: Converts the value of <exprN> to an octal string and returns it in X\$.

**ON ERROR GOTO** (statement, 1)

Format: ON ERROR GOTO <line number>

Function: Defines the starting line of the error handling routine.

**ON GOSUB** (statement, 1)

Format: ON <exprN> GOSUB <line number> [, <line number> ...]

Function: Executes the subroutine that starts in the <line number> according to <exprN>.

**ON GOTO** (statement, 1)

Format: ON <exprN> GOTO <line number> [, <line number> ...]

Function: Jump to the line <line number> according to <exprN>.

**ON INTERVAL GOSUB** (statement, 1)

Format: ON INTERVAL = <time> GOSUB <line number>

Function: Defines the interval and the line number for time interruption. <time> is defined in 1/60 second units on a NTSC machines and in 1/50 seconds in PAL machines.

**ON KEY GOSUB** (statement, 1)

Format: ON KEY GOSUB <line number> [, <line number> ...]

Function: Defines the line numbers for interrupting function keys.

**ON SPRITE GOSUB** (statement, 1)

Format: ON SPRITE GOSUB <line number>

Function: Defines the line number for sprite collision interruption.

**ON STOP GOSUB** (statement, 1)

Format: ON STOP GOSUB <line number>

Function: Defines the line number for interruption by pressing the CTRL+STOP keys.

**ON STRIG GOSUB** (statement, 1)

Format: ON STRIG GOSUB <line number> [, <line number> ...]

Function: Defines the line numbers for interruption by pressing the joystick trigger buttons.

**OPEN** (declaration, 1-D)

Format: OPEN <filename> [FOR {INPUT | OUTPUT}]  
AS#<file number> [LEN = <record size>]

Function: Open a file on tape or disk.

**OR** (logical operator, 1)

Format: <exprA1> OR <exprA2>

Function: Performs logical OR operation between <exprA1> and <exprA2>.

$$\begin{array}{ll} 0 \text{ or } 0 \rightarrow 1 & 1 \text{ or } 0 \rightarrow 0 \\ 0 \text{ or } 1 \rightarrow 0 & 1 \text{ or } 1 \rightarrow 1 \end{array}$$

**OUT** (statement, 1)

Format OUT <port number>, <exprN>

Function: Writes the value of <exprN> to an I/O port of the Z80.

**PAD** (function, 1-2)

Format: X = PAD (<exprN>)

Function: Examines the state of the mouse, trackball, lightpen or digitizer tablet and returns the value obtained in X.  
<exprN> can be:

- 0 – Check touch pad on port 1 (255 if connected)
- 1 – Returns the X coordinate (horizontal).
- 2 – Returns the Y (vertical) coordinate.
- 3 – Returns the key state (255 if pressed).
- 4 – Check touch pad on port 2 (255 if connected).
- 5 – Returns the X (horizontal) coordinate.
- 6 – Returns the Y (vertical) coordinate.
- 7 – Returns the key state (255 if pressed).
- 8 – Check lightpen (255 if connected or touching the screen).
- 9 – Returns the X (horizontal) coordinate.
- 10 – Returns the Y (vertical) coordinate.
- 11 – Returns the key state (255 if pressed).
- 12 – Check mouse on port 1 (255 if connected).
- 13 – Returns X coordinate offset (horizontal).
- 14 – Returns Y coordinate offset (vertical).
- 15 – Always 0.
- 16 – Check mouse on port 2 (255 if connected).
- 17 – Returns X coordinate offset (horizontal).

18 – Returns Y coordinate offset (vertical).

19 – Always 0.

20 – Checks 2nd lightpen (255 if connected or touching the screen). \* obs

21 – Returns the X (horizontal) coordinate. \* obs

22 – Returns the Y (vertical) coordinate. \* obs

23 – Returns the key state (255 if pressed). \* obs

Note: Values 20 to 23 require the use of the CALL ADJUST statement beforehand. Only available for MSX2 manufactured by Daewoo.

**PAINT** (statement, 1-2)

Format: PAINT {(X, Y) | STEP (X, Y)} [, <color> [, <border color>]]

Function: Fills the area enclosed by a line with the color <border color> with the color <color>.

**PDL** (function, 1)

Format: X = PDL (<paddle number>)

Function: Returns in X the state of the specified paddle. The paddle number can be:

1, 3, 5, 7, 9, 11 – Paddles connected to port 1.

2, 4, 6, 8, 10, 12 – Paddles connected to port 2.

**PEEK** (function, 1)

Format: X = PEEK (<address>)

Function: Returns in X the value of the byte contained in <address>.

**PLAY** (macro statement, 1)

Format: PLAY <expr\$ 1> [, <expr\$ 2> [, <expr\$ 3>]]

Function: Plays the notes specified by <expr\$> on PSG. The valid commands for <expr\$> are as follows:

An~Gn Plays an encrypted note with duration n (1~64, default is 4).

Rn Pause of duration n (1~64, default is 4).

# or + Sustain

– Flat

. Duration increase by 50%

On Octave (default is 4)

|                    |   |
|--------------------|---|
| Ln                 | Duration of notes (1~64, default is 4)        |
| Tn                 | Time and quarters of note per minute (32~255) |
| Vn                 | Volume (0~15, default is 8)                   |
| Nn                 | Absolute grade (1~96)                         |
| Mn                 | Wrap period (1~65 535, default is 255)        |
| Sn                 | Waveform (0~15, default is 0)                 |
| X <sub>serie</sub> | Executes macro in serie.                      |

Ex. A\$ = "ABC#" → PLAY "XAS"

= <variable> – Place a parameter as an integer after the command. Ex. A\$ = A\$ = "ABC#", S = 10,  
→ PLAY "XA\$; R = S; V = S"

**PLAY** (function, 1)

Format: X = PLAY (<n>)

Function: Returns in X the state of the voice <n> (playing = -1; not playing = 0).

**PLAY#** (macro declaration, M-4)

Function: Play the notes specified by <expr\$> on PSG, OPLL, MSX-Audio or MSX MIDI.

The  $\langle n \rangle$  value can be:

0 Play only PSG (same as PLAY)

- 1 Play through the MIDI interface.

2 or 3 Play through PSG and OPLL / MSX-Audio, depending on the chip activated with CALL MUSIC (OPLL) or CALL AUDIO (MSX-Audio).

→ The commands valid for <expr\$> are the same as for the PLAY statement, plus those described below for OPLL

(Note:  $M_n$  and  $S_n$  are exclusive to PSG):

Qn Sound width division (1~8, default is 8)

- > Increase an octave

< Decrease one octave

= x; Sets parameters to x

& Ligature

**{ } n** Define the notes between { } in n. (n=1~8, default is Ln)

@n Change the instrument (1~64)

→ For battery parts, the commands are as follows:

B Bass Drum

S Snare Drum

W Tom Tom

C Cymbals

H Hi hat

@Vn Detailed volume change (0~127)

@Nn Maintains the duration defined by n (1~64, default Ln)

n The “n”th note is paused (1~64)

! Accentuates the previous note

@An Sets the volume for accented voices (0~15)

→ Obs.: Tn, Vn, @Vn, Rn, X, = x; and . are identical to the other instruments.

**POINT** (function, 1)

Format: X = POINT (X, Y)

Function: Returns in X the color code of the point (X, Y) of the graphic screen.

**POKE** (statement, 1)

Format: POKE <address>, <data>

Function: Writes a byte of data to the memory <address>.  
<data> must be a numeric value between 0 and 255.

**POS** (system variable, 1)

Format: X = POS (0)

Function: Stores the horizontal position of the cursor in text mode.

**PRESET** (statement, 1-2)

Format: PRESET {(X, Y) | STEP (X, Y)} [, <color> [, <logical oper>]]

Function: Turns off the point specified by (X, Y) on the graphic screen. “STEP”, if specified, defines the offset.

**PRINT** (declaration, 1)

Format: PRINT [<exprA> [{; | ,} <exprA> ...]]

Function: Displays the characters corresponding to the expressions <exprA> on the screen. “;” does not generate linefeed and “,” advances to the next tab position.

**PRINT#** (declaration, 1-D)

Format: PRINT# <file number>, [<exprA> [{; | ,} <exprA> ...]]

Function: Writes the value of <exprA> to the specified file. ";" does not generate linefeed and "," advances to the next tab position.

**PRINT USING** (statement, 1)

Format: PRINT USING <"format">; <exprN> [{; | ,} <exprN> ...]  
PRINT USING <"expr\$ format">

Function: Displays on the screen the characters corresponding to the expressions <exprN> or <expr\$>, formatting. ";" does not generate linefeed and "," advances to the next tab position. The formatting characters are described below:

→ Numeric formatting:

- # Space for one digit
- . Includes decimal point
- + Indicates + or -; used before or after the number
- Indicates -; used after the number
- \$\$ Put "\$" to the left of the number
- \*\* Replaces leading spaces with asterisks
- \*\*\$ Places a "\$" to the left preceded by asterisks
- ^^^^ Displays the number in scientific notation

→ Alphanumeric formatting:

- \ \ Character space
- ! Space for a character & Variable spacing
- \_ Next character will print normally
- other Print character

**PRINT# USING** (declaration, 1-D)

Format: PRINT# <file number> USING <"format">; <exprA> [{; | ,} <exprA>...]

Function: Writes the value of <exprA> to the specified file, formatting. The formatting characters are the same as those for PRINT USING.

**PSET** (statement, 1)

Format: PSET {(X, Y) | STEP (X, Y)} [, <color> [, <logical operation>]]

Function: Draws the point specified by (X, Y) on the graphic screen. "STEP", if specified, defines the offset.

**PUT** (statement, D)

Format: PUT [#]<file number> [,<record number>]

Function: Writes a record to a random file.

**PUT HAN** (declaration, Daewoo CPC 400 / 400S)

Format: PUT HAN [(X, Y)],<Hangul code> [,<color> [,<logical  
operation> [,<mode>]]]

Function: Displays a Korean Hangul character on the screen.

<mode> sets the size of the Hangul character:

0 – 16x16 points

1 – 16x8 points (shows only odd lines)

2 – 16x8 points (shows only even lines)

**PUT KANJI** (statement, 1-2-Kanji)

Format: PUT KANJI [(X, Y)],<JIS code> [,<color> [,<logical  
operation> [,<mode>]]]

Function: Displays a Kanji character on the screen. <JIS code> may vary from &H2120 to &H4F53 for JIS1 and from &H5020 to &H7424 for JIS2.

<mode> defines the size of the Kanji:

0 – 16x16 points

1 – 16x8 points (shows only odd lines)

2 – 16x8 points (shows only even lines)

**PUT SPRITE** (statement, 1-2)

Format: PUT SPRITE <sprite plane> [, {(X, Y) | STEP (X, Y)} [,<color>  
[,<pattern number>]]]

Function: Displays a sprite on the screen. “STEP”, if specified, sets the offset. <sprite plan> is a number from 0 to 31 and specifies the display priority. Larger numbers will be displayed over smaller numbers. <pattern number> defines the pattern to be displayed. It can vary from 0 to 255 for 8x8 sprites and from 0 to 63 for 16x16 sprites. If not specified, it will be the same as <sprite plan>.

**READ** (statement, 1)

Format: READ <variable name> [,<variable name> ...]

Function: Reads data from the DATA command and stores it in <variable name>'s.



**REM** (statement, 1)

Format: REM <comments>

Function: Put comments (remarks) in the program.

**RENUM** (command, 1)

Format: RENUM [<new line number> [,<old line number>  
[,<increment>]]]

Function: Renumber program lines.

**RESTORE** (declaration, 1)

Format: RESTORE [<line number>]

Function: Specifies the initial DATA line number to be read by READ.

**RESUME** (statement, 1)

Format: RESUME {[0] | NEXT | <line number>}

Function: Ends error handling routine.

0 – Execution returns to the same command where the error occurred.

NEXT – Execution continues on the command following the one from which the error occurred.

<line number> – Execution will continue on the specified line.

**RETURN** (statement, 1)

Format: RETURN [<line number>]

Function: Returns from a subroutine.

**RIGHT\$** (function, 1)

Format: X\$ = RIGHT\$ (<expr\$>, <exprN>)

Function: Returns the <exprN> right <expr\$> characters in X\$.

**RND** (function, 1)

Format: X = RND [(<exprN>)]

Function: Returns in X a random number between 0 and 1. It is advisable to use “-TIME” in <exprN> to obtain better randomness.

**RSET** (declaration, D)

Format: RSET <string variable> = <expr\$>

Function: Stores the content of <expr\$> on the right of the string variable defined by the FIELD declaration.

**RUN** (command, 1-D)

Format: RUN [{<line number> | "filename"]

Function: Run a BASIC program in memory or load a program from disk and execute it. If <line number> is specified, execution will start on that line.

**SAVE** (command, 1-D)

Format: SAVE "<filename>" [,A]

Function: Saves the BASIC program to disk or tape. If "A" is specified, save in ASCII form and not in tokenized form.

**SCREEN** (statement, 1-2-3)

Format: SCREEN <screen mode> [,<sprite size> [,<key click> [,<cassette rate> [,<printer type> [,<interlace>]]]]]

Function: Selects screen mode and other values.

<screen mode> – 0 to 12, depending on the MSX version.

"Screen 9" only works on Korean computers or those loaded with Hangul BASIC.

<sprite size> – 0 → 8x8 sprites (default)

1 → 8x8 sprites expanded to 16x16

2 → 16x16 sprites

3 → 16x16 sprites enlarged to 32x32

<click keys> – 0 → turn off "click"

1 → turn on "click" (default)

<cassette rate> – 1 → write at 1200 baud (default)

2 → write at 2400 baud

<printer type> – 0 → MSX printer (default)

1 → non-MSX printer

<interlace> – 0 → normal (default)

1 → interlaced (Screen 0 standard)

2 → normal (alternate display)

3 → interlaced (alternate display)

**SET ADJUST** (statement, 2)

Format: SET ADJUST (<X coordinate>, <Y coordinate>)

Function: Changes the location of the screen. X and Y can vary from -7 to 8.

**SET BEEP** (declaration, 2)

Format: SET BEEP <timbre>, <volume>

Function: Selects the beep type and volume. <timbre> can vary from 1 to 4 and <volume> from 0 to 15.

**SET DATE** (declaration, 2)

Format: SET DATE <expr\$> [,A]

Function: Changes the date of the clock. [,A] changes the alarm date. <expr\$> must contain a valid date specification.

**SET HAN** (statement, Hangul-BASIC 2nd version)

Format: SET HAN [<size>], [<screen>], [<printer>]

Function: Defines how the Hangul characters will be displayed in Screen 0 to 8 and on the printer.

<size> – 0 → 8x8 point characters

1 → 8x16 point characters

<screen> – 0 → ungrouped characters

1 → characters grouped in blocks

<printer> – 0 → ungrouped characters

1 → characters grouped in blocks

**SET PAGE** (declaration, 2)

Format: SET PAGE <displayed page>, <active page>

Function: Select video pages. <displayed page> is the page that is being displayed on the screen and <active page> is the page on which the commands will be executed.

**SET PASSWORD** (declaration, 2)

Format: SET PASSWORD <expr\$>

Function: Activates the password. <expr\$> must contain a password with a maximum of 6 characters.

**SET PROMPT** (declaration, 2)

Format: SET PROMPT <expr\$>

Function: Activates a new prompt for BASIC. <expr\$> must contain the new prompt with a maximum of 6 characters.

**SET SYSTEM** (statement, Daewoo CPC300 / 400 / 400S)

Format: SET SYSTEM (<mode>) [CPC 300]

Function: Defines how the computer boots.

<mode> – 0 → starts the additional software in ROM  
 1 → starts BASIC

Format: SET SYSTEM [<dummy>], [<screen>], [<printer>]  
 [CPC 400 / 400S]

Function: Defines the initial parameters for the Hangul system.

<dummy> – Null action for any specified value  
 <screen> – 0 → ungrouped characters  
 1 → characters grouped in blocks  
 <printer> – 0 → ungrouped characters  
 1 → characters grouped in blocks

**SET SCREEN** (statement, 2)

Format: SET SCREEN

Function: Writes the data defined in the SCREEN statement to the SRAM of the clock.

**SET TIME** (statement, 2)

Format: SET TIME <expr\$> [,A]

Function: Changes the clock time. [,A] changes the alarm time.  
 <expr\$> must contain a valid time specification.

**SET TITLE** (declaration, 2)

Format: SET TITLE <expr\$> [,<title color>]

Function: Defines the title and color of the home screen. <expr\$>  
 must contain the title with a maximum of 6 characters.  
 <title color> can vary from 1 to 4

**SET VIDEO** (declaration, 2, optional)

Format: SET VIDEO [<mode> [,<Ym> [,<CB> [,<sync> [,<audio>  
 [,<video output> [,<AV control>]]]]]]]

Function: Defines superimposition and other modes.

<mode> can range from 0 to 3:  
 0 – Internal synchronization (default value)  
 1 – Digitization (external synchronization)  
 2 – Superimpose (external synchronization)  
 3 – External video (external synchronization)

<Ym> (external luminance):      0 = normal;    1 = halftone  
 <CB> (color bus):                0 = Input;    1 = Output  
 <sync> (synchronization mode): 0 = Internal;   1 = external  
     <audio> – Select the audio source:  
         0 – Computer only  
         1 – Computer + external right channel  
         2 – Computer + external left channel  
         3 – Computer + external right and left channels  
     <video out> – Select the video out mode:  
         0 – RGB;                      1 – Composite video  
     <AV control> – Selects the RGB output for audio and video.  
         0 – Not selected;          1 – Selected.

**SGN**            (function, 1)

Format: X = SGN (<exprN>)

Function: Returns the result of the <exprN> sign in X.

–1 → Negative expression

0 → The results of the expression is zero

1 → Positive expression

**SIN**            (function, 1)

Format: X = SIN (<exprN>)

Function: Returns in X the sine value of <exprN> (exprN must be expressed in radians).

**SOUND**        (statement, 1)

Format: SOUND <register number>, <data>

Function: Writes the value of <data> to the PSG register. <register number> can range from 0 to 13 and <data> from 0 to 255.

**SPACE\$**        (function, 1)

Format: X\$ = SPACE\$ (<exprN>)

Function: Returns a string with <exprN> spaces in X\$.

**SPC**            (function, 1)

Format: PRINT SPC (<exprN>)

Function: Prints <exprN> spaces.

**SPRITE** (statement, 1)

Format: **SPRITE** {ON | OFF | STOP}

Function: Enables, disables or suspends interruption due to sprite collision.

**SPRITE\$** (statement or system variable, 1)

Format: **SPRITE\$** (<sprite number>) = <expr\$>

**X\$** = **SPRITE\$** (<sprite\$>)

Function: Sets or reads the sprites pattern.

**SQR** (function, 1)

Format: **X** = **SQR** (<exprN>)

Function: Returns in **X** the square root value of <exprN>.

**STICK** (function, 1)

Format: **X** = **STICK** (<joystick port number>)

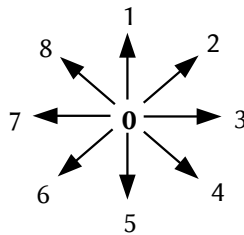
Function: Examines the direction of the joystick and loads the result in **X**.

<joystick port number> - 0 → Keyboard

1 → port #1

2 → port #2

The “X” value is illustrated below, according direction:



**STOP** (statement, 1)

Format: **STOP**

Function: It paralyzes the execution of a program.

Format: **STOP** {ON | OFF | STOP}

Function: Enables, disables or supposes interruption by pressing the CTRL+STOP keys.

**STRIG** (function / declaration, 1)

Format: **X** = **STRIG** (<joystick port number>)

**Function:** Examines the state of the trigger buttons and returns the result in X. The value will be -1 if it is being pressed or 0 otherwise.

<joystick port number> can be:

0 = Space bar

1 = joystick on port 1, button A

2 = joystick on port 2, button A

3 = joystick on port 1, button B

4 = joystick on port 2, button B

**Format:** STRIG (<joystick port number>) {ON | OFF | STOP}

**Function:** Enables, disables or suspend interruption by pressing the trigger buttons.

**STR\$** (function, 1)

**Format:** X\$ = STR\$ (<exprN>)

**Function:** Converts the value of <exprN> to a decimal string and returns the value obtained in X\$.

**STRING\$** (statement, 1)

**Format:** X\$ = STRING\$ (<exprN1>, {<expr\$> | <exprN2>})

**Function:** Returns in X\$ a string of length <exprN1>, where all characters are equal, formed by the first character of <expr\$> or by the character whose ASCII code is represented by <exprN2>.

**SWAP** (declaration, 1)

**Format:** SWAP <variable name>, <variable name>

**Function:** Exchanges the content of two variables. The variables must be of the same type.

**TAB** (function, 1)

**Format:** PRINT TAB (<exprN>)

**Function:** Produces <exprN> spaces for the PRINT instructions.

**TAN** (function, 1)

**Format:** X = TAN (<exprN>)

**Function:** Returns in X the tangent value of <exprN> (exprN must be expressed in radians).

**TIME** (system variable, 1)

Format: X = TIME

Function: Returns the value of TIME (it is an integer variable that is continuously incremented 60 times per second in NTSC machines and 50 times in PAL machines).

Format: TIME = <exprN>

Function: Assigns the value of <exprN> to the TIME variable. It must be an integer value.

**TROFF** (command, 1)

Format: TROFF

Function: Turn off the line tracking of the running program.

**TRON** (command, 1)

Format: TRON

Function: Turn on the line tracking of the running program.

**USR** (function, 1)

Format: X = USR [<number>] (<argument>)

Function: Executes an assembly routine. <number> can be a value from 0 to 9.

**VAL** (function, 1)

Format: X = VAL (<expr\$>)

Function: Converts <expr\$> to a numeric value and stores it in X.

**VARPTR** (function, 1-D)

Format: X = VARPTR (<variable name> | #<file number>)

Function: Returns in X the address where the <variable> is stored or the FCB address of <file number>

**VDP** (system variable, 1-2-3)

Format: X = VDP (<register number>)

VDP (<register number>) = <data>

Function: Read or write data in a VDP register. <data> must be a numeric value between 0 and 255.

**VPEEK** (function, 1-2)

Format: X = VPEEK (<address>)

Function: Returns in X the content of the VRAM byte specified by <address>.



**VPOKE** (statement, 1-2)

Format: VPOKE <address>, <data>

Function: Writes a data byte in the <address> of the VRAM. <data> must be a numeric value between 0 and 255.

**WAIT** (statement, 1)

Format: WAIT <port number>, <exprN1> [, <exprN2>]

Function: Stops the execution of the program until the specified port value matches the value of <exprN1> or <exprN2>.

**WIDTH** (declaration, 1-2)

Format: WIDTH <number>

Function: Specifies the number of characters per line in text modes (Screen 0 and 1).

**XOR** (logical operator, 1)

Format: <exprA1> XOR <exprA2>

Function: Performs logical XOR operation between <exprA1> and <exprA2>.

|             |             |
|-------------|-------------|
| 0 xor 0 → 0 | 1 xor 0 → 1 |
| 0 xor 1 → 1 | 1 xor 1 → 0 |

### 3.3 – EXTENDED COMMANDS

The CALL command allows the MSX-BASIC instructions to be expanded indefinitely, allowing access to new devices in cartridges or new features. Below is described a majority of the instructions available through the CALL command.

DM-System2 BASIC

(Installable extension for BASIC)

|         |        |         |         |
|---------|--------|---------|---------|
| BGMOFF  | BLOCK  | COS     | FILES   |
| BGMON   | CALL   | DMM     | FSIZE   |
| BGMTMP  | CELLO  | DMMINI  | HELP    |
| BGMTRS  | CHGCPU | DMMOFF  | HMMM    |
| BGMVOL  | CHGDRV | DMMON   | HMMV    |
| BGMWAIT | CHGPLT | EXT     | INTWAIT |
| BINLOAD | COLOR= | EXTCOPY | KBOLD   |

|         |         |        |         |
|---------|---------|--------|---------|
| KCOLOR  | PACSAVE | SEOFF  | UPPER   |
| KINIT   | PAUSE   | SEON   | VCOPY   |
| KPRINT  | PCMON   | SETBIN | VDPWAIT |
| KPUT    | PEEK    | SETPLT | VMOFF   |
| KSIZE   | PEEKS   | SETSE  | VMON    |
| LMMM    | PEEKW   | SIN    | VMWAIT  |
| LMMV    | POKE    | STATUS | WAIT    |
| LOAD    | POKES   | SYSOFF | XY      |
| MALLOC  | POKEW   | SYSON  | YMMM    |
| PACLOAD | SAVE    | SYSTEM |         |

### FM-X BASIC

(Available with the Fujitsu MB22450 interface when inserted into the FM-X expansion slot)

CALL MON                      CALL PRINTERSETUP

### FormatMaster-BASIC

(Available with an installable extension that comes on the Future Magazine Extra disc. FORMAT.BIN – FORMAT.MEM – FORMAT.TXT files)

CALL FORMAT

### GR8NET BASIC

(Available with the installation of the GR8NET cartridge)

|           |           |             |            |
|-----------|-----------|-------------|------------|
| DSK       | FLUPDATE  | NETDUMP     | NETGETMAP  |
| DSKCFG    | NET       | NETEXPRT    | NETGETMASK |
| DSKFMT    | NETBITOV  | NETFIX      | NETGETMD   |
| DSKGETIMG | NETBLOAD  | NETFKOPLL   | NETGETMEM  |
| DSKLDIMG  | NETBROWSE | NETFWUPDATE | NETGETMIX  |
| DSKHELP   | NETBTOV   | NETGETCLK   | NETGETMMV  |
| DSKLDIMG  | NETCDTOF  | NETGETCLOUD | NETGETNAME |
| DSKSETIMG | NETCFG    | NETGETDA    | NETGETNTP  |
| DSKSVIMG  | NETCODE   | NETGETDNS   | NETGETOPL  |
| DSKSTATE  | NETDHCP   | NETGETGW    | NETGETPATH |
| FLINFO    | NETDIAG   | NETGETHOST  | NETGETPORT |
| FLLIST    | NETDNS    | NETGETIP    | NETGETPSG  |

|            |             |            |             |
|------------|-------------|------------|-------------|
| NETGETQSTR | NETRESST    | NETSETMEM  | NETSTAT     |
| NETGETTSHN | NETSAVE     | NETSETMIX  | NETSYSINFO  |
| NETGW      | NETSDCRD    | NETSETMMV  | NETRCHKS    |
| NETHELP    | NETSETCLK   | NETSETNAME | NETTELNET   |
| NETIMPRT   | NETSETCLOUD | NETSETNTP  | NETTERM     |
| NETIP      | NETSETDA    | NETSETOPL  | NETTGTMAP   |
| NETLDBUF   | NETSETDM    | NETSETPATH | NETTSYNC    |
| NETLDRAM   | NETSETDNS   | NETSETPORT | NETVARBRSTR |
| NETMASK    | NETSETGW    | NETSETPSG  | NETVARBSIZE |
| NETNTP     | NETSETHOST  | NETSETQSTR | NETVARRWTH  |
| NETPLAYBUF | NETSETIP    | NETSETTSHN | NETVARUDTO  |
| NETPLAYVID | NETSETMAP   | NETSNDDTG  | NETVER      |
| NETPLAYWAV | NETSETMASK  | NETSNDVOL  |             |

### Hangul-BASIC

(Disponível em micros coreanos. Veja também PUT HAN, SET HAN e SET SYSTEM)

|        |        |         |        |
|--------|--------|---------|--------|
| CLS    | HELP   | KLEN    | REBOOT |
| ENG    | KCHR   | KMID    | RTCINI |
| FONT   | KCODE  | KTYPE   | VER    |
| HANOFF | KEXT   | MODE9   |        |
| HANON  | KINSTR | PALETTE |        |

### Hitachi-BASIC

(Disponível em alguns micros Hitachi)

- A versão 1 está disponível no modelo MB-H1 (MSX1)
- A versão 2 está disponível no modelo MB-H2 (MSX1)
- A versão 3 está disponível no modelo MB-H3 (MSX2)

|          |         |       |        |
|----------|---------|-------|--------|
| AUTOMUTE | CMT     | MON   | REW    |
| BLSCAN   | CSCAN   | MUTE  | SCOPY  |
| CCOPY    | CSCOPY  | NSCAN | STDBY  |
| CDCOPY   | FF      | PAUSE | STOP   |
| CFILES   | HCOPY   | PLAY  | TABOFF |
| CHCOPY   | IDTRACE | REC   | TABON  |

**Kanji-BASIC**

(Disponível em micros japoneses MSX2+ ou superior. Veja também PUT KANJI)

|       |        |       |         |
|-------|--------|-------|---------|
| AKCNV | KACNV  | KLEN  | PALETTE |
| ANK   | KANJI  | KMID  | SJIS    |
| CLS   | KEXT   | KNJ   |         |
| JIS   | KINSTR | KTYPE |         |

**Mega Assembler**

(Disponível com a instalação do cartucho Mega-Assembler)

ASM                      START

**MSX-Aid BASIC**

(Disponível com a instalação do cartucho MSX-AID)

|        |           |          |
|--------|-----------|----------|
| CFILES | MESSAGE   | TRACE ON |
| FIND   | MON       | VARLIST  |
| HELP   | TRACE OFF | XREF     |

**MSX-Audio BASIC**

(Disponível com a instalação de cartuchos com o MSX-Audio BIOS)

|           |          |          |            |
|-----------|----------|----------|------------|
| APEEK     | COPY PCM | MK VOICE | RECMOD     |
| APOKE     | INMK     | MK VOL   | REC PCM    |
| APPEND MK | KEY OFF  | PCM FREQ | SAVE PCM   |
| AUDIO     | KEY ON   | PCM VOL  | SET PCM    |
| AUDREG    | LOAD PCM | PITCH    | STOPM      |
| BGM       | MK PCM   | PLAY     | TEMPER     |
| CONT MK   | MK STAT  | PLAY MK  | TRANPOSE   |
| CONVA     | MK TEMPO | PLAY PCM | VOICE      |
| CONVP     | MK VEL   | REC MK   | VOICE COPY |

**MSX-Music BASIC**

(Disponível através de cartuchos ou internamente)

|         |       |          |            |
|---------|-------|----------|------------|
| AUDREG  | MUSIC | STOPM    | VOICE      |
| BGM     | PITCH | TEMPER   | VOICE COPY |
| MDR (*) | PLAY  | TRANPOSE |            |

(\*) Disponível apenas no MSX turbo R FS-A1GT

**Network-BASIC**

(Extensão BASIC disponível apenas nos computadores MSX2 Yamaha YIS-503IIIR e YIS-805/128R2, usados em escolas da União Soviética)

|          |         |         |        |
|----------|---------|---------|--------|
| BRECEIVE | NETEND  | PON     | SNDRUN |
| BSEND    | NETINIT | RCVMAIL | STOP   |
| CHECK    | OFFLINE | RECEIVE | TALK   |
| DISCOM   | ONLINE  | RUN     | WHO    |
| ENACOM   | PEEK    | SEND    |        |
| HELP     | POFF    | SNDCMD  |        |
| MESSAGE  | POKE    | SNDMAIL |        |

**NewModem-BASIC**

(Disponível para os modems Philips NMS 1255 e Micro Technology MT-Telcom II)

|         |         |         |            |
|---------|---------|---------|------------|
| ANSWER  | FILEOUT | LOGFILE | RECFILE    |
| BREAK   | GET     | LS      | RTSOFF     |
| CARRIER | INIMDM  | MC      | RTSON      |
| CHKMDM  | INITMD  | MRING   | SENDFILE   |
| CONNECT | LINEOFF | MSTART  | SPEAKEROFF |
| DTROFF  | LINEON  | MSTOP   | SPEAKERON  |
| DTRON   | LB      | OFFHOOK | TDIAL      |
| ECHOOFF | LEN     | ONHOOK  | TERMINAL   |
| ECHOON  | LO      | PDIAL   |            |

**Nextor-BASIC**

(Disponível através de cartuchos IDE com Nextor)

|         |         |          |        |
|---------|---------|----------|--------|
| CURDRV  | DRVINFO | MAPDRV   | NEXTOR |
| DRIVERS | LOCKDRV | MAPDRV L | USR    |

**Pioneer-BASIC (P-BASIC)**

(Disponível nos micros da Pioneer PX-7, PX-V7 e PX-V60)

|             |           |        |        |
|-------------|-----------|--------|--------|
| BLIND       | FRAME     | MUTE   | SEARCH |
| CHAPTER     | FRAME OFF | PAN    | SYMBOL |
| CHAPTER OFF | IMPOSE    | REMOTE | VIDEO  |
| DEF UNIV    | LCOPY     | SCLOAD |        |
| EXTV        | LD        | SCSAVE |        |

**Printer-BASIC**

(Disponível nos micros da Toshiba HX20 até HX23F e HX31 até HX34)

LCOPY

SPOLOFF

SPOLON

**QuickDisk BASIC**

(Disponível nos micros Daewoo (versão 1.0) e Casio, Philips e Sanyo (versão 1.1))

BLOAD

LOAD

QDFORMAT

RUN

BSAVE

MERGE

QDKEY

SAVE

CASQD

QDFILES

QDKILL

**RMSX BASIC**

(Extensão que vem com o emulador RMSX para o MSX Turbo R)

?

CHCAS

FILES

LICENSE

CASAUTOREW

CHDIR

HELP

MUTE

CASREW

CHDSK

HZ

PALETTE

CASRUN

EXIT

IOSOUND

RESET

**RookieDrive BASIC**

(Disponível com a instalação do cartucho RookieDrive)

CREATEDISK

HELP

REBOOT

USBRESET

EJECT

INSERTDISK

USBCD

FNAME

LOADROM

USBERROR

FORMAT

MOUNT

USBFILES

**SFG-BASIC**

(Disponível com a instalação do cartucho Yamaha FM Music Macro)

CANCEL

PATTERN

STOP

CLDVOICE

PHRASE

SYNCOUT

ERASE

PLAY

TEMPO

EVENT ON/OFF/STOP

RCANCEL

TIMER

INIT

REPORT

TRACK

INMKEY

RHYTHM

TRANSPPOSE

INST

RSTOP

TSTOP

LENGTH

SELPATTERN

TUNE

LFO

SELVOICE

USERHYTHM

LOOK

SOUND

VLIST

MODINST

STANDBY

WAIT

ON EVENT...GOSUB

START

**StudioFM BASIC**

(Disponível com a instalação do StudioFM para tocar músicas .MUS geradas pelo FAC Soundtracker. Usar BLOAD"SFMDRV1.BIN",R )

|       |       |       |       |
|-------|-------|-------|-------|
| MFADE | MLOAD | MPLAY | MSTOP |
|-------|-------|-------|-------|

**SVI-Modem BASIC**

(Disponível apenas no modem Spectravideo SVI-737)

|             |         |         |       |
|-------------|---------|---------|-------|
| COMBREAK    | COMOFF  | COMTERM | TDIAL |
| COMDTR      | COMON   | OFFLINE |       |
| COM...GOSUB | COMSTAT | ONLINE  |       |
| COMINI      | COMSTOP | PDIAL   |       |

**X-BASIC**

(Disponível com a instalação do compilador em tempo real X-BASIC)

|     |          |                |
|-----|----------|----------------|
| '#C | CALL BC  | CALL TURBO ON  |
| '#I | CALL RUN | CALL TURBO OFF |
| '#N |          |                |

**3.3.1 – Commands Description**

? (statement, RMSX-BASIC)

Format: CALL ?

Function: Displays help for RMSX-BASIC. It is equivalent to the CALL HELP command.

**ADJUST** (command, Daewoo)

Format: CALL ADJUST

Function: Enables the internal lightpen interface. Available only for MSX2 manufactured by Daewoo.

**AKCNV** (statement, Kanji-BASIC)

Format: CALL AKCNV (<variable>, "<character string>")

Function: Converts single-byte characters to 2-byte Kanji.  
 <string variable> receives the converted characters.  
 <character string> are the ASCII characters to be converted.

**ANK** (statement, Kanji-BASIC)

Format: CALL ANK

Function: Exits Kanji mode (the memory used by the Kanji driver is not released).

**ANSWER** (function, New Modem BASIC)

Format: CALL ANSWER (<speed>)

Function: Detects the speed of a connection. This instruction works only in BBS programs. The detected speeds are:

| Value | Standard | Recep. speed | Transm. speed |
|-------|----------|--------------|---------------|
| 1     | V21      | 300 baud     | 300 baud      |
| 2     | V23      | 1200 baud    | 75 baud       |
| 3     | V23      | 75 baud      | 1200 baud     |

**APEEK** (function, MSX-Audio)

Format: CALL APEEK (<address>, X)

Function: Returns in X the value of the byte corresponding to the memory address of MSX-Audio. The address can range from 0000H to 7FFFH.

**APOKE** (statement, MSX-Audio)

Format: CALL APOKE (<address>, <data>)

Function: Writes a byte of data to the <address> of the audio memory. <data> must be a numeric value between 0 and 255. The address can range from 0000H to 7FFFH.

**APPEND MK** (statement, MSX-Audio)

Format: CALL APPEND MK (<matrix name>)

CALL APPEND MK (<start address>, <end address>)

CALL APPEND MK (A), where the sequence A must be previously declared in the DIM and REC MK instructions.

Function: Adds an additional recording played on the musical keyboard.

**ASM** (command, Mega Assembler)

Format: CALL ASM

Function: Calls the Mega Assembler without initializing the variables. To call the MA by initializing the variables, use CALL START.



**AUDIO** (statement, MSX-Audio)

Format: CALL AUDIO (<mode>, <channels with instruments>, <channels for string 1>, <channels for string 2>, ....., <channels for string 9>)

<mode> defines the use of MSX-Audio. The default is 1.

| Mode | FM melody | PCM     | FM rhythm | Type   |
|------|-----------|---------|-----------|--------|
| 0    | 9 voices  |         |           | Normal |
| 1    | 6 voices  |         | 3 voices  | Normal |
| 2    | 9 voices  | 1 voice |           | Normal |
| 3    | 6 voices  | 1 voice | 3 voices  | Normal |
| 4    | 9 voices  |         |           | CSM    |
| 5    | 6 voices  |         | 3 voices  | CSM    |
| 6    | 9 voices  | 1 voice |           | CSM    |
| 7    | 6 voices  | 1 voice | 3 voices  | CSM    |

In CSM mode, the control of all FM sounds (melody and rhythm) is invalid. CSM stands for Composite Sinusoidal Modeling. Using all operators in parallel, this mode can be used to synthesize speech.

<channels with instruments> defines how many channels will be assigned to an instrument.

<channels for string n> defines how many channels will be used for each string related to the FM melody in the PLAY instruction.

**AUDREG** (declaration, MSX-Music and MSX-Audio)

Format: CALL AUDREG <register>, <data> [, <channel>]

Function: Writes the value of <data> in the OPLL or MSX-Audio register. <channel> specifies the channel to be used (MSX-Audio only). It can be 0 or 1, the default is 0.

Note: Previous use of CALL MUSIC or CALL AUDIO is required.

**AUTOMUTE** (command, Hitachi-BASIC version 2)

Format: CALL AUTOMUTE

Function: Adds a 4 second pause before activating the internal data reader of some Hitachi computers.

**BGM** (declaration, MSX-Music and MSX-Audio)

Format: CALL BGM (n)

Function: Arrow executing commands while the music is being played. <n> can be 0 or 1, as below:

0 – No commands can be executed during the music.

1 – Commands can be executed during music (default).

**BGMOFF** (declaration, DM-System2 BASIC)

Format: CALL BGMOFF (<fade>)

Function: Mutes the music played by OPLL / MSX Music. Requires BGM driver.

<fade> – 1 → without fade (immediate stop)

2 → with fade out

**BGMON** (declaration, DM-System2 BASIC)

Format: CALL BGMON (<start address> [, <num of repetitions>])

Function: Plays music using the BGM driver. Requires BGM driver.

<start address> is a pointer to the BGM data in the Main RAM.

<repetition number> is the number of times the song will be played. "0" indicates infinite repetitions.

**BGMTMP** (declaration, DM-System2 BASIC)

Format: CALL BGMTMP (<time>)

Function: Adjusts the “tempo” of the song. Requires BGM driver.

<time> is a value between 0 and 255 representing the percentage. The default value is 100.

**BGMTRS** (declaration, DM-System2 BASIC)

Format: CALL BGMTRS (<transpose>)

Function: Adjusts the key of the music. Requires BGM driver.

<transpose> is a one-byte value between –128 and +127.

The default value is 0.

**BGMVOL** (declaration, DM-System2 BASIC)

Format: CALL BGMVOL ([<Master>] [, <OPLL>] [, <PSG>] [, <SCC>])

Function: Adjusts the volume of the various sound generators individually. It can vary from 0 to 15 for each one and the default value for all is 15. Requires BGM driver.

**BGMWAIT** (declaration, DM-System2 BASIC)

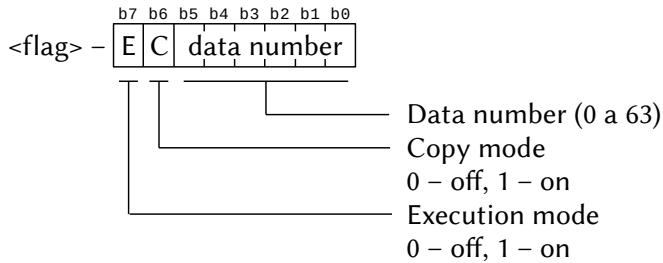
Format: CALL BGMWAIT

Function: Pause or restart the BGM. Requires BGM driver.

**BINLOAD** (command, DM-System2 BASIC)

Format: CALL BINLOAD (&lt;flag&gt; [,&lt;flag 2&gt;] [,&lt;destination address&gt;] [,&lt;size&gt;])

Function: Transfers concatenated data from the binary table in VRAM to RAM, being able to execute it.



&lt;flag 2&gt; - Value of a byte that replaces the same flag in the binary table. By default, the table flag is used.

&lt;destination address&gt; - 2-byte value that specifies the initial destination address for the data.

&lt;size&gt; - 2-byte value that specifies the number of bytes to be transferred.

Obs. - &lt;destination address&gt; and &lt;size&gt; must be omitted when the data format has the same format as the COPY instruction (the first byte is the X coordinate and the second is the Y coordinate).

**BLIND** (statement, Pioneer-BASIC)

Format: CALL BLIND ([&lt;string&gt;], [S | L])

Function: Delete or re-enable the display of the screen. Works only on Screen 2.

&lt;string&gt; can range from 0 to 9 and specifies the sequence in which the screen will be cleared or enabled.

S - Saves the screen while it is being erased

L - Loads the screen previously saved by "S".

**BLOAD** (command, QuickDisk BASIC)

Format: CALL BLOAD ("[QD [n]:]<filename>" {[,R] | [,S]} [, offset])

Function: Load the binary code from the QuickDisk device.

QD [n] specifies the QuickDisk device to be used. It can range from 0 to 7, the default being 0.

<filename> must be in the format 8.3 characters.

“, R” automatically executes the binary code of the loaded file

“, S” uploads the content to VRAM.

<displacement> indicates that the program will be loaded at the start address + offset. This parameter also affects the execution address.

**BLOCK** (command, DM-System2 BASIC)

Format: CALL BLOCK ([@]<source address>, [@]<destination address>, <size>)

Function: Copy data between Main RAM and VRAM. If the <address> is preceded by “@”, VRAM will be specified. To avoid error messages, decimal numbers should be used for addresses greater than FFFFH (65 535).

**BLSCAN** (command, Hitachi-BASIC version 2)

Format: CALL BLSCAN

Function: Makes the internal data reader of the Hitachi micro MB-H2 searches for files recorded with BSAVE and that can be loaded with BLOAD.

**BREAK** (command, New Modem BASIC)

Format: CALL BREAK

Function: Assign the interrupt call to the CODE key. This will allow you to interrupt the RING and DIAL routines just by pressing the CODE key.

**BRECEIVE** (command, Network-BASIC)

Format: CALL BRECEIVE ([[<unit name>:]<filename>] [,<student number>] [,<start address>] [,<end address>] [,S])

**Function:** Receives binary data in RAM or VRAM from (other) students' computers. This instruction can be used by the teacher and students who have been authorized by the teacher through CALL ENACOM. The short version \_BREC can be used. <unit name> can be "A:" or "B:."; <student number> can range from 0 to 15; <start address> and <end address> can range from &H0000 to &HFFFF and "S" specifies VRAM.

**BSAVE** (command, QuickDisk BASIC)

**Format:** CALL BSAVE ("[QD [n]:]<filename>",<start address>,<end address> [,<run address>])  
CALL BSAVE ("[QD [n]:]<filename>",<start address>,<end address>, S)

**Function:** Saves a memory area on the specified QuickDisk device. QD [n] specifies the QuickDisk device to be used. It can range from 0 to 7, the default being 0. <filename> must be in the format 8.3 characters. <start address>, <end address> and <run address> can vary from &H0000 to &HFFFF. If <run address> is omitted, <start address> will be used instead. "S" is used to save VRAM content.

**BSEND** (command, Network-BASIC)

**Format:** CALL BSEND ([<unit name>:]<filename>] [,<student number>] [,<starting address>] [,<end address>] [, S])

**Function:** Sends binary data from RAM or VRAM from (other) students' computers. See BRECEIVE for more information.

**CALL** (declaration, DM-System2 BASIC)

**Format:** CALL CALL (<address> [,<AF>] [,<HL>] [,<DE>] [,<BC>] [,<IX>] [,<IY>])

**Function:** Calls a machine language routine in Main-RAM, unless the address is less than 2000H (below that it will be called Main-ROM to allow access to BIOS routines). If <AF> is less than 256, the value will be loaded into register A.

**CANCEL** (declaration, SFG-BASIC)

Format: CALL CANCEL (<instrument number>)

Function: Cancels an instrument. <instrument number> can vary from 1 to 4. Short version: \_CANC.

**CARRIER** (statement, New Modem BASIC)

Format: CALL CARRIER (: GOSUB <line number>)

Function: Specifies the GOSUB routine to be performed when the operator is absent for an unknown reason or because the caller has just hung up. This instruction is only useful in BBS programs.

**CASAUTOREW** (command, RMSX-BASIC)

Format: CALL CASAUTOREW [ON] | [OFF]

Function: Enables or disables the automatic rewinding of a tape image (CAS file) back to the beginning. Without a parameter, this instruction switches between the two options.

ON – Enables automatic rewinding of the tape image.

OFF – Disables automatic rewinding

**CASQD** (command, QuickDisk BASIC)

Format: CALL CASQD ["[CAS:]" <filename1>"] [, "[QD [n]:"  
[ " <filename2>"] )]

Function: Transfer the specified file from cassette to QuickDisk.

Without parameters, this command transfers the tape file to the standard QuickDisk device with the same name.

QD [n] specifies the QuickDisk device to be used. It can range from 0 to 7, the default being 0.

<filename1> is the name of the file to be copied from the tape.

<filename2> is the name of the file to be written to the Quick Disk. The format is limited to 6 characters with no extension. If <filename2> is omitted, <filename1> is repeated.

**CASREW** (command, RMSX-BASIC)

Format: CALL CASREW

Function: Manually rewind a tape image (CAS file) back to the beginning.

**CASRUN** (command, RMSX-BASIC)

Format: CALL CASRUN ["[<drive letter>:] [<path> \]  
<filename.CAS>"]

Function: Load and execute files contained in the specified tape image (CAS file). If <filename.CAS> is omitted, the first file in the image inserted with CALL CHCAS will be executed.  
<Drive letter>: can go from A: to H:.

**CCOPY** (command, Hitachi-BASIC version 3)

Format: CALL CCOPY

Function: Sends to the printer a darker copy of a graphic screen in Screens 2, 4 or 5 simulating shades of gray.

**CDCOPY** (command, Hitachi-BASIC version 3)

Format: CALL CDCOPY

**Function:** Sends to the printer a copy of a graphical screen in Screens 2, 4 or 5 using only white and black dots.

**CELLO** (declaration, DM-System2 BASIC)

Format: CALL CELLO (<X0>, <Y0>) – [STEP] (<X1>, <Y1>) [,<n0>]  
[,<n1>] [,<n2>]

Function: Changes the colors of a specified rectangular area of the screen. Does not work on Screens 0 to 4.  
Screens 5, 6, 7: Replaces one color with another  
Screen 8: Modify colors according to RGB  
Screen 10, 11, 12: Modify the Y values of the colors  
<X0>, <Y0> coords of the starting corner of the rectangle  
<X1>, <Y1> coordinates of the final corner of the rectangle  
<n0> color to replace in Screens 5 to 7 (0~15);  
          value to add to the red in Scr 8 (-7~+ 7);  
          value to add Y to Scr 10 to 12 (-31~+ 31).  
<n1> new color for Screens 5 to 7 (0~15);  
          value to add to green on Screen 8 (-7~+ 7).  
<n2> value to add to the blue on Screen 8 (-7~+ 7).

**CFILES** (command, Hitachi-BASIC, MSX Aid BASIC)

Format: CALL CFILES [Hitachi-BASIC 2]

Function: Lists the contents of the tape inserted in the internal data reader of some Hitachi computers. It is recommended to rewind the tape with the CALL REW command first.

Format: CALL CFILES [MSX Aid BASIC]

Function: Lists the contents of the tape inserted in the data reader connected to the MSX. The list specifies whether a file is binary (OBJ), BASIC (BAS) or ASCII (ASC). It also returns the size of the binary files. It is recommended to rewind the tape first.

## **CHAPTER** (statement, Pioneer-BASIC)

Format: CALL CHAPTER (<chapter num>, GOSUB <line num>)  
CALL CHAPTER OFF

Function: Specifies the subroutine line number that will be executed when the chapter <chapter number> is reached. <chapter number> should be in the range between 50 and 54,000. If "OFF" is specified, cancel the line number assignment. This command is specific for use with the Pioneer Laser Vision Player LD-700 and cannot be used in conjunction with the FRAME command.

## **CHCAS** (control, RMSX-BASIC)

Format: CALL CHCAS (" [<drive letter>:] [\<path>] <filename.CAS>")

Function: Assembles (inserts) the specified tape image (CAS file) in the virtual cassette player of the MSX1 or MSX2 computer emulated on a Turbo R with the rMSX emulator.  
<drive letter>: can range from A: to H :.

## **CHCOPY** (command, Hitachi-BASIC version 3)

Format: CALL CHCOPY

Function: Sends to the printer a lighter copy of a graphic screen in Screens 2, 4 or 5 simulating shades of gray.

## **CHDIR** (declaration, Disk-BASIC 2nd version, RMSX-BASIC)

Format: CALL CHDIR ([<drive letter>:] [\<path>) [Disk-BASIC 2]

Function: Change subdirectory. The argument can also be just "." or "" to return directories.

Format: CALL CHDIR ([<drive letter>:] [\<path>) [RMSX-BASIC]

Function: Change the current working directory of an actual disk on an MSX Turbo R drive, used as a host for an MSX1/MSX2 computer emulated on this machine with the rMSX emulator. The argument can also be just "." or "" to return directories.



**CHDRV** (command, Disk-BASIC 2nd version)

Format: CALL CHDRV (<drive letter> :)

Function: Change the drive according to “<drive letter>”. If Nextor is installed, the argument can be replaced with a number (1 = A:, 2 = B:, etc.); otherwise, you must indicate the drive letter (A: up to H:).

## CHKDSK (command, RMSX-BASIC)

Format: CALL CHKDSK (see parameters below)

Function: Mount (insert) the specified disk image (DSK file) on the virtual disk drive of the MSX1 / MSX2 computer emulated on a Turbo R with the rMSX emulator and / or activate a specified disk number (the Turbo disk drive R can also be used with a real disc).

→ To mount the disk image to the current disk number (if the parameter is not provided or is empty, the actual disk unit will be used)

**CALL CHDSK** [(("[<drive letter>:] [\<path>] <filename.DSK>"))]

→ To mount the disk image on the specified disk number  
(without activation)

```
CALL CHDSK ("[<drive letter>:] [\<path>\]  
          <filename.DSK> ").<disk number>
```

→ To activate the specified disc number and eventually mount the disc image on it

CALL CHDSK (<drive letter>), [("[<device name>:  
[\\<path>\\]<filename.DSK>")]

**CHECK** (command, Network-BASIC)

Format: CALL CHECK ([<connection variable>] [,<communication variable>])

Function: Checks which students are connected to the network and / or which students are able to communicate with others. This instruction is only available to the teacher. <connection variable> contains the binary representation of connected / unconnected students. <communication variable> contains the binary representation of students with extended communication enabled or disabled. Both are 16-bit integer variables, where bit 0 is associated with

student 1, bit 1 is associated with student 2 and so on, up to bit 14. “0” means connected or active student and “1” means disconnected or inactive student.

**CHGCPU** (command, DM-System2 BASIC)

Format: CALL CHGCPU ([<mode>] [,<variable>])

Function: Switch or return CPU mode on MSX turbo R.

<mode> – Mode to be applied

<variable> – Value before being changed to <mode>

The two parameters have the following format:

**CHGDRV** (command, DM-System2 BASIC)

Format: CALL CHGDRV ([<drive number>] [,<variable>])

Function: Change or return the current drive unit.

<drive number> – Must be a number between 1 and 8,  
where 1 = A :, 2 = B :, etc.

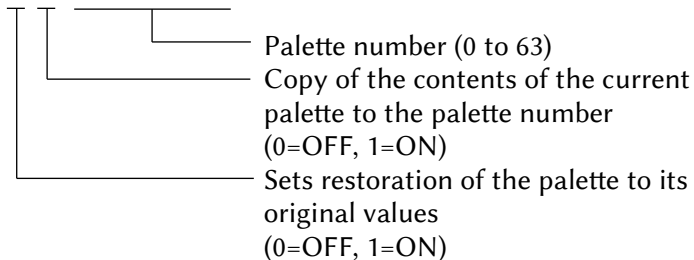
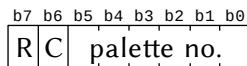
<variable> – Numeric variable that will receive the file size.

**CHGPLT** (declaration, DM-System2 BASIC)

Format: CALL CHGPLT (<number>)

Function: Change the colors of the palette. The palette data must be previously placed in RAM.

<number> is a one-byte value with the following format:



**CKHMDM** (function, New Modem BASIC)

Format: CALL CKHMDM (<numeric variable>)

Function: Checks whether the modem is present. If <numeric variable> is 0, the modem has been detected, otherwise there is no modem.

**CLDVOICE** (command, SFG-BASIC)

Format: CALL CLDVOICE [((<display name>), (<device>))]

Function: Loads the voice data into memory, from the cassette or the cartridge memory. <display name> specifies whether to display the voice name during loading. If it is "0" the name will not be displayed and if it is "1" it will be displayed. <device> can be: 0 – Cassette; 1 – Cartridge.

Short version: CLDV.

**CLS** (declaration, Kanji-BASIC and Hangul-BASIC 4)

Format: CALL CLS

Function: Clears the screen in Kanji mode.

**CMT** (command, Hitachi-BASIC version 2)

Format: CALL CMT

Function: Starts the “Tape Utility” on the Hitachi MB-H2 computer.

**COLOR=** (declaration, DM-System2 BASIC)

Format: CALL COLOR = (<palette number>, <red level>, <green level>, <blue level>)

Function: Change the colors of the palette to a single color. The level can vary from 0 to 7 for each primary color. Changes are stored only in (NEWPLT) and not in the VRAM palette table.

**COMBREAK** (remote, BASIC Modem, SVI BASIC Modem)

Format: CALL COMBREAK ([<port number> [,<number of  
characters>]])

Function: Sends instruction to block messages. <port number> can range from 0 to 4 and <character number> to be blocked can range from 3 to 32767.

## COM GOSUB (declaration, Modem BASIC, SVI Modem BASIC)

Format: CALL COM ([<port number> GOSUB <line number>])

Function: Specifies the subroutine that will be called when an interruption occurs in RS232-C. <port number> can vary from 0 to 4; if omitted it will be 0. The subroutine will start at the specified <line number>.

**COMINI** (command, Modem BASIC, SVI Modem BASIC)

Format: CALL COMINI ([<data>] [,<reception speed>]  
[,<transmission speed>] [,<timeout>])

Function: Initializes the modem with the <data> provided.

<data> is an alphanumeric string of up to 10 characters, the default of which, if omitted is "0: 8N1XHNNN". The starting number is the RS232C port followed by ":" and the following characters represent:

3rd - word size (5 to 8) or "del"

4th - parity (E-even, O-odd, I-ignore, N-without parity) or "ins"

5th - size. stop bit: 1- 1 bit, 2- 1.5 bits, 3- 2 bits

6th - XON / XOFF: X- xon, N- xoff

7th - H- handshaking, N- without handshaking

8th - LF: A- inserts LF, N- does not insert LF

9th - LF: A- delete LF, N- do not delete LF

10th - Shift in / out: S- enable, N- disabled

<reception speed> can vary from 50 to 1200. Valid values are: 50, 75, 110, 300, 600, 1200, 1800, 2000, 2400, 3600, 4800, 7200, 9600, 19200. If omitted, it will assume 1200.

<transmission speed> can vary from 50 to 1200. If omitted, it assumes the same as <speed. reception>.

<timeout> is specified in seconds and can range from 0 to 255. If omitted, it will assume 0.

**COMOFF** (remote, BASIC Modem, SVI BASIC Modem)

Format: CALL COMOFF ([<"port number:">])

Function: Disables the interruption coming from the RS232-C port.

**COMON** (command, Modem BASIC, SVI Modem BASIC)

Format: CALL COMON ([<"port number:">])

Function: Enables the interruption coming from the RS232-C port.

**COMSTAT** (function, Modem BASIC, SVI Modem BASIC)

Format: CALL COMSTAT ([<"port number:">],<integer variable>)

Function: Returns the status of the RS232-C port.

<"Port number:"> should vary and 0: to 4 :. If omitted, it will be 0 :.

<integer variable> returns the following values:

- bit 15: Receive buffer overflow error  
0- no error, 1- error occurred
- bit 14: Timeout error  
0- no error, 1- error occurred
- bit 13: Framing error (binary bit "0" was received instead of the stop bit.)  
0- no error, 1- error occurred
- bit 12: Saturation error (data received before the receive buffer is empty)  
0- no error, 1- error occurred
- bit 11: Parity error  
0- no error, 1- error occurred
- bit 10: Pressing [CTRL] + [STOP]  
0- not pressed, 1- pressed
- bit 9: Reserved
- bit 8: Reserved
- bit 7: CS signal status (CTS)  
0- off, 1- on
- bit 6: Timer/counter set for timeout error detection  
0- not defined, 1-defined
- bit 5: Reserved
- bit 4: Reserved
- bit 3: DR signal status (DSR)  
0- off, 1- on
- bit 2: Stop sequence detected while COMSTAT is executed  
0- not detected, 1- detected
- bit 1: Reserved
- bit 0: CD signal status  
0- off, 1- on

### **COMSTOP** (command, Modem BASIC, SVI Modem BASIC)

Format: CALL COMSTOP ([<"port number:">])

Function: Suspend the interruption coming from the RS232-C port.  
<"Port number:"> should vary and 0: to 4:. If omitted, it will be 0:.

**COMTERM** (command, Modem BASIC, SVI Modem BASIC)

Format: CALL COMTERM ([<"port number:">])

Function: Puts the MSX in terminal mode. To exit terminal mode, press CTRL+STOP together. <"Port number:"> should vary and 0: to 4 :. If omitted, it will be 0 :. Once in terminal mode, use the following keys:

[SHIFT] + [F1] – Displays the received control codes.

[SHIFT] + [F2] – Displays the pressed keys.

[SHIFT] + [F3] – Displays and prints the pressed keys.

[STOP] – Press and hold to send the interrupt sequence to the host.

**CONNECT** (command, New Modem BASIC)

Format: CALL CONNECT (<numeric variable>)

Function: Establishes connection in a Terminal program, with the speed defined in CALL INIMDM.

<numeric variable> stores the result of the operation:

0 – An error occurred while trying to connect

1 – Operator detected – The modem is connected

2 – The routine was aborted by pressing the CODE key

3 – The phone number is busy

**CONT MK** (command, MSX-Audio)

Format: CALL CONT MK

Function: Continues a playback or recording of the musical keyboard that was canceled by the STOPM command.

**CONVA** (declaration, MSX-Audio)

Format: CALL CONVA (<source file>, <destination file>)

Function: Convert PCM data to ADPCM data. <source file> and <destination file> are defined by a number that can vary from 0 to 15.

**CONVP** (declaration, MSX-Audio)

Format: CALL CONVP (<source file>, <destination file>)

Function: Convert PCM data to ADPCM data. <source file> and <destination file> are defined by a number that can vary from 0 to 15.

**COPY PCM** (remote, MSX-Audio)

Format: CALL COPY PCM (<source file>, <dest file>, [<source offset>, [<file size>, [<dest offset>]]])

Function: Copies ADPCM and PCM data.

<source file> and <dest file> are defined by a number that can vary from 0 to 15.

<source offset> and <dest offset> define the offset in units of 256 bytes.

<file size> is the file size in bytes.

**COS** (function, DM-System2 BASIC)

Format: CALL COS (<variable>, <angle>, <value>)

Function: Returns the cosine of an angle. The result is obtained by multiplying the cosine of the angle by a numerical value.  
 <variable> – Numeric variable that will receive the result.  
 <angle> – Is the angle value in degrees.  
 <value> – Number of two bytes (integer value).

**CREATEDISK** (command, RookieDrive BASIC)

Format: CALL CREATEDISK (<disc name>)

Function: Creates a new disk image without formatting it. The created disk image is full of 0xFF characters.  
 Experimental instruction and not fully implemented (limited to 720 Kbytes disks).

**CSCAN** (command, Hitachi-BASIC version 2)

Format: CALL CSCAN

Function: Makes the internal data reader of the Hitachi micro MB-H2 searches for files recorded with CSAVE and that can be loaded with CLOAD.

**CSCOPY** (command, Hitachi-BASIC version 3)

Format: CALL CSCOPY (<c1> [,<c2>, <c3>, <c4>.... <c15>])

Function: Sends to the printer a copy of a graphic screen in Screens 2, 4 or 5 using a formula based on the selected colors. The difference with CALL SCOPY is unknown.

**CURDRV** (statement, Nextor)

Format: CALL CURDRV

Function: Displays the active drive unit.

**DEF UNIV** (command, Pioneer-BASIC)

Format: CALL DEF UNIV (<device number>, <device code>)

Function: Defines the device to be controlled by the REMOTE command. <device number> can range from 3 to 15 and <device code> can range from 1 to 255.

**DISCOM** (command, Network-BASIC)

Format: CALL DISCOM (<student number>)

Function: Disables the sending of messages from a student. This instruction is only available to the teacher. By default, after initialization, students can only send messages to the teacher. <student number> can vary from 1 to 15. Short version: \_DISC.

**DMM** (function, DM-System2 BASIC)

Format: CALL DMM (<variable> [,<time>]) [,S])

Function: Performs the device entry and returns the result in <variable>. It can be aborted by CTRL+STOP. (Requires DEV driver).

<variable> must be numeric. The return values are:

0 – Not pressed 10 – GRAPH 13 – ESC

1 to 8 – 8 directions 11 – STOP 14 – HOME

9 – Space 12 – TAB 15 – SELECT

<time> that the command awaits, in units of 1/60 sec.

[, S] – If specified, the sprite defined in "CALL DMMINI" will be moved automatically.

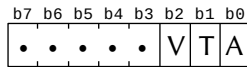
**DMMINI** (declaration, DM-System2 BASIC)

Format: CALL DMMINI ([<mode>] [,<sprite number>])

Function: Defines the device entry. When DM-System2 is started, the mouse and joystick are configured as input devices. (Requires DEV driver).

<mode> is a one-byte value with the following format (default values are 0):





Action to be taken:

0 – coordinate update

1 – return value

Screen output:

0 – loop,                      1 – don't move

Vertical loop range:

0 – 192/212,                1 – 256

<sprite number> is a 1-byte value that specifies the sprite displayed when running CALL DMM. If omitted, "0" is used.

**DMMON** (command, DM-System2 BASIC)

Format: CALL DMMON ([<address>])

Function: Activates continuous device verification, placing the result in the DM-System2 information area. <address> defines the execution address when a device event is detected. Requires DEV driver.

**DMMOFF** (command, DM-System2 BASIC)

Format: CALL DMMOFF

Function: Disables continuous device verification. Requires DEV driver.

**DRIVERS** (statement, Nextor)

Format: CALL DRIVERS

Function: Displays information about the drivers available for Nextor and MSXDOS.

**DRVINFO** (statement, Nextor)

Format: CALL DRVINFO

Function: Displays information about all available drive letters.

**DSK** (command, GR8NET-BASIC)

Format: CALL DSK

Function: Displays help and status and makes diagnostics.

**DSKCFG** (command, GR8NET-BASIC)

Format: CALL DSKCFG (<max num of pages>, <num of pages>)

Function: Get or manage the state of the disk image.

<max number of pages> is a variable that receives the maximum number of logical pages from RAM-Disk.

<number of pages> is a variable or constant that defines the size of the RAM-Disk. It must be between 0 and <max num of pages>.

**DSKFMT** (command, GR8NET-BASIC)

Format: CALL DSKFMT

Function: Initializes the image in RAM-Disk.

**DSKGETIMG** (function, GR8NET-BASIC)

Format: CALL DSKGETIMG [(<string variable>)]

Function: Gets the current location of the disk image and returns the path in the <string variable>.

**DSKHELP** (declaration, GR8NET-BASIC)

Format: CALL DSKHELP

Function: Displays help for GR8NET.

**DSKLDIMG** (command, GR8NET-BASIC)

Format: CALL DSKLDIMG

Function: Load the current disk image into the GR8NET buffer.

**DSKSETIMG** (command, GR8NET-BASIC)

Format: CALL DSKSETIMG [(<path>)]

Function: Defines the location of the image according to <path>, which can be a string variable or an alphanumeric expression.

**DSKSVIMG** (command, GR8NET-BASIC)

Format: CALL DSKSVIMG [(<path>)]

Function: Saves the disk image of the GR8NET buffer to the SD card. If <path> is omitted, the path defined by DSKSETIMG will be used.

**DSKSTATE** (command, GR8NET-BASIC)

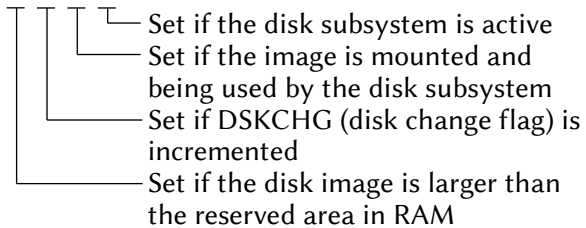
Format: CALL DSKSTATE (<status>, <flags>)

Function: Gets or sets disk subsystem.

<status> defines the state of the system. If it is 0, the disk will be disabled; if it is 1, it will be activated. The change will take effect on the next warm start of the system. Hardware boot will force the return to the default.

<flags> will return with the following values:

| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
|----|----|----|----|----|----|----|----|
| 0  | 0  | 0  | 0  | R  | D  | M  | A  |

**DTROFF** (command, New Modem BASIC)

Format: CALL DTROFF

Function: Disables the DTR (Data Terminal Ready) signal.

**DTRON** (command, New Modem BASIC)

Format: CALL DTRON

Function: Enables the DTR (Data Terminal Ready) signal.

**ECHOOFF** (command, New Modem BASIC)

Format: CALL ECHOOFF

Function: Send characters to the phone line only. The screen will display only received characters.

**ECHOON** (remote, New Modem BASIC)

Format: CALL ECHOON

Function: Enables the sending of characters simultaneously to the telephone line and to the screen.

**EJECT** (command, RookieDrive BASIC)

Format: CALL EJECT

Function: Ejects the currently inserted disk image and deletes its name from the USBMSX.INI file.

**ENACOM** (command, Network-BASIC)

Format: CALL ENACOM (<student number>)

Function: Enables the sending of messages to a student. This instruction is only available to the teacher. By default, after initialization, students can only send messages to the teacher. <student number> can vary from 1 to 15. Short version: \_ENAC.

**ENG** (command, Hangul-BASIC 3)

Format: CALL ENG

Function: Returns to Screen 0 text mode.

**ERASE** (command, SFG-BASIC)

Format: CALL ERASE (<track number>)

Function: Deletes the content of the specified track. <track number> can range from 1 to the number specified by CALL TRACK. Short version: \_ERAS.

**EVENT** (command, SFG-BASIC)

Format: CALL EVENT ([<event number>]) ON | OFF | STOP

Function: Enables, disables or interrupts the interruption by event specified in ON EVENT... GOSUB. <event number> can be:  
 1~4 – Stops when playback of the specified instrument ends.  
 5 – Stops when rhythm playback ends.  
 6 – Interrupts according to the time programmed in the FM unit timer.

If <event number> is omitted, the command will be applied to all events. Short version: \_EVEN.

**EXIT** (command, RMSX-BASIC)

Format: CALL EXIT

Function: Exits the rMSX emulator and returns to the 'normal' use of the MSX Turbo R computer.

**EXT** (command, DM-System2 BASIC)

Format: CALL EXT ([@]<source address>, [@]<dest address>)

Function: Extract compressed data in BPE format.  
 <source address> is the address of the compressed data.

<destination address> is the destination address of the unzipped data.

Note: If “@” is specified, it means VRAM.

### **EXTCOPY** (command, DM-System2 BASIC)

Format: CALL EXTCOPY ([@]<source address>, <X>, <Y>  
[,<direction>]) [<logical operator>]

Function: Unzips data in BPE format to a rectangular area on the screen.

<source address> – Address of the compressed data. If “@” is specified, it means VRAM (maximum 64K).

<X> – Horizontal destination coordinate (0 to 511)

<Y> – Vertical destination coordinate (0 to 1023)

<direction> – Is the unpacking direction on the screen.

0 – Right and down (default)

1 – Left and down

2 – Right and up

3 – Left and up

<logical operator> can be [T]PSET, [T]PRESET, [T]XOR, [T]OR or [T]AND. The default is PSET.

### **EXTV** (remote, Pioneer-BASIC)

Format: CALL EXTV (<variable>)

Function: Checks if there is an external video signal at the input terminal and returns the result in <variable>, that can be:

0 – There is no video signal at the input

1 – External video signal detected

### **FF** (command, Hitachi-BASIC version 2)

Format: CALL FF

Function: Puts the Hitachi MB-H2 computer's built-in data reader into fast search mode.

### **FILEOUT** (command, New Modem BASIC)

Format: CALL FILEOUT (" [<device>:]<filename>"),<variable>)

Function: Send a text file or the typed text directly to a terminal.

<variable> contains an option on the question

“More? (Y/ N)” and then stores a control code.

The values in <variable> can be:

- 0 – This option is on  
1 – The option is off (required for ASCII upload)

## Control codes

- 0 – Text was sent correctly  
3 – The operation was aborted with CTRL+C or C  
7 – The file was not found

**FILES** (declaration, DM-System2 BASIC, RMSX BASIC)

[illegible]

Function: Returns filenames and places them in the DM-System2's work area (address 7A00H). The filenames will be placed one after the other every 12 bytes.

<device> can be drive A: to H: or COM: for computers connected with RS232C.

**<path>** specifies the location of the folder or file

<filename> accepts wildcards (\* and?)

<variable> is a numeric variable that will receive the number of files found.

Format: CALL FILES ("[<device>:] [\ <path>] [[\]<filename>])  
[RMSX BASIC]

Function: Lists the contents of an actual disk in an MSX Turbo R drive, used to host an MSX1 or MSX2 emulation with the rMSX emulator.

<device> can be drive A: to H .:

`<path>` specifies the location of the folder or file.

<filename> accepts wildcards (\* and ?).

**FIND** (command, MSX-Aid BASIC)

Format: CALL FIND ("variable" [, [starting line number],  
[ending line number], [P]))

Function: List part of the MSX-BASIC program that is in memory, where an alphanumeric variable or specific string is used. <variable> must be one or two characters. You can also specify the <starting line number> and <ending line number> of the BASIC program to be listed. If [P] is specified, the listing will be sent to the printer.

**FLINFO** (declaration, GR8NET-BASIC)

Format: CALL FLINFO

Function: Displays information about the serial flash memory.

**FLLIST** (declaration, GR8NET-BASIC)

Format: CALL FLLIST

Function: Lists the contents of the serial flash memory.

**FLUPDATE** (command, GR8NET-BASIC)

Format: CALL FLUPDATE (<sector> [,F])

Function: Updates the contents of the serial flash memory. <sector> specifies the sector number where the update will begin. If parameter “, F” is included, the update will start immediately without asking for confirmation.

**FNAME** (declaration, RookieDrive BASIC)

Format: ?

Function: ?

**FONT** (remote, Hangul-BASIC 4)

Format: CALL FONT

Function: Enables alternation between Korean characters (HANGUL key) and non-Korean characters available through the KANA, CYRILLIC or CODE keys. It will return error if used in Screen 9.

**FORMAT** (command, Disk-BASIC, FormatM. BASIC, RookieD. BASIC)

Format: CALL FORMAT [Disk-BASIC]

Function: Formats a floppy disk. It offers two options:

1 – 1 side, double track (single face, 360K)

2 – 2 sides, double track (double side, 720K)

Format: CALL FORMAT [FormatMaster-BASIC]

Function: Formats a floppy disk offering additional options. Disk-BASIC version 1 required (this instruction is not compatible with Disk-BASIC version 2).

1) 40 tracks – 8 sectors per track – FA

2) 80 tracks – 8 sectors per track – FB

3) 40 tracks – 9 sectors per track – F8

4) 80 tracks – 9 sectors per track – F9

Format: CALL FORMAT [RookieDrive BASIC]

Function: Formats the disc inserted in a standard USB floppy drive or disc image connected to a Rookie Drive interface. Offers 4 options:

- 1) 720K, full format
- 2) 720 K, fast format
- 3) 1.44M, full format
- 4) 1.44M, fast format

**FRAME** (statement, Pioneer-BASIC)

Format: CALL FRAME (<frame number>, GOSUB <line number>)  
CALL FRAME OFF

**Function:** Specifies the subroutine line number that will be executed when the frame <frame number> is reached. <frame number> must be in the range between 50 and 54,000. If “OFF” is specified, cancel the line number assignment. This command is specific for use with the Pioneer Laser Vision Player LD-700 and cannot be used in conjunction with the CHAPTER command.

**FSIZE** (function, DM-System2 BASIC)

Format: CALL FSIZE ("[<device>:] [\ <path>] [[\]<filename>]",  
<variable>)

Function: Returns the file size  
 <device> can be drive A: to H: or COM: for computers connected with RS232C.  
 <path> specifies the location of the folder or file  
 <filename> accepts wildcards (\* and?)  
 <variable> receives the file size.

**GET** (function, New Modem BASIC)

Format: CALL GET (<variable>)

Function: Retrieves the ASCII code of a character pressed on the keyboard or received on the telephone line (if that line has not been deactivated with CALL LINEOFF). <variable> stores the ASCII code. Special shortcuts in a BBS program: Pause: CTRL+S or S; continue (after a pause): any key; stop: CTRL+C or C.



- HANOFF** (command, Hangul-BASIC 1)  
 Format: CALL HANOFF  
 Function: Disables the feature to group characters in blocks (characteristic of Hangul characters, used in Korea, available after pressing the HANGUL key). Returns error if used in Screen 9.
- HANON** (command, Hangul-BASIC 1)  
 Format: CALL HANON  
 Function: Enables the feature to group characters in blocks (characteristic of Hangul characters, used in Korea, available after pressing the HANGUL key). Returns error if used in Screen 9.
- HCOPY** (command, Hitachi-BASIC version 2-3)  
 Format: CALL HCOPY  
 Function: Sends to the printer a copy of the text screen (Screens 0 or 1), on Hitachi MB-H2 and MB-H3 computers.
- HELP** (declaration, DM-System2 BASIC, Hangul-BASIC 4, MSX Aid BASIC, Netw. BASIC, RMSX BASIC, RookieD. BASIC)  
 Format: CALL HELP  
 Function: Shows help on BASIC in use.
- HIRO** (remote, MSX turbo R model FS-A1ST)  
 Format: CALL HIRO  
 Function: Calls the menu for programs in ROM on the MSX turbo R model FS-A1ST. For FS-A1GT, use CALL MWR.
- HMMM** (declaration, DM-System2 BASIC)  
 Format: CALL HMMM (<X0>, <Y0>) – [STEP] (<X1>, <Y1>) TO (<X2>, <Y2>)  
 Function: Executes the VDP HMMM (quick copy in bytes) command. Available for Screens 5 to 12.  
 <X0> – X coordinate of the first point in the source area.  
 <Y0> – Y coordinate of the first point in the source area.  
 <X1> – X coordinate of the second point in the source area.  
 <Y1> – Y coordinate of the second point in the source area.  
 <X2> – Left X coordinate of the target area.  
 <Y2> – Upper Y coordinate of the target area.  
 STEP, if specified, indicates relative coordinates.  
 Note: <X> can vary from 0 to 511 and <Y> from 0 to 1023.

**HMMV** (declaration, DM-System2 BASIC)

Format: CALL HMMV (<X0>, <Y0>) – [STEP] (<X1>, <Y1>) <v>

Function: Executes the VDP's HMMV command (Quick Paint VRAM).

Available for Screens 5 to 12.

<X0> – X coordinate of the first point in the area.

<Y0> – Y coordinate of the first point in the area.

<X1> – X coordinate of the second point in the area.

<Y1> – Y coordinate of the second point in the area.

<v> - byte to be sent to VRAM. Specifies one point for Screens 8 to 12, two points for Screens 5 and 7, and four points for Screen 6.

STEP, if specified, indicates relative coordinates.

Note: <X> can vary from 0 to 511 and <Y> from 0 to 1023.

**HZ** (control, RMSX-BASIC)

Format: CALL HZ [50 | 60]

Function: Select the screen refresh rate (VDP frequency). No parameter toggles rates.

50 - The VDP will have a frequency of 50 Hz (European, Russian or Arabic MSX).

60 - The VDP will have a frequency of 60 Hz (Japanese, Korean or Brazilian MSX).

**IDTRACE** (command, Hitachi-BASIC version 2)

Format: CALL IDTRACE

Function: Puts the built-in data reader of the Hitachi MB-H2 in ID tracking mode to verify if the correct tape has been inserted in the reader.

**IMPOSE** (remote, Pioneer-BASIC)

Format: CALL IMPOSE (<mode>)

Function: Selects the video mode.

<mode> can be:

0 – Computer screen (internal synchronization)

1 – Superimpose (composite video)

2 – External video

**INIMDM** (command, New Modem BASIC)

Format: CALL INIMDM (&lt;variable&gt;)

Function: Initializes the modem speed according to the &lt;variable&gt;, whose value is described in the table below:

| Value | Standard | Recep.<br>Speed | Transm.<br>Speed | Note               |
|-------|----------|-----------------|------------------|--------------------|
| 0     | V21      | 300 baud        | 300 baud         | Caller             |
| 1     | V21      | 300 baud        | 300 baud         | Receiver           |
| 2     | V23      | 1200 baud       | 75 baud          | -                  |
| 3     | V23      | 75 baud         | 1200 baud        | -                  |
| 4     | V23      | 1200 baud       | 75 baud          | for bad connection |
| 5     | V23      | 75 baud         | 1200 baud        | for bad connection |
| 6     | V23      | 600 baud        | 75 baud          | -                  |
| 7     | V23      | 75 baud         | 600 baud         | -                  |

**INIT** (command, SFG-BASIC)

Format: CALL INIT

Function: Initializes the FM Music Macro.

**INITMD** (command, New Modem BASIC)

Format: CALL INITMD

Function: Initializes the X8N1 communication protocol.

X = Xon / Xoff protocol enabled

8 = 8 bits of data

N = without parity

1 = One stop bit

**INMK** (function, MSX-Audio)Format: CALL INMK [[[<variable 1>] [, [<variable 2>]]  
[,<variable 3>]]]

Function: Reports changes when using the musical keyboard.

&lt;variable 1&gt; – key number (0 to 127)

&lt;variable 2&gt; – key status (0 if pressed, otherwise 1)

&lt;variable 3&gt; – Frequency of ADPCM corresponding to the key pressed.

**INMKEY** (function, SFG-BASIC)

Format: CALL INMKEY (&lt;variable&gt;)

Function: Checks if any key on the musical keyboard is being pressed. &lt;variable&gt; returns the key code. If it is 0, no key is being pressed. Short version: \_INMK.

**INSERTDISK** (command, RookieDrive BASIC)

Format: CALL INSERTDISK ("<disk name>")

Function: Insert a new disk image into the USB virtual drive.  
Currently, it is limited to disk images with a maximum of 720 Kbytes.

**INST** (statement, SFG-BASIC)

Format: CALL INST (<instrument number> [,<number of voices>]  
[,<MIDI>] [,<MIDI channel>])

Function: Defines the instruments to be used by the FM Music Macro  
Up to 4 instruments can be defined by this command.

<instrument number> can vary from 1 to 4.

<number of voices> specifies the number of simultaneous  
voices used by the instrument. It can vary from 1 to 8.

<MIDI> specifies whether the data will be sent to the MIDI  
interface. "MIDI ON" uses the MIDI interface and  
"MIDI OFF" does not (default).

<MIDI channel> can range from 1 to 16. If omitted, channel  
1 will be used.

**INTWAIT** (command, DM-System2 BASIC)

Format: CALL INTWAIT

Function: Pauses the system until the next interruption of DM  
System2. It can be aborted by CTRL+STOP.

**IOSOUND** (command, RMSX-BASIC)

Format: CALL IOSOUND [ON] | [OFF]

Function: Enables or disables sounds from emulated cassettes and  
disks.

ON – Activates all I/O sounds.

OFF – Disables all I/O sounds.

**JIS** (statement, Kanji-BASIC)

Format: CALL JIS (<string variable>, <character string>)

Function: Converts the first character in a string to a 4-digit  
hexadecimal JIS code.

<string variable> receives the hexadecimal code

<character string> contains the characters to be converted.

**KACNV** (statement, Kanji-BASIC)

Format: CALL KACNV (<string variable>, <character string>)

Function: Converts two-byte Kanji characters to one-byte characters.  
 <string variable> receives the converted characters  
 <character string> contains the Kanji characters to be converted.

**KANJI** (command, Kanji-BASIC)

Format: CALL KANJI [<n>]

Function: Activates Kanji mode. <n> can range from 0 to 3, but modes 1 to 3 only work on an MSX2 or higher. When in Kanji mode, press CTRL + SPACE or GRAPH + SELECT to activate Kanji input mode.

- 0 – 13 lines of 32 or 64 characters (16x16 or 8x16)
- 1 – 13 lines of 40 or 80 characters (12x16 or 6x16)
- 2 – 24 lines of 32 or 64 characters in interlaced mode (16x16 or 8x16)
- 3 – 24 lines of 40 or 80 characters in interlaced mode (16x16 or 8x16)

Note: In Kanji mode, the commands CLS, COLOR= And SCREEN 9 are disabled.

**KBOLD** (declaration, DM-System2 BASIC)

Format: CALL KBOLD ([<width>] [,<height>] [,<X edge>] [,<Y edge>] [,<X shadow>] [,<Y shadow>])

Function: Defines the style of the text characters. (Requires FNT driver).

- <width> of the character (1 to 16, default is 1)
- <height> of the character (1 to 16, default is 1)
- <edge X> – X edge thickness (1 to 8, default is 1)
- <edge Y> – Y edge thickness (1 to 8, default is 1)
- <shadow X> – Horizontal thickness of the shadow character (1 to 32, default is 1)
- <shadow Y> – Vertical thickness of the shadow character (1 to 32, default is 1)

If the thickness of the shadow is 0, it will be determined automatically.

**KCHR** (function, Hangul-BASIC 3)

Format: CALL KCHR (<string variable>, <hexadecimal code>)

Function: Returns in <string variable> the Korean character specified by the 4-digit <hexadecimal code>.

**KCODE** (function, Hangu-BASIC 3)

Format: CALL KCHR (<string variable>, <string>)

Function: Returns in <string variable> the 4-digit hexadecimal code of the first Korean character in <string>.

## KCOLOR (declaration, DM-System2 BASIC)

Format: CALL KCOLOR ([<character color>] [<background color>]  
,<border color>] [<shadow color>])

Function: Defines the text characters colors. (Requires FNT driver).  
 <character color> can range from 0 to 15 (default: 15)  
 <background color> can range from 0 to 15 (default: 0)  
 <border color> works only for the “border” function. It can vary from 0 to 15 and the default is 1.  
 <shadow color> works only for the “shadow” function. It can vary from 0 to 15 and the default is 14.

**KEXT** (function, Kanji-BASIC, Hangel-BASIC 3)

Format: CALL KEXT (<string variable>, <char string>, <function>)

Function: Extracts only 2 bytes or 1 byte characters from a string.  
 <string variable> receives the extracted characters  
 <character string> contains the characters to be extracted.  
 <function> – If 0, only one byte character will be extracted  
                   for Kanji-BASIC or non-Korean characters for  
                   Hangul-BASIC. If it is 1, only 2 byte characters will be  
                   extracted for Kanji-BASIC or Korean characters for  
                   Hangul-BASIC.

**KEY ON / OFF** (function, MSX-Audio)

Format: CALL KEY ON (<key number>, <speed>)

CALL KEY OFF (<key number>)

Function: Informs if the key is pressed or released regardless of its real condition.

<key number> can range from 0 to 127.

<speed> can range from 0 to 15 (8 is the default).

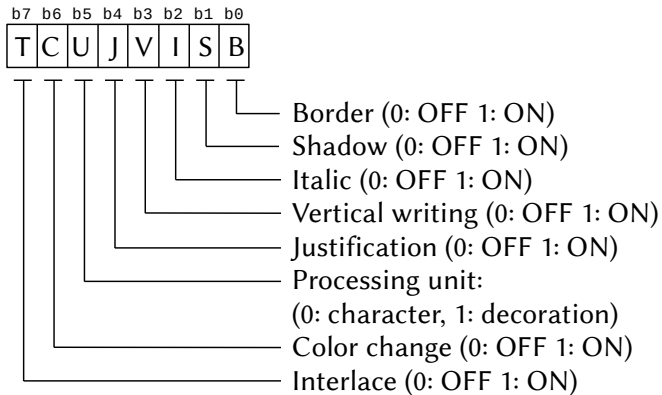
**KINIT** (declaration, DM-System2 BASIC)

Format: CALL KINIT

Function: Returns all text definitions, including KBOLD and KSIZE, to their default values. (Requires FNT driver).

Format: CALL KINIT ([<configuration>] [,<shadow X>  
[,<shadow Y>] [,<italic>] [,<color table>])

Function: Defines all the decoration options for the character.  
 <configuration> is a one-byte value, with the following flags (the initial value for all is 0):



<shadow X> and <shadow Y> define the position of the shadow in relation to the upper left corner of the character, which can vary between -128 and 127.

*<italic>* defines the offset to the right of each line of the character, including the border, which can vary between -128 and 127. If omitted, system values will be used.

<color table> defines the colors for each character line from the top, including the border. If omitted, the system color table will be used. The default value is C000H.

**KINSTR** (function, Kanji-BASIC, Hangeul-BASIC 3)

Format: CALL KINSTR (<numeric variable> [<search start>],  
<string 1>, <string 2>)

Function: Searches for the occurrence of <string 2> in <string 1> and returns the position in <numeric variable>. If there are no occurrences, it returns 0. <search start> is an optional value and indicates the position of the start char for the search.

**KLEN** (function, Kanji-BASIC, Hangul-BASIC 3)

Format: CALL KLEN (<numeric variable>, <character string>,  
[<function>]) [Kanji-BASIC]

Format: CALL KLEN (<numeric variable>, <character string>)  
[Hangu-BASIC]

Function: Returns in <numeric variable> the size of the <character string>. If <function> is 0 or omitted, returns the total number of characters; if it is 1, it returns the number of 1-byte characters and if it is 2 it returns the number of 2-byte characters. Hangul-BASIC does not allow the <function> parameter.

**KMID** (function, Kanji-BASIC, Hangul BASIC 3)

Format: CALL KMID (<string variable>, <character string>, <shift>  
[,<size>])

Function: Extract <size> characters from position <shift> of the <character string> and put it in <variable string>.

**KNJ** (statement, Kanji-BASIC)

Format: CALL KNJ (<string variable>, <character string>)

Function: Assigns the <string variable> a kanji character equivalent to the 4-digit hexadecimal kanji code specified in <character string>. When the kanji code is less than 8000H, it will be considered as JIS; when it is larger it will be considered as a JIS shift.

**KPRINT** (declaration, DM-System2 BASIC)

Format: CALL KPRINT (<character string>, [<limit character>],  
<logic operation code>]

Function: Prints a kanji string on the screen.

**KPUT** (declaration, DM-System2 BASIC)

Format: CALL KPUT (<string> [,<number of characters>])

Function: Displays a string at high speed. (Requires FNT driver).  
 <string> is the string to be displayed.  
 <number of characters> is the maximum number of  
 characters to be displayed. If omitted, all characters  
 will be displayed.



**KSIZE** (declaration, DM-System2 BASIC)

Format: CALL KPUT (<width>, <height> [,<separation>])

Function: Defines the character size of a byte. (Requires FNT driver).  
 <width> can vary from 1 to 32 (default: 8)  
 <height> can vary from 1 to 64 (default: 16)  
 <partition> defines the space between characters and can vary from 0 to 15 (default: 0)

**KTYPE** (function, Kanji-BASIC, Hangul-BASIC 3)

Format: CALL KTYPE (<numeric variable>, <character string>, <character position>)

Function: Returns in the <numeric variable> the value 0 if the character corresponding to the <character position> in the <character string> is one byte and the value 1 if the character is 2 bytes.

**LB** (command, New Modem BASIC)

Format: CALL LB (<string variable>) [:]

Function: Sends a text to the screen and/or the phone line according to the following table:

| Eco          | Line         | Screen | Phone |
|--------------|--------------|--------|-------|
| CALL ECHOON  | CALL LINEON  | Yes    | Yes   |
| CALL ECHOON  | CALL LINEOFF | Yes    | No    |
| CALL ECHOOFF | CALL LINEON  | No     | Yes   |

The text can be paused with CTRL+S but cannot be interrupted.

**LCOPY** (remote, Printer-BASIC or Pioneer-BASIC)

Format: CALL LCOPY [Printer-BASIC]

Function: Prints the data that is still in the temporary buffer of 32 Kbytes when the print spooler is used.

Format: CALL LCOPY (<mode>) [Pioneer-BASIC]

Send a copy of the Screen 2 to the printer. If <mode> is 0, make a positive copy and if it is 1, make a negative copy.

**LD** (remote, Pioneer-BASIC)

Format: CALL LD

Function: Executes the interactive software present on a CPE (Computer Program Encoded) disk.

**LEN** (function, New Modem BASIC)

Format: CALL LEN (<string variable>,<numeric variable>

Function: Returns the length of a string, without the final control chars (space, tab, return, etc.). The <numeric variable> will return the length in printable characters of the <string var>.

**LENGTH** (function, SFG-BASIC)

Format: CALL LENGTH ([<track 1>] [,<track 2>]... [,<track 8>])

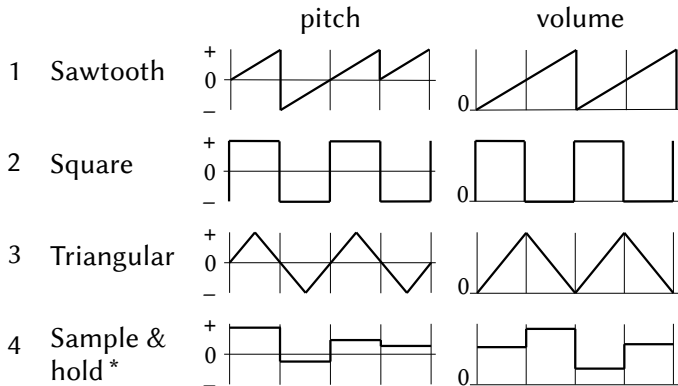
Function: Returns the size of the data on a music track. The units returned correspond to 1/192 of an entire note. <track 1> to <track 8> are numeric variables.

**LFO** (declaration, SFG-BASIC)

Format: CALL LFO (<waveform number> [,<speed>] [,<tremolo>]  
[,<vibrate>])

Function: Defines the LFO (Low Frequency Oscillator) data.

<waveform number> can vary from 1 to 4:



\* "Sample & Hold" values are random.

<speed> specifies the frequency of the LFO in relation to the volume. It can vary from 1 to 100. The higher, the higher the frequency and the speed.

<tremolo> specifies the modulation in relation to the volume. It can vary from 1 to 100. The higher, the more the volume will be changed.

<vibrate> specifies the frequency modulation (pitch). It can vary from 1 to 100. The higher, the more the pitch will be changed.

**LICENSE** (declaration, RMSX-BASIC)

Format: CALL LICENSE

Function: Displays license information about the used version of the rMSX emulator, developed by the Finnish NYYYRIKKI for Turbo R. computers.

**LINEOFF** (command, New Modem BASIC)

Format: CALL LINEOFF

Function: Hang up the phone line but keep the connection active.

**LINEON** (remote, New Modem BASIC)

Format: CALL LINEON

Function: Connects the telephone line.

**LMMM** (declaration, DM-System2 BASIC)Format: CALL LMMM (<X0>, <Y0>) – [STEP] (<X1>, <Y1>) TO  
(<X2>, <Y2>) [<logical operator>]

Function: Executes the VDP command LMMM (logical copy in points). Available for Screens 5 to 12.

&lt;X0&gt; – X coordinate of the first point in the source area.

&lt;Y0&gt; – Y coordinate of the first point in the source area.

STEP, if specified, indicates relative coordinates.

&lt;X1&gt; – X coordinate of the second point in the source area.

&lt;Y1&gt; – Y coordinate of the second point in the source area.

&lt;X2&gt; – Left X coordinate of the target area.

&lt;Y2&gt; – Upper Y coordinate of the target area.

&lt;logical operator&gt; can be [T] PSET, [T] PRESET, [T] XOR, [T] OR or [T] AND. The default is PSET.

Note: &lt;X&gt; can vary from 0 to 511 and &lt;Y&gt; from 0 to 1023.

**LMMV** (declaration, DM-System2 BASIC)Format: CALL LMMV (<X0>, <Y0>) – [STEP] (<X1>, <Y1>), <color>  
[<logical operator>]

Function: Executes the LMMV command (logical copy in points from VDP to VRAM). Available for Screens 5 to 12.

&lt;X0&gt; – X coord of the first point in the destination area.

&lt;Y0&gt; – Y coord of the first point in the destination area.

STEP, if specified, indicates relative coordinates.

<X1> – X coord of the second point in the destination area.

<Y1> – Y coord of the second point in the destination area.

<color> specifies the color of the rectangle to be painted.

<logical operator> can be [T] PSET, [T] PRESET, [T] XOR, [T] OR or [T] AND. The default is PSET.

Note: <X> can vary from 0 to 511 and <Y> from 0 to 1023.

**LO** (command, New Modem BASIC)

Format: CALL LO (<string variable>) [;,<variable>]

Function: Sends a text to the screen and/or the phone line according to the following table:

| Eco          | Line         | Screen | Phone |
|--------------|--------------|--------|-------|
| CALL ECHOON  | CALL LINEON  | Yes    | Yes   |
| CALL ECHOON  | CALL LINEOFF | Yes    | No    |
| CALL ECHOOFF | CALL LINEON  | No     | Yes   |

Any characters received are ignored, except for the pause, abort and continue keys. The key buffers remain empty; no keys are stored. The final control codes are sent with the text.

<variable> parameter returns the result of the execution:

0 = the text was sent correctly

3 = the operation was aborted with CTRL+C or C

**LOAD** (command, DM-System2 BASIC, QuickDisk BASIC)

**Format:** CALL LOAD ("[<device>:] [\ <path>] []<filename>", [@  
                    <destination address> [,<size>] [,<offset>])  
  [DM-System2 BASIC]

Function: Reads a file or part of it.

<device> can be drive A: to H: or COM: for computers connected with RS232C.

<path> specifies the location of the folder or file.

<filename> is the file to be read.

<destination address> is the destination address for the data. If preceded by "@" it means VRAM.

`<size>` specifies the number of bytes to read.

<offset> specifies the offset in the source file.

Format: CALL LOAD ([QD[n]:]"<filename>"[,R]) [QuickDisk-BASIC]

Function: Load a non-binary file from the specified Quick Disk device. It can be a BASIC file in tokenized mode or in ASCII text. QD [n] specifies the QuickDisk device to be used. It can range from 0 to 7, the default being 0. <filename> must be in the format 8 chars + "." + 3 chars. [,R], if specified, runs the BASIC file right after loading.

### **LOAD PCM** (command, MSX-Audio)

Format: CALL LOAD PCM (<"filename">, <file number>)

Function: Load ADPCM and PCM data from disk. <filename> – filename on disk. <file number> – File number in the audio memory. It can range from 0 to 15.

### **LOADROM** (command, RookieDrive BASIC)

Format: CALL LOADROM ("<filename>")

Function: Loads an 8kb, 16kb or 32kb ROM file into RAM and starts its execution by restarting the computer. The ROM file must be located in the root directory of the USB device. <filename> must be in 8.3 format.

### **LOCKDRV** (command, Nextor)

Format: CALL LOCKDRV (<drive letter>: N)

Function: Lock or unlock drive letters, or display the list of blocked drives. If "N" is 0, it unlocks the drive; any other number locks.

### **LOGFILE** (command, New Modem BASIC)

Format: CALL LOGFILE (<filename>) [;],<variable>

Function: Stores everything on the screen in a text file. <filename> must be a string variable.

### **LOOK** (function, SFG-BASIC)

Format: CALL LOOK ([<instrument 1>] [,<instrument 2>] [,<instrument 3>] [,<instrument 4>])

Function: Returns the status of the instrument, whether it is being played or not. The <instrument 'n'> parameters are numeric variables. For instrument not defined by \_INST, it returns 0.

**LS** (remote, New Modem BASIC)

Format: CALL LS (<string variable>) [;],<variable>

Function: Sends a text to the screen and / or the phone line according to the following table:

| Eco          | Line         | Screen | Phone |
|--------------|--------------|--------|-------|
| CALL ECHOON  | CALL LINEON  | Yes    | Yes   |
| CALL ECHOON  | CALL LINEOFF | Yes    | No    |
| CALL ECHOOFF | CALL LINEON  | No     | Yes   |

Any characters received are ignored, except for the pause, abort and continue keys. The key buffers remain empty; no keys are stored. The final control codes are sent with the text. The <variable> parameter returns the result of the execution:

0 = the text was sent correctly.

3 = the operation was aborted with CTRL+C or C.

**MALLOC** (command, DM-System2 BASIC)

Format: CALL MALLOC ([<number of pages>] [,<variable>])

Function: Enables access to the Memory Mapper.

<number of pages> is the number of pages to be allocated.

If it is 0, the allocated pages will be released. If omitted, the current number of allocated pages will return in <variable>.

<variable> is a numeric variable that will contain the number of pages actually allocated.

**MAPDRV** (command, Nextor)

Format: CALL MAPDRV (<drive> [,<partition> [,<device> [,<slot> | 0]])

Function: Maps a drive unit in the Nextor system.

<drive> letter or drive number to be mapped

<partition> is a number as following:

0 – The drive will be mapped from the device's absolute zero sector.

1 – First primary partition

2 to 4 – Refer to extended partitions 2.1 to 2.4, if partition 2 is extended; otherwise, they refer to primary partitions.

5 – Onwards refer to extended partitions.

<device> – Device index (1 to 7)

<slot> – Slot number (0 to 3). If the slot is expanded, use the formula  $\text{<main slot>} + 4 * \text{<subslot>}$ . If “0” is specified, the primary unit slot will be selected.

**MAPDRV** (command, Nextor)

Format: CALL MAPDRV (<drive> [, <partition> [, <device> [, <slot> | 0]])

Function: Maps a drive unit in the Nextor system and locks the specified drive. The parameters are identical to MAPDRV.

**MC** (declaration, New Modem BASIC)

Format: CALL MC (<string variable>)

Function: Converts the alphabetic characters of the <string variable> to uppercase.

**MDR** (command, MSX turbo R model FS-A1GT)

Format: CALL MDR

Function: Activates the MSX-MUSIC output to the MIDI interface. Only MSX turbo R model FS-A1GT.

**MEMINI** (command, 2)

Format: CALL MEMINI [(RAM disk size)]

Function: Activates the RAM disk in the lower 32K of memory.

**MERGE** (command, QuickDisk BASIC)

Format: CALL MERGE ("[QD [n]:]<filename>")

Function: Merges a BASIC or DATA program saved in ASCII on the QuickDisk device with the program that is in the MSX memory.

QD [n] specifies the QuickDisk device to be used. It can range from 0 to 7, the default being 0.

<filename> must be in the format 8 chars + “.” + 3 chars.

**MESSAGE** (statement, MSX-Aid BASIC, Network BASIC)

Format: CALL MESSAGE [MSX-Aid BASIC]

Function: Displays an encouraging message for programmers using MSX-Aid.

Format: CALL MESSAGE (<message>, [<student number>])  
[Network BASIC]

Function: Sends a message of up to 56 characters to a specific student. This instruction is only available to the teacher. <student number> can range from 1 to 15. The short version: \_MESS can be used.

**MFADE** (declaration, StudioFM BASIC)

Format: CALL MFADE (<degree of fade>)

Function: Produces a fade-out when playing back .MUS files in the Studio FM. <degree of fade> can vary between 0 and 255, with 0 no fade and 255 will produce the longest fade-out.

**MFILES** (command, 2)

Format: CALL MFILES

Function: Lists the RAM disk files of the lower 32K of memory.

**MK PCM** (declaration, MSX-Audio)

Format: CALL MK PCM (<file number>)

CALL MK PCM OFF

Function: Defines which ADPCM file will be played as an instrument. If specified OFF, it cancels the previously defined instrument. <file number> can range from 0 to 15.

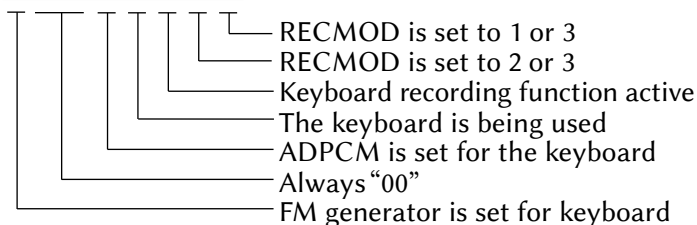
**MK STAT** (function, MSX-Audio)

Format: CALL MK STAT (<variable>)

Function: Returns the recording or playback status of the musical keyboard.

<variable> is a numerical value defined according to the figure below.

| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
|----|----|----|----|----|----|----|----|
| FM | 0  | 0  | AD | KB | KR | R2 | R1 |





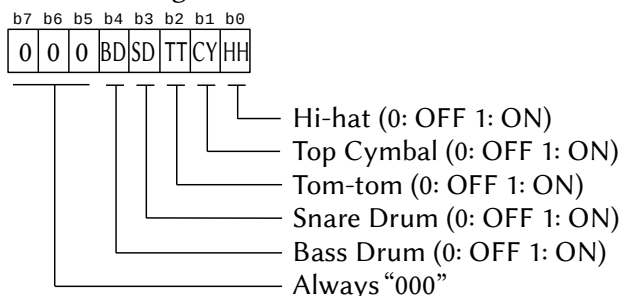
**MK TEMPO** (statement, MSX-Audio)

Format: CALL MK TEMPO (<speed>, <percussion map>)

Function: Specifies the recording / playback speed of the musical keyboard or activates the metronome function. In this case, the AUDIO command must be previously defined. This command affects the speed of the MK PLAY, MK REC and MK APPEND instructions.

<speed> must be in the range 25~360, the initial value being 120.

<percussion map> is a numerical value defined according to the figure below.

**MK VEL** (statement, MSX-Audio)

Format: CALL MK VEL (<speed>)

Function: Specifies the speed, or pressure force, that is applied to a key on the musical keyboard.

<speed> can range from 0 to 15, the initial value being 8.

**MK VOICE** (statement, MSX-Audio)

Format: CALL MK VOICE ([@]<instrument number>)

Function: Defines the instrument to be associated with the musical keyboard. <instrument number> is a numeric variable that defines the instrument number, which can vary from 0 to 63. If there is no @, the variable will be assumed to be a matrix, where the values in sequence define the instrument.

**MK VOL** (statement, MSX-Audio)

Format: CALL MK VOL (<volume>)

Function: Defines the volume associated with the musical keyboard. <volume> can range from 0 to 63.

**MKDIR** (command, Disk-BASIC 2nd version)

**Format:** CALL MKDIR (<subdirectory>)

Function: Creates the <subdirectory> with the specified name.

**MKILL** (command, 2)

Format: CALL MKILL (“<filename>”)

Function: Deletes the file <filename> from the RAM disk of the lower 32K of memory.

## MLOAD (command, StudioFM BASIC)

Format: CALL MLOAD (“<filename>”,<address>)

Function: Loads a song in the Studio FM format (.MUS).

**MNAME** (command, 2)

Format: CALL MNAME (“<filename1>” AS “<filename2>”)

Function: Renames file <filename1> with <filename2> on the RAM disk of the lower 32K of memory.

**MODE9** (remote, Hangul-BASIC 4)

Format: CALL MODE9

Function: Go to Screen 9 automatically using Width 80.

### MODINST (declaration, SFG-BASIC)

Format: CALL MODINST (<instrument> [,<voice>]  
[,<transposition>] [,<volume>] [,<portamento>]  
[,<portamento speed>] [,<support>] [,<trigger mode >  
[,<LFO sync>] [,<tremolo>] [,<vibrate>])

Function: Changes the instrument data. Short version: `_MODI`.

<instrument> specifies the tone of the instrument defined by \_INST.

<voice> specifies the pitch of the instrument.

1~48 – Selects the tone of the FM Sound Synthesizer ROM. 47 and 48 are reserved and do not contain tones.

49~56 – Select tones defined by \_SEL.

<transposition> allows the various instruments to be transposed separately. It can range from -12 to +12 in half-step intervals.

- <volume> sets the volume separately for each instrument. It can vary from 0 to 100, with 100 being the maximum volume.
- <portamento> can take two values:
  - 0 – Portamento only during playback
  - 1 – Portamento all the time
- <portamento speed> can vary from 0 to 100, with 100 being the slowest. 0 turns off the portamento.
- <support> can take on two values:
  - 0 – Standard lift time
  - 1 – Support time is doubled
- <trigger mode> determines whether to trigger when the keys pressed are released.
  - 0 – “attack” when the keys are released
  - 1 – No “attack” during legacy playback
- <LFO sync> determines whether there will be synchronization with the LFO when the pressed keys are released.
  - 0 – No synchronization
  - 1 – The LFO starts the waveform whenever a key is released.
- <tremolo> specifies the degree of the tremolo. It can vary from 0 to 100, with 100 being the highest degree of sensitivity. (despite the range 0 to 100, there are only 4 degrees of tremolo).
- <vibrate> specifies the degree of vibrato. It can vary from 0 to 100, with 100 being the highest degree of sensitivity. (despite the range 0 to 100, there are only 8 degrees of vibrato).

**MON** (command, FM-X BASIC, Hitachi BASIC, MSXAid BASIC)  
 Format: CALL MON [FM-X BASIC]  
 Function: Starts the monitor when the Fujitsu FM-X is connected to the FM-7 machine with the MB22450 interface.  
 Format: CALL MON [Hitachi BASIC 1-2]  
 Function: Starts the System Monitor Utility command line on the Hitachi MB-H1 and MB-H2 computers. For a list of all commands available in this utility, type H on the command line.

Format: CALL MON [MSXAid BASIC]

Function: Starts the internal monitor of the MSX-Aid utility. To get help using this monitor, first enter a RAM address and then press F6.

## MOUNT (command, RookieDrive BASIC)

Format: ?

Function: ?

**MPLAY** (statement, StudioFM BASIC)

Format: CALL MPLAY (<memory address>)

Function: Plays a song in the StudioFM (.MUS) format loaded in memory with \_MLOAD.

**MRING** (command, New Modem BASIC)

Format: CALL MRING (<numeric variable>)

Function: Checks whether the phone is ringing. <numeric variable> can return the following values:

0 – The phone is ringing

1 – The phone is not ringing

2 – Routine interrupted (CODE key pressed)

**MSTART** (command, New Modem BASIC)

Format: CALL MSTART (<numeric variable>)

Function: Repare the modem for data communication. If <numeric variable> returns 0, everything is fine; otherwise there was an error.

**MSTOP** (command, New Modem BASIC, StudioFM BASIC)

Format: CALL MSTOP [New Modem BASIC]

Function: Interrupts the modem's functions.

Format: CALL MSTOP [StudioFM BASIC]

Function: Stops playback of a song in StudioFM format (.MUS) played in the background.

**MUSIC** (command, MSX-Music)

Function: Starts MSX-MUSIC and determines which voices will be used and how.

<n1> can be:

0 – Select pure melody mode (n3~n9 can be specified)

1 – Select melody + battery mode (n3~n6 can be specified)

<n3> to <n9> can be:

1 – Select melody

2 – Select battery

**MUTE** (control, Hitachi-BASIC, Pioneer-BASIC, RMSX-BASIC)

Format: CALL MUTE [Hitachi-BASIC 2]

Function: Adds a 4 second pause before recording data to the internal register of the Hitachi HB-M2.

Format: CALL MUTE [R | L] [Pioneer-BASIC]  
CALL MUTE OFF

Function: Mutes the right (R), left (L) audio channels or both if there is no channel specification. If OFF is specified, the mute function is canceled.

Format: CALL MUTE [ON] | [OFF] [RMSX-BASIC]

Function: Enables or disables the audio output. If the parameter is omitted, it just reverses the state.

**MWP** (command, MSX turbo R model FS-A1GT)

Format: CALL MWP

Function: Calls the menu for programs in ROM on the MSX turbo R model FS-A1GT. For FS-A1ST, use CALL HIRO.

**NET** (command, GR8NET-BASIC)

Format: CALL NET

Function: Displays GR8NET help.

**NETBITOV** (command, GR8NET-BASIC)

Format: CALL NETBITOV (<page>, <address>, <VRAM bank>, <VRAM address>)

Function: Transfer the image of the GR8NET buffer icon to VRAM.

<page> is the logical page number of the icon data.

<address> is the address of the icon data and can only vary from 6000H to 7FFFH.

<VRAM bank> must be 0 for 0000H~FFFFH or 1 for 10 000H~1FFFFH.

<VRAM address> is the address within the selected VRAM bank.

### **NETBLOAD** (command, GR8NET-BASIC)

Format: CALL NETBLOAD (<url>, <execution flag>, <logical page>, <GR8NET address>)

Function: Load binary file from SD card or remote web server using HTTP.

<url> is the URL string for remote access. For the first partition on the SD card, use "SDC: //".

<execution flag> indicates the action to be taken for executable files.

0 – Data will not be loaded (default value)

1 – Data will be loaded but not executed

2 – The file will be loaded and executed.

<logical page> of the GR8NET (00H to 7FH)

<GR8NET address> can vary from 6000H to 7FFFH.

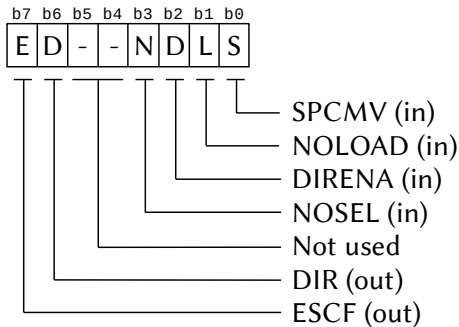
### **NETBROWSE** (command, GR8NET-BASIC)

Format: CALL NETBROWSE (<url>, <flags>)

Function: Calls the web browser and SD card.

<url> is the starting URL string.

<flags> is a one-byte value with the following meanings:



SPCMV: if this bit is set, when the user presses the SPACE key in the selection, the url will be loaded into the GR8NET RAM, but no action will be performed and the browser exits;

**NOLOAD:** if this bit is set, the browser will not load the selected url in the GR8NET RAM.

**DIRENA:** if this bit is set, when pressing the space bar in the directory it will be selected and the browser will be closed; if this bit is reset, pressing space in the directory will load the content and navigation will continue.

**NOSEL:** if set, it does not force the source device selection page (Internet / SD card), and navigation proceeds directly to the device identified by the URL string.

**DIR:** this bit returns set if the content is a directory entry or a page with a list of directories generated by the WEB server.

**ESCF:** this bit returns set when the browser is closed with the ESC key.

#### **NETBTOV** (command, GR8NET-BASIC)

Format: CALL NETBTOV (<VRAM bank>, <offset address>)

Function: Moves binary data from the GR8NET buffer to the VRAM. <VRAM bank> must be 0 for 0000H~FFFFH or 1 for 10000H~1FFFFH.

<address offset> is the offset of the address specified in the header of the binary file.

#### **NETCDTOF** (command, GR8NET-BASIC)

Format: CALL NETCDTOF

Function: Copy the DHCP configuration to the fixed IP address configuration.

#### **NETCFG** (command, GR8NET-BASIC)

Format: CALL NETCFG

Function: Activates the interactive configuration of the GR8NET.

#### **NETCODE** (function, GR8NET-BASIC)

Format: CALL NETCODE (<error\_code>, [<http\_oper>])

Function: Returns the status of the last operation and the HTTP response code.

**NETDHCP** (command, GR8NET-BASIC)

Format: CALL NETDHCP

Function: Performs DHCP search and makes its dynamic configuration.

**NETDIAG** (command, GR8NET-BASIC)

Format: CALL NETDIAG (&lt;V&gt;)

Function: Turns on / off diagnostic mode. If &lt;V&gt; is 0, it turns off; otherwise it turns on.

**NETDNS** (command, GR8NET-BASIC)

Format: CALL NETDNS [ ( [&lt;A&gt;], [&lt;B&gt;], [&lt;C&gt;], [&lt;D&gt;] ) ]

Function: Gets the IP of the current DNS domain. If there are no arguments, print the address on the screen.

**NETDUMP** (command, GR8NET-BASIC)

Format: CALL NETDHCP

Function: Performs DHCP search and makes its dynamic configuration.

**NETEND** (command, Network-BASIC)

Format: CALL NETEND

Function: Disables the MSX network (MSX Network). Short version: \_NETE.

**NETEXPRT** (command, GR8NET-BASIC)

Format: CALL NETEXPRT

Function: Create BASIC program containing GR8NET config data.

**NETFIX** (command, GR8NET-BASIC)

Format: CALL NETFIX

Function: Configure fixed IP address information for the network.

**NETFKOPLLR** (command, GR8NET-BASIC)

Format: CALL NETFKOPLLR

Function: Load the OPLL ROM (MSX-Music) into the mapped memory. This command is intended for software that runs in GR8NET mapper modes 1 to 6 (game mapper) and cannot be run in mode 8 when MSX-Music ROM is available in GR8NET subslot 3.



**NETFWUPDATE** (command, GR8NET-BASIC)

Format: CALL NETFWUPDATE ([<argument>])

Function: Updates the GR8NET firmware. If <argument> is omitted or is 0, it only displays information about the current firmware. If it is 1 (one), it updates only the main firmware and if it is 3 (three) it also updates the configuration area.

**NETGETCLK** (function, GR8NET-BASIC)

Format: CALL NETGETCLK ([<source>],<frequency>)

Function: Returns the clock frequency. If <source> returns zero, the frequency will be that of the MSX main bus; if it is different from zero, the frequency of the GR8NET internal oscillator will return. <frequency> must be a numeric variable.

**NETGETCLOUD** (command, GR8NET-BASIC)

Format: CALL NETGETCLOUD

Function: Prints the status of the GR8cloud virtual volume on the screen.

**NETGETDA** (function, GR8NET-BASIC)

Format: CALL NETGETDA (<adapter number>, <active adapters>)

Function: Returns the number of the standard adapter in <adapter number> and the list of active adapters in <active adapters>. <adapter number> must be a numeric variable.  
<active adapters> must be a numeric variable where bits 0 to 3 will receive the state of the adapters (the respective bit will be set if the device is active).

**NETGETDNS** (command, GR8NET-BASIC)

Format: CALL NETGETDNS ([<A>], [<B>], [<C>], [<D>])

Function: Gets the DNS address of the fixed IP. [<A>], [<B>], [<C>] and [<D>] must be numeric variables.

**NETGETGW** (function, GR8NET-BASIC)

Format: CALL NETGETGW ([<A>], [<B>], [<C>], [<D>])

Function: Get fixed IP address of the gateway. [<A>], [<B>], [<C>] and [<D>] must be numeric variables.

**NETGETHOST** (command, GR8NET-BASIC)

Format: CALL NETGETHOST (<flag>, <name> | <A>, <B>, <C>, <D>)

Function: Gets the name and IP address of the remote host. <A>, <B>, <C> and <D> and <flag> must be numeric variables and <name> must be an alphanumeric variable.

**NETGETIP** (function, GR8NET-BASIC)

Format: CALL NETGETIP ([<A>], [<B>], [<C>], [<D>])

Function: Gets the fixed IP address. [<A>], [<B>], [<C>] and [<D>] must be numeric variables.

**NETGETMAP** (function, GR8NET-BASIC)

Format: CALL NETGETMAP (<flags>)

Function: Gets the type of Memory Mapper and other data. <flags> must be a 16-bit numeric variable, where bits 0 to 7 contain the current logical page of the Memory Mapper and bits 8, 13 and 4 are bits of the system mode register.

**NETGETMASK** (function, GR8NET-BASIC)

Format: CALL NETGETMASK ([<A>], [<B>], [<C>], [<D>])

Function: Obtains the fixed IP address mask. [<A>], [<B>], [<C>] and [<D>] must be numeric variables.

**NETGETMD** (function, GR8NET-BASIC)

Format: CALL NETGETMD (<logical page>, <address>, variable)

Function: Get a 4-byte (32-bit) word from memory, convert and store it in a BASIC variable.

<logical page> is the number of the logical page in bank 1 of the GR8NET (6000-7FFF).

<address> is the address visible by the Z80

<variable> is a BASIC variable capable of accommodating the read value (single or double precision).

**NETGETMEM** (function, GR8NET-BASIC)

Format: CALL NETGETMEM (<logical page>, <address>, [<A>],  
[<B>], [<C>], [<D>])

Function: Reads a sequence of 4 bytes in memory.

<logic page> logic page number in bank 1 of the GR8NET (6000H~7FFFH).

<address> is the memory address visible to the Z80.

<A> = Address, <B> = Address + 1, etc.

**NETGETMIX** (function, GR8NET-BASIC)

Format: CALL NETGETMIX ([<number>])

Function: Returns the configuration of the audio mixer. If bit 15 is 0, the audio is mono, if it is 1 it is stereo. If <number> is omitted, the setting will be printed on the screen.

<number> is a 16-bit numeric value:

| b15 | b14 | b13 | b12 | b11 | b10   | b9   | b8   | b7  | b6  | b5 | b4 | b3 | b2 | b1 | b0 |
|-----|-----|-----|-----|-----|-------|------|------|-----|-----|----|----|----|----|----|----|
| S   | 0   | 0   | 0   | PSG | Y8950 | OPLL | Wave | SCC | PCM |    |    |    |    |    |    |

Where each 2 bits represent the following:

00 – Mute                      10 – Right channel

01 – Left channel      11 – Both channels

**NETGETMMV** (function, GR8NET-BASIC)

Format: CALL NETGETMMV ([<user home page>], [<top of RAM>]  
[<disk image start page>], [maximum no. of pages],  
[<init page of Y8950 RAM>])

Function: Returns the configuration of the memory manager. All are numeric values and any can be omitted. If all are omitted, the command prints the values on the screen.

**NETGETNAME** (function, GR8NET-BASIC)

Format: CALL NETGETNAME ([<filename>])

Function: Returns the filename of the remote resource.

**NETGETNTP** (function, GR8NET-BASIC)

Format: CALL NETGETNTP ([<A>], [<B>], [<C>], [<D>])

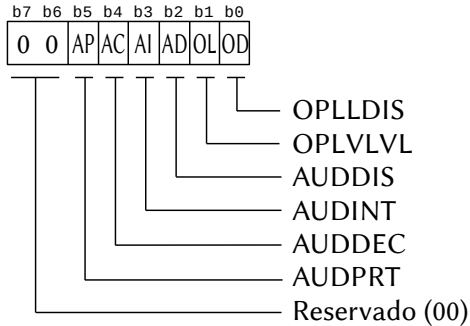
Function: Gets the properties on the NTP server in the fixed IP address configuration. [<A>], [<B>], [<C>] and [<D>] must be numeric variables.

**NETGETOPL** (function, GR8NET-BASIC)

Format: CALL NETGETOPL (<OPL state>, <num sample RAM  
pages>, <sample RAM size>)

Function: Gets the status of the OPLL / Y8950 and the size of the Sample RAM.

<OPL state> is a byte of flags with the following format:



- OPLLDIS – 0 – OPLL output is active (default)  
1 – OPLL output turned off
- OPLVLVL – 0 – Normal volume OPLL / Y8950 (default)  
1 – Duplicated volume OPLL / Y8950
- AUDDIS – 0 – MSX-Audio is enabled (default)  
1 – MSX-Audio disabled
- AUDINT – 0 – Interrupt. MSX-Audio enabled (default)  
1 – MSX-Audio interrupts disabled
- AUDDEC – 0 – MSX-Audio unconfigured / unavailable  
1 – MSX-Audio configured in AUDPRT
- AUDPRT – 0 – MSX-Audio configured port C0-C1  
1 – MSX-Audio configured port C2-C3
- <num sample RAM pages> allocated (8K each)  
<sample RAM size> required for 8K pages

### **NETGETPATH** (function, GR8NET-BASIC)

Format: CALL NETGETPATH ([<path>])

Function: Returns the <path> of the remote resource. <path> must be an alphanumeric variable. If omitted, print the path on the screen.

### **NETGETPORT** (function, GR8NET-BASIC)

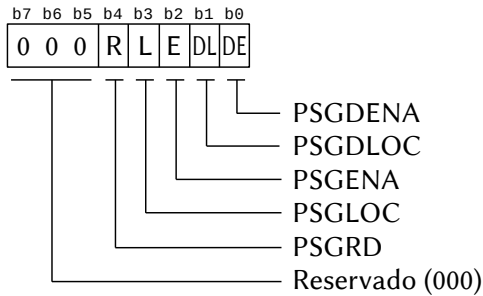
Format: CALL NETGETPATH ([<remote port>], [<local port>])

Function: Returns the <remote port> and <local port> (must be numeric variables).

### **NETGETPSG** (function, GR8NET-BASIC)

Format: CALL NETGETPSG ([<flags>])

Function: Returns some data from the PSG. <flags> is a data byte with the following format:



Where:

- PSGDENA and PSGDLOC are the desired initial state of PSG (see \_NETSETPSG);
- PSGENA and PSGLOC are the real state (1 = Activated) and the location (0 = 0xA0, 1 = 0x10) of the PSG;
- If PSGRD is 1, the PSG registers can be read and if it is 0 the PSG is in write-only mode.

### **NETGETQSTR** (function, GR8NET-BASIC)

Format: CALL NETGETQSTR ([<query string>])

Function: Returns the <query string> defined for the remote resource. If the argument is omitted, print the result on the screen.

### **NETGETTSHN** (function, GR8NET-BASIC)

Format: CALL NETGETTSHN ([<time server>])

Function: Returns the host name of the <time server>. If the argument is omitted, print the result on the screen.

### **NETGW** (function, GR8NET-BASIC)

Format: CALL NETGW ([<A>], [<B>], [<C>], [<D>])

Function: Returns the configuration of the current gateway. <A>, <B>, <C> and <D> must be numeric variables.

### **NETHELP** (command, GR8NET-BASIC)

Format: CALL NETHELP <command>

Function: Displays help for the specified GR8NET <command>. If <command> is omitted, print a list of all available commands.

**NETIMPRT** (command, GR8NET-BASIC)

Format: CALL NETIMPRT

Function: Fill the GR8NET system variables with data from the BASIC program created by NETEXPRT.

**NETINIT** (command, Network-BASIC)

Format: CALL NETINIT

Function: Initializes the MSX network. Use only after CALL NETEND, as the network is automatically started when the computer is turned on. Short version: \_NETI

**NETIP** (function, GR8NET-BASIC)

Format: CALL NETIP ([<A>], [<B>], [<C>], [<D>])

Function: Gets the IP address of the current adapter. <A>, <B>, <C> and <D> must be numeric variables.

**NETLDBUF** (command, GR8NET-BASIC)

Format: CALL NETLDBUF (<adapter page>, <adapter address>, <block size>, <RAM address>, [<mapper type>])

Function: Copy data from main memory to the adapter's buffer.

**NETLDRAM** (command, GR8NET-BASIC)

Format: CALL NETLDRAM (<adapter page>, <adapter address>, <block size>, <RAM address>)

Function: Downloads data from the adapter's buffer to main memory. <adapter address> must be between &H6000 and &H7FFF.

**NETMASK** (function, GR8NET-BASIC)

Format: CALL NETMASK ([<A>], [<B>], [<C>], [<D>])

Function: Gets the subnet mask of the current adapter. <A>, <B>, <C> and <D> must be numeric variables.

**NETNTP** (function, GR8NET-BASIC)

Format: CALL NETNTP (<A>, <B>, <C>, <D>, <TZF>)

Function: Obtains the effective configuration of the NTP server. <A>, <B>, <C>, <D> and <TZF> must be variables or constants numeric. If bit 7 of TZF is set, the RTC will be synchronized with the NTP server. Bits 0 to 6 represent a positive or negative value (-64 [40H] to +63 [3FH]) and define the time zone in 15-minute increments.

**NETPLAYBUF[#]A** (command, GR8NET-BASIC)

Format: CALL NETPLAYBUF[#]A (<logical page>, <address>, <size>)

Function: Defines the address and initial size of buffer No. [#] for the PCM. <logical page> must be between 00H~7FH, <address> between 6000H~7FFFH and <size> must be calculated so as not to exceed the GR8NET memory of 1 MB. [#] must be between 0 and 9.

**NETPLAYBUF[#]C** (command, GR8NET-BASIC)

Format: CALL NETPLAYBUF[#]C

Function: Continue playback by refilling PCM buffer [#], which must be between 0 and 9.

**NETPLAYBUF[#]P** (command, GR8NET-BASIC)

Format: CALL NETPLAYBUF[#]P (<size>, <frequency>)

Function: Start the reproduction of the data pre-stored in the PCM buffer n° [#]. <size> can be 8 or 16 bits and <frequency> can range from 1 to 65,536. [#] must be between 0 and 9. To prevent the buffer from emptying, the command \_NETPLAYBUF[#]C must be used.

**NETPLAYBUF[#]R** (command, GR8NET-BASIC)

Format: CALL NETPLAYBUF[#]R

Function: Reset the buffer [#] reproduction mechanism, which must be between 0 and 9.

**NETPLAYBUF[#]S** (command, GR8NET-BASIC)

Format: CALL NETPLAYBUF[#]S (<state>)

Function: Return the playback status of buffer [#]. <state> must be a numeric variable. If <state> returns -1, playback has ended and if it returns 0, data is still being played. [#] must be between 0 and 9.

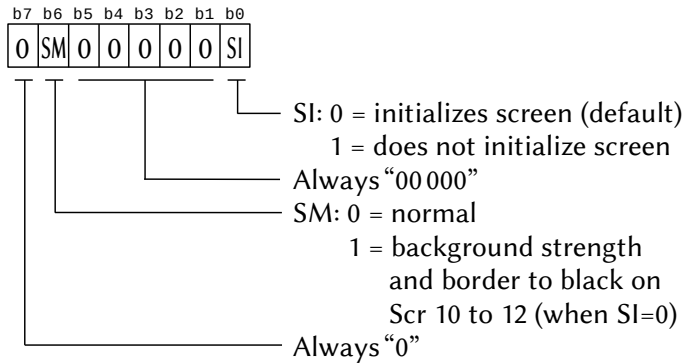
**NETPLAYVID** (command, GR8NET-BASIC)

Format: CALL NETPLAYVID (<path> [,<flags>])

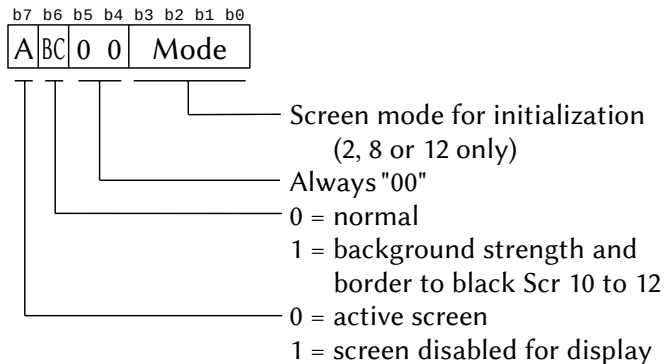
CALL NETPLAYVID (<screen mode>)

Function: Play video from the SD card. It works in two ways, depending on the first argument. If string, specify the

<path> of the video file on the SD card. <flags> is an 8-bit value whose meanings are described below:



If the first argument is an integer, its lowest 8 bits are flags with the following meanings:



### **NETPLAYWAV** (command, GR8NET-BASIC)

Format: CALL NETPLAYWAV (<path>)

Function: Play audio in wave format. <path> is a name or string variable that identifies the location of the URI of the remote wave file.

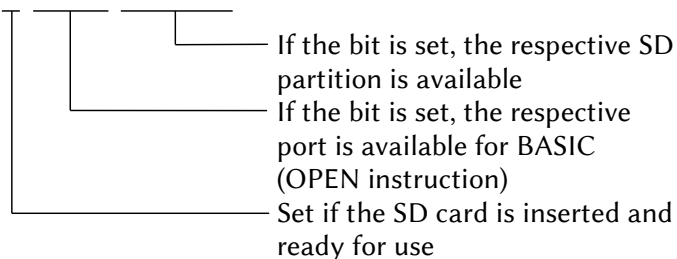
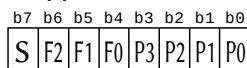
### **NETRESST** (command, GR8NET-BASIC)

Format: CALL NETRESST (<flags>)

CALL NETRESST (<inbound URI>, <outbound URI>,  
<flags>, <size>)



Function: Return the resource's state. If the first argument is an integer variable, the lowest 8 bits will contain the flags as described below. If it is alphanumeric, <input URI> and <output URI> will contain the respective paths and <size> is a numeric variable that returns the size of the resource. <flags> is an integer variable whose lowest 8 bits are mapped as follows:



**NETSAVE** (command, GR8NET-BASIC)

Format: CALL NETSAVE

Function: Save the current configuration of the ROM configuration page.

**NETSDCRD** (command, GR8NET-BASIC)

Format: CALL NETSDCRD (<logical page>, <address>, <sector>, <number of sectors to read>)

Function: Read sectors from the SD card. <logical page> is the page number in bank 1 of the GR8NET (6000H~7FFFH), <address> is the visible address for the Z80 and <sector> is the number of the first sector to be read.

**NETSETCLK** (command, GR8NET-BASIC)

Format: CALL NETSETCLK (<source>)

Function: Defines the frequency source for speed measurement. If <source> is 0, the frequency of the MSX internal bus will be used; if different from 0, the frequency of the GR8NET internal oscillator (3.579545 MHz) will be used.

**NETSETCLOUD** (command, GR8NET-BASIC)

Format: CALL NETSETCLOUD (<hostname: port>, <password>)  
CALL NETSETCLOUD (<activation flag>)

Function: Configure access to the GR8NET virtual volume.  
<hostname: port> can be up to 70 characters long, with the port number separated by a colon. The access <password> can be up to 16 characters. To enable the GR8cloud subsystem, <activation flag> must contain the numeric value 1, but the volume will only be fully accessible after the restart.

**NETSETDA** (command, GR8NET-BASIC)

Format: CALL NETSETDA (<adapter number>)

Function: Defines the number of the standard adapter.  
<adapter number> must be a value from 0 to 3.

**NETSETDM** (command, GR8NET-BASIC)

Format: CALL NETSETDM (<logical page>, <address>, <variable>)

Function: Gets the value of a BASIC variable, converts it to a 32-bit value and stores it in memory.  
<logic page> logic page number in bank 1 of the GR8NET (6000H~7FFFH).  
<address> is the memory address visible to the Z80.  
<variable> can be a number, an expression or a numeric variable of any type.

**NETSETDNS** (command, GR8NET-BASIC)

Format: CALL NETSETDNS ([<A>], [<B>], [<C>], [<D>])

Function: Defines fixed IP address. At least one of the <A>, <B>, <C> or <D> values must be defined.

**NETSETGW** (command, GR8NET-BASIC)

Format: CALL NETSETGW ([<A>], [<B>], [<C>], [<D>])

Function: Defines the fixed IP address of the gateway. At least one of the <A>, <B>, <C> or <D> values must be defined.

**NETSETHOST** (command, GR8NET-BASIC)

Format: CALL NETSETHOST (<URI>)  
CALL NETSETHOST (<A>, <B>, <C>, <D>)

Function: Defines the name of the remote host and, if necessary, performs a simple DNS query. The <URI> must be written without a protocol definition and without the final slash (eg “www.gr8bit.ru”)

**NETSETIP** (command, GR8NET-BASIC)

Format: CALL NETSETIP ([<A>], [<B>], [<C>], [<D>])

Function: Defines fixed IP address. At least one of the <A>, <B>, <C> or <D> values must be defined.

**NETSETMAP** (command, GR8NET-BASIC)

Format: CALL NETSETMAP [<A>, <M>, <MRPD>]

Function: Defines the type of Memory Mapper and restarts the system.

<A> identifies the type of memory mapped and the location of the special register set.

<M> 0 – Reading disabled.

1 – Reading enabled.

2 – Automatic detection (default)

<MRPD> RAM mapped with pending disable bit  
(0 – Enable; 1 – Disable).

**NETSETMASK** (command, GR8NET-BASIC)

Format: CALL NETSETMASK ([<A>], [<B>], [<C>], [<D>])

Function: Sets the mask for the fixed IP address. At least one of the <A>, <B>, <C> or <D> values must be defined.

**NETSETMEM** (command, GR8NET-BASIC)

Format: CALL NETSETMEM (<logical page>, <address>, [<A>], [<B>], [<C>], [<D>])

Function: Writes a sequence of 4 bytes in the memory.

<logic page> logic page number in bank 1 of the GR8NET (6000H~7FFFH).

<address> is the memory address visible to the Z80.

<A> = Address, <B> = Address + 1, etc.

**NETSETMIX** (command, GR8NET-BASIC)

Format: CALL NETSETMIX (<number>)

CALL NETSETMIX (<string>)

Function: Configures the audio mixer.

<number> – 16-bit numeric value with following format:

|     |     |     |     |     |       |      |      |     |     |    |    |    |    |    |    |
|-----|-----|-----|-----|-----|-------|------|------|-----|-----|----|----|----|----|----|----|
| b15 | b14 | b13 | b12 | b11 | b10   | b9   | b8   | b7  | b6  | b5 | b4 | b3 | b2 | b1 | b0 |
| 0   | 0   | 0   | 0   | PSG | Y8950 | OPLL | Wave | SCC | PCM |    |    |    |    |    |    |

Each 2 bits represent the following:

00 – Mute

10 – Right channel

01 – Left channel

11 – Both channels

If it is a 6-character string, each char means:

M – Mute

R – Right channel

L – Left channel

B – Both channels

Another character, the setting will be preserved.

**NETSETMMV** (command, GR8NET-BASIC)

Format: CALL NETSETMMV (<numeric variable>)

**Function:** Defines the value of the memory manager. You can manage only the home page of RAM protected by the user.

## NETSETNAME (command, GR8NET-BASIC)

Format: CALL NETSETMMV (<filename>)

Function: Defines the filename of the remote resource. The maximum length of the filename is 63 characters.

**NETSETNTP** (command, GR8NET-BASIC)

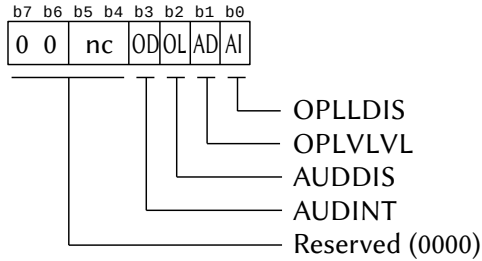
Format: CALL NETSETNTP (<A>, <B>, <C>, <D>, <TZF>)

Function: Defines the properties of the NTP server within the configuration of fixed IP and time setting flags. <A>, <B>, <C> and <D> define the IP address of the NTP server and <TZF> is the time zone update flag (see NETNTP command).

### NETSETOPL (command, GR8NET-BASIC)

Format: CALL NETSETOPL (<flags>, <memory size>)

Function: Enables or disables the OPLL/Y8950, controls the doubling of the output amplitude and defines the size of the Y8950's audio memory. <memory size> defines the size of the audio memory in 8 Kbyte increments, the maximum and default value being 32 (32 \* 8 = 256K). <flags> is a 1-byte value whose structure is described in the next page.



- OPLLDIS – 0 – Enable OPLL output  
 1 – Disable OPLL output
- OPLVLVL – 0 – Normal volume OPLL / Y8950 (default)  
 1 – Duplicated volume OPLL / Y8950
- AUDDIS – 0 – Enables MSX-Audio (default)  
 1 – Disable MSX-Audio
- AUDINT – 0 – Enables interrupt. MSX-Audio (default)  
 1 – Disable MSX-Audio interrupts

### **NETSETPATH** (command, GR8NET-BASIC)

Format: CALL NETSETPATH (<path>)

Function: Defines the path of the remote resource. <path> is a name or string variable with a maximum length of 239 characters and no trailing bar and represents the absolute value.

### **NETSETPORT** (command, GR8NET-BASIC)

Format: CALL NETSETPORT (<remote port>, <local port>)

Function: Defines the communication port numbers in the standard URI structure. If <local port> is 0, dynamic port number (default value) will be used. This command does not check the validity of the ports.

### **NETSETPSG** (command, GR8NET-BASIC)

Format: CALL NETSETPSG (<value>)

Function: Configure the PSG.

<value> is a variable or bitmap constant, where bit set 0 defines whether PSG should be activated in (re) configuration and bit set 1 designates the port location to 0x10 if, on reset, the port is 0xA0 (built-in mirrored PSG). If the argument is omitted, the PSG will be reconfigured.

**NETSETQSTR** (command, GR8NET-BASIC)

Format: CALL NETSETQSTR (<parameter>)

Function: Defines the sequence of queries for processing remote resources. <parameter> is a variable or string constant that must start with the character “?” and have a maximum of 63 characters.

**NETSETTSHN** (command, GR8NET-BASIC)

Format: CALL NETSETTSHN (<name>)

Function: Defines the name of the time server. <name> is a variable or string constant that must have a maximum of 63 chars.

**NETSNDDTG** (command, GR8NET-BASIC)

Format: CALL NETSNDDTG (<file number> [, <A>, <B>, <C>, <D>]  
[ , <RP>] )

Function: Sends pending datagram / data to the remote host.  
<file number> is the BASIC file number. <A>, <B>, <C> and <D> represent the IP address of the remote device (can be omitted). RP is the port number of the remote device and can also be omitted.

**NETSNDVOL** (command, GR8NET-BASIC)

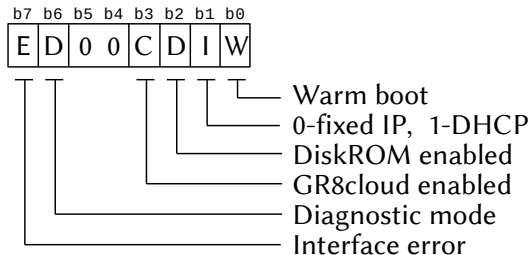
Format: CALL NETSNDVOL (<principal>, <SCC>, <waveform>,  
<PCM>, <OPLL>, <Y8950>, <PSG>)

Function: Read or change the volume of the audio generators. All arguments must be in the range 0 (mute) to 128 (maximum volume) and anyone can be omitted.

**NETSTAT** (command, GR8NET-BASIC)

Format: CALL NETSTAT (<Mode:>)

Function: Displays adapter status information. “Mode:” is a one-byte value with the following meaning:



**NETSYSINFO** (command, GR8NET-BASIC)

Format: CALL NETSYSINFO (<MSX version>, <clock frequency>, <T cycle performance>, <VDP version>, <vertical rate / VRAM size>)

Function: Returns system performance information and data.

<MSX version> – 0=MSX1; 1=MSX2; 3=MSX2+, 4 = MSX TR.

For MSX turbo R, bit5 = 0 → R800; bit5 = 1 → Z80

and bit6 = 0 → DRAM mode; bit6 = 1 → ROM mode

<clock frequency> returns the slot clock (3579560)

<T cycle performance> returns the total number of times a 51 T cycle instruction (plus 8 of the M1 cycle) is executed in one second ( $60671 * (51 + 8)$ )

<VDP version> – 0 = TMS; 1 = V9938; 2 = V9958

<vertical rate / VRAM size> – Value of two bytes, where the lowest byte returns the frame rate (0 = 60 Hz, 1 = 50 Hz, 255 = error) and the highest byte returns the size of the block VRAM (1 = 8K; 2 = 16K; 4 = 32K; 8 = 64K; 16 = 128K; 255 = error).

**NETRCHKS** (command, GR8NET-BASIC)

Format: CALL NETRCHKS (<data block>, <address>, <number of bytes> [<, checksum>])

Function: Calculates the 16-bit checksum of the contents of the RAM buffer of <data block> in bank 1 of the GR8NET. If the <checksum> variable is provided, it will receive the checksum, otherwise, the sum will be printed on the screen.

**NETTELNET** (command, GR8NET-BASIC)

Format: CALL NETTELNET ([<url | IP: port>], <signal>)

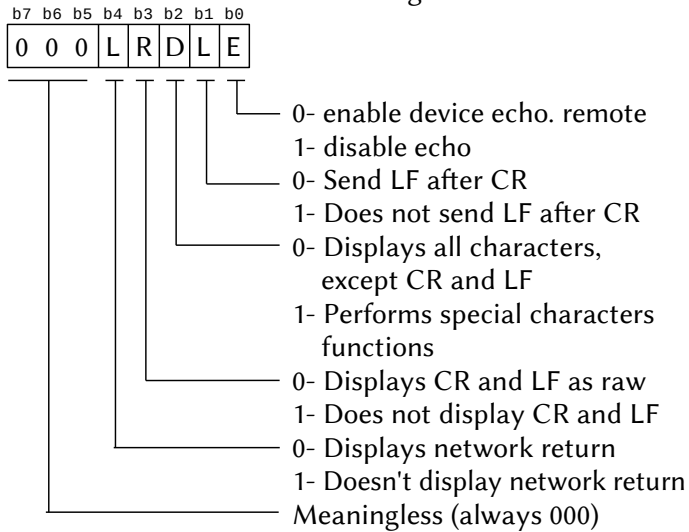
Function: Run telnet session using TCP. <signal> is a one-byte value where only bit 1 has meaning. If it is 1, it means that the telnet application does not add the character LF after the CR; if it is 0, pressing RETURN will send CR + LF to the remote host.

**NETTERM** (command, GR8NET-BASIC)

Format: CALL NETTERM ([<url | IP: port>], <flags>)

Function: Run telnet session using TCP. This command does not perform special translation of ESC code (&H1B). <flags>

must be kept at 0 if the remote device echoes what it receives back to the GR8NET. The meaning of the bits is as follows:



### **NETTGTMAP** (command, GR8NET-BASIC)

Format: CALL NETTGTMAP [(**<A>**, **<M>**, **<MRPD>**)]

Function: Defines the type of Memory Mapper. Unlike NETSETMAP and that this command does not restart the machine.

**<A>** identifies the type of memory mapped and the location of the special register set.

**<M>** 0 – Reading disabled

1 – Reading enabled

2 – Automatic detection (default)

**<MRPD>** RAM mapped with pending disable bit  
(0 – Enable; 1 – Disable)

### **NETTSYNC** (command, GR8NET-BASIC)

Format: CALL NETTSYNC

Function: Displays and synchronizes the system time.

### **NETVARBRSTR** (function, GR8NET-BASIC)

Format: CALL NETVARBRSTR (**<alphanumeric variable>**)

Function: Get the URL string of the location selected by the user in the browser and store it in **<alphanumeric variable>**. If the length exceeds 254 characters, an error will be generated.



**NETVARBSIZE** (function, GR8NET-BASIC)

Format: CALL NETVARBSIZE (<numeric variable>)

Function: Gets the size of the data loaded in bytes and stores it in <numeric variable>.

**NETVARRWTH** (command, GR8NET-BASIC)

Format: CALL NETVARRWTH (<current value>, <new limit>)

Function: Define the limit of the network RX window. <current value> must be a numeric variable that receives the current size (default is 0). <new limit> can be variable or numeric constant between 0 and 2047.

**NETVARUDTO** (command, GR8NET-BASIC)

Format: CALL NETVARUDTO (<current value>, <new limit>)

Function: Set UDP packet timeout for DHCP and DNS operations. <current value> is a variable that receives the current timeout and DHCP request retry count value. <new limit> is variable or constant, setting a new timeout value and counting DHCP request retries. Bits 7~0 identify the UDP timeout value (0-255), in periods of 100 ms. The default is 20 (2s). Bits 11~8 identify the number of DHCP request attempts attempted when GR8NET is started.

**NETVER** (function, GR8NET-BASIC)

Format: CALL NETVER

Function: Displays the GR8NET firmware version in the screen.

**NEXTOR** (command, Nextor)

Format: CALL NEXTOR

Function: Displays the list of commands added by Nextor.

**NSCAN** (command, Hitachi-BASIC version 2)

Format: CALL NSCAN

Function: Causes the Hitachi MB-H2 micro's built-in data reader to search for empty parts on the tape.

**OFFHOOK** (command, New Modem BASIC)

Format: CALL OFFHOOK

Function: Lift the phone handset.

**OFFLINE** (command, Network-BASIC, SVI-Modem BASIC)

Format: CALL OFFLINE [Network-BASIC]

Function: Disconnects the computer from the network. This instruction is only available to students and must be preceded by CALL NETINIT. Short version: \_OFFL.

Format: CALL OFFLINE [SVI-Modem BASIC]

Function: Take the modem offline.

**ON EVENT (n) GOSUB** (statement, SFG-BASIC)

Format: CALL ON EVENT (<event number>) GOSUB

Function: Defines the subroutine that will be executed when a specific event occurs. <event number> can be:

1~4 – Stops when playback of the specified instrument ends.

5 – Stops when rhythm playback ends.

6 – Interrupts according to the time programmed in the FM unit timer.

If <event number> is omitted, the command will be applied to all events. The priority of the events is as follows:

|                    |                    |
|--------------------|--------------------|
| 1st – BASIC        | 5th – Instrument 4 |
| 2nd – Instrument 1 | 6th – Rhythm       |
| 3rd – Instrument 2 | 7th – Timer        |
| 4th – Instrument 3 |                    |

Short version: \_ON EVEN (<event number>) GOSUB.

**ONHOOK** (command, New Modem BASIC)

Format: CALL ONHOOK

Function: Hangs up the telephone handset.

**ONLINE** (command, Network-BASIC, SVI-Modem BASIC)

Format: CALL ONLINE [Network-BASIC]

Function: Connect the computer to the network. This instruction is only available to students. Eventually it may be necessary to run CALL NETINIT beforehand. Short version: \_ONLI.

Format: CALL ONLINE [SVI-Modem BASIC]

Function: Put the modem in online mode.

**PACLOAD** (command, DM-System2 BASIC)

Format: CALL PACLOAD (<PAC address>, [@]<destination address> [,<length>])

Function: Reads data from the PAC SRAM (Pana Amusement Cartridge).

<PAC address> is the absolute address of the PAC cartridge and can vary from 0000H to 1FFDH.

<destination address> is the address to which the scanned data will be transferred. If preceded by "@", it means VRAM.

<length> is the number of bytes read. If omitted, 1024 bytes (1 block) will be read.

**PACSAVE** (command, DM-System2 BASIC)

Format: CALL PACSAVE ([@]<starting address>, <PAC address> [,<length>])

Function: Writes data in the PAC SRAM (Pana Amusement Cartridge).

<start address> is the address from which the data will be read. If preceded by "@", it means VRAM.

<PAC address> is the absolute address of the PAC cartridge and can vary from 0000H to 1FFDH.

<length> is the number of bytes to write. If omitted, 1024 bytes (1 block) will be written.

**PALETTE** (declaration, 3, Kanji-BASIC, Hangul-BASIC, RMSX BASIC)

Format: CALL PALETTE (<palette number>, <R>, <G>, <B>)  
[MSX-BASIC version 3, Kanji-BASIC, Hangul-BASIC]

Function: Specifies the colors for the palette. <palette number> can range from 0 to 15 and "<R>, <G>, <B>" from 0 to 7. All BASIC versions have the same syntax, except RMSX-BASIC.

Format: CALL PALETTE <palette/monitor> [RMSX BASIC]

Function: Selects the palette or monitor emulation to be used on the MSX1 computer emulated on a Turbo R machine by the rMSX emulator. <palette/monitor> can be MSX1, MSX2, GREEN or GRAY.

**PAN** (statement, Pioneer-BASIC)

Format: CALL PAN (<X axis>, <volume>, <string>)

Function: Generates sound according to the parameters provided.

<X axis> – Location of the generated sound. It can range from 0 to 255, with 0 corresponding to the extreme left and 255 to the extreme right.

<volume> can range from 0 to 7.

<string> – Macrocommands identical to those of the PLAY instruction for PSG, except for V, S, M and X. The string can contain up to 79 characters.

**PATTERN** (statement, SFG-BASIC)

Format: CALL PATTERN (<standard number>, <alphanum var (n)>)

Function: Defines the patterns of the rhythms through an alpha numeric matrix of one dimension (vector). Short version: \_PATT.

<standard number> can be 7 or 8 (only two patterns can be defined).

<alphanum var (n)> points to a vector whose indices define different aspects:

xx\$ (0) defines the size:

"3" – A quarter note times 3

"4" – A quarter note times 4

"8" – A quarter note times 8

xx\$ (1) defines "close high-hat"

xx\$ (2) defines "open high hat"

xx\$ (3) defines "bass drum"

xx\$ (4) defines "high tomtom"

xx\$ (5) defines "low tomtom"

"Close high-hat" and "open high hat" cannot be played simultaneously.

The string for indices (1) to (5) must be composed of a sequence of "0" s and "1" s that represent units of 1/12 of a quarter note. The size depends on the value defined by xx\$ (0):

xx\$ (0) = "3" → 36 characters

xx\$ (0) = "4" → 48 characters

xx\$ (0) = "8" → 96 characters

**PAUSE** (command, MSX-BASIC 4, ChakkariCopy BASIC,  
DM- System2 BASIC, Hitachi BASIC 2)

Format: CALL PAUSE (<time>)  
[MSX-BASIC 4] [DM-System2 BASIC]

Function: Pauses the execution of the program in BASIC. <time> is specified in milliseconds and can range from 0 to 65,535. It can be aborted by CTRL+STOP.

Format: CALL PAUSE [ChakkariCopy BASIC]

Function: Put the Chakkari Copy cartridge in pause mode.

Format: CALL PAUSE [Hitachi BASIC 2]

Function: Put the internal data reader of the Hitachi MB-H2 micro in pause mode.

**PCM FREQ** (command, MSX-Audio)

Format: CALL PCM FREQ (<frequency>)

Function: Defines the sampling frequency for ADPCM. <frequency> can range from 1,800 to 49,716 Hz.

**PCM VOL** (controller, MSX-Audio)

Format: CALL PCM VOL (<volume>)

Function: Sets the reproduction volume for ADPCM and PCM. <volume> can range from 0 to 63. Initial values are 55 for ADPCM and 32 for PCM.

**PCMON** (declaration, DM-System2 BASIC)

Format: CALL PCMON ([@]<starting address>, [@]<ending  
address>, <rate> [,<loop>])

Function: Plays data through PCM on MSX2 onwards. Requires PCM driver.

<start address> of the data to be reproduced. If preceded by "@", it means VRAM.

<final address> of the data to be reproduced. If preceded by "@", it means VRAM.

<rate> can be: 0 → 15.75 Khz      2 → 5.25 KHz

1 → 7.875 Khz      3 → 3.9375 KHz

<loop> defines the number of times the data will be played. It can vary from 1 to 255. 0 = Infinite loop.

**PCMPPLAY** (declaration, 4)

**Format:** CALL PCMPLAY (@<start address>, <end address>, <sampling rate> [,S])

Function: Plays PCM data stored in RAM or VRAM.  
 <sampling rate> can be 0 to 3.  
 <start address> and <end address> are the starting and  
 ending addresses.  
 [,S] specifies VRAM.

**PCMREC** (command, 4)

**Format:** CALL PCMREC (@ <start address>, <end address>,  
                                <sampling rate>, [[<trigger level>], [<save>], S))

Function: Writes PCM data to RAM or VRAM.  
 <start address> and <end address> can range from 0000H to FFFFH,  
 <sampling rate> can vary from 0 to 3,  
 <trigger level> can vary from 0 to 127,  
 <save> can be 0 (unsaved) or 1 (saves in RAM),  
 [S] records in the VRAM.

**PDIAL** (remote, New Modem BASIC, SVI Modem BASIC)

Format: CALL PDIAL (<string variable>,<numeric variable>  
[New Modem BASIC]

Function: Calls a specific phone number via pulse dialing. <string variable> stores the phone number to be called, where only numbers are allowed and the character “-”, which add 1 second wait. <numeric variable> returns the state: if it is 0, the input value is correct; if it is "1" it is not.

Format: CALL PDIAL (“<phone number>”) [SVI Modem BASIC]

Function: Calls a specific phone number via pulse dialing. <phone number> must be in quotes and only numeric characters are allowed.

**PEEK** (function, DM-System2 BASIC, Network-BASIC)

Format: CALL PEEK ([@]<address> [,<variable>])  
[DM-System2 BASIC]

Function: Reads a byte from any area of Main RAM, VRAM or Memory Mapper.

<address> – Is the address to be read. If greater than 65534, the Memory Mapper specifications will be used. If it is preceded by “@”, it indicates VRAM.

<variable> is a numeric variable that will receive the value of the byte read. If omitted, the value will be displayed on the screen.

Format: CALL PEEK (<variable>, <address>[, <student number>]  
[,<N>] ) [Network-BASIC]

Function: Reads a byte of data from NetRAM (student or teacher) or NetRAM / RAM (teacher only).

<variable> receives the value of the read byte.

<address> must be between 7800H and 7FFFH for NetRAM.

<student number> is a number between 1 and 15. Only the teacher can use this parameter.

<N> must be used to read an address in NetRAM. Useful only for the teacher.

## **PEEKs** (function, DM-System2 BASIC)

Format: CALL PEEKs ([@]<address>, <size>, <string variable>)

Function: Reads several consecutive bytes in Main RAM, VRAM or Memory Mapper, converts them to characters and stores them in a string variable.

<address> – Is the address to be read. If greater than 65534, the Memory Mapper specifications will be used. If it is preceded by “@”, it indicates VRAM.

<size> – Number of bytes to be read, ranging from 1 to 255.

<string variable> receives the bytes read.

## **PEEKw** (function, DM-System2 BASIC)

Format: CALL PEEKw ([@]<address> [,<variable>])

Function: Reads two consecutive bytes in Main RAM, VRAM or Memory Mapper.

<address> – Is the address to be read. If greater than 65534, the Memory Mapper specifications will be used. If it is preceded by “@”, it indicates VRAM.

<variable> receives the value read. If not specified, the value read will be displayed on the screen in hex format.

**PHRASE** (macro-declaration, SFG-BASIC)

Format: CALL PHRASE (<track number>, <playlist> [,<brand>])

Function: Writes the audio playback data to the specified track.

Short version: \_PHRA.

<track number> can vary from 1 to the value defined by \_PLAY.

<brand> is a number that can vary from 1 to 254. If omitted, it will be considered equal to <track number>.

<playlist> contains the music macros.

A~G Plays an encrypted note with duration n (1~64, pattern 4).

# or + Sustain.

– Flat.

! Returns the note to its original value (K and S commands).

On Octave (n → 1 to 8; the default is 3).

Nn Pitch (n → 25 to 120).

Ln Note length (n → 1 to 64, default: 4).

. Duration increased by 50%.

Rn Pause of duration n (n → 1 to 64, default is 4).

Wn Note duration in 1/96 units (n: 1 to 96).

Tn Time (n → 1 to 200, specified by \_TEMPO).

Vn Volume (n → 1 to 100, the default is 50)

& Ligature

Mn Period in units of n / 4 (n → 3 to 8)

// The string between two bars will be considered a block of duration specified by Mn.

Sn “n” specifies the number of “Sharps” (#).

Kn “n” specifies the number of “Flats” (b).

% n “Staccato” and “tenuto”. “n” specifies the time proportion of the note will be played (0% to 100%).

[ ] Wake up. Comma-separated strings inside the bracket are played simultaneously.

{ }n Define the notes between {} in n. (n = 1~64, default is Ln)

**PITCH** (declaration, MSX-Music)

Format: CALL PITCH (<n>)

Function: Fine adjustment of the sound. <n> can range from 410 to 459, the default value being 440 (central “La” tone).



**PLAY** (macro-declaration, MSX-Music/Audio, Hitachi-BASIC, SFG-BASIC)

Format: CALL PLAY (<n>, <numeric variable>) [Music /Audio]

Function: Returns in the <numeric variable> the state of the voice <n> of the OPLL (touching [-1] or not [0]). <n> can range from 0 to 9. If 0, all voices are checked. 1 to 9 checks the respective voice.

Format: CALL PLAY [Hitachi-BASIC 2]

Function: Puts the internal data reader of the Hitachi MB-H2 micro in playback mode. This instruction does not support files in ASCII mode (BASIC or data).

Format: CALL PLAY (<instrument>, <range> [,<brand>]) [SFG-BASIC]

Function: Play the previously written song with the CALL PHRASE instruction.

<instrument> is a number from 1 to 4

<range> is a number from 1 to 9, with 2 to 8 having to be previously defined by the CALL TRACK instruction and 9 when reproducing via the musical keyboard.

<brand> is a number between 1 and 255 to specify the CALL PHRASE tag used for reproduction. If omitted, the same <track> number is considered.

**PLAY MK** (statement, MSX-Audio)

Format: CALL PLAY MK (<matrix name>)  
CALL PLAY MK (<start address>, <end address>)  
CALL PLAY MK (A), where the sequence A must be previously declared in the DIM and REC MK instructions.

Function: Plays file recorded by the musical keyboard.

**PLAY PCM** (statement, MSX-Audio)

Format: CALL PLAY PCM (<file number>, <offset>, <size>, <sampling frequency>)

Function: Plays an audio file via PCM /ADPCM.

<file number> – Audio file number (0 to 15).

<offset> – Offset in units of 256 bytes.

<size> – Size in bytes of the audio file.

<sampling frequency> – Can range from 1,800 to 49,716 Hz for ADPCM and from 1,800 to 16,000 Hz for PCM.

- POKE** (statement, DM-System2 BASIC, Network-BASIC)  
 Format: CALL POKE ([@]<address>, <value>) [DM-System2 BASIC]  
 Function: Writes a byte in any area of Main RAM, VRAM or Memory Mapper.  
 <address> – Is the address to be read. If greater than 65 534, the Memory Mapper specifications will be used. If it is preceded by “@”, it indicates VRAM.  
 <value> is the value to be written. It must be in decimal between 0 and 255, it can also be an expression.
- Format: CALL POKE (<value>, <address> [,<student number>] [,<N>]) [Network-BASIC]  
 Function: Writes a byte of data in NetRAM (student or teacher) or NetRAM / RAM (teacher only).  
 <value> must be a decimal number between 0 and 255.  
 <address> must be between 7800H and 7FFFH for NetRAM.  
 <student number> is a number between 1 and 15. Only the teacher can use this parameter.  
 <N> must be used to write an address on NetRAM. Useful only for the teacher.
- POKES** (declaration, DM-System2 BASIC)  
 Format: CALL POKES ([@]<address>, <string>)  
 Function: Converts a string to a sequence of bytes and saves them in any area of Main RAM, VRAM or Memory Mapper.  
 <address> – Is the writing start address. If greater than 65 534, the Memory Mapper specifications will be used. “@” Indicates VRAM.  
 <string> = String of characters to be converted to bytes.
- POKEW** (declaration, DM-System2 BASIC)  
 Format: CALL POKEW ([@]<address>, <value>)  
 Function: Writes two consecutive bytes to Main RAM, VRAM or Memory Mapper.  
 <address> – Is the address to be written. If greater than 65 534, the Memory Mapper specifications will be used. If it is preceded by “@”, it indicates VRAM.  
 <value> is the value to be written. It must be in decimal between 0 and 65,535, it can also be an expression.

**PON** (command, Network-BASIC)

Format: CALL PON

Function: Starts the student search. This instruction is only available to the teacher.

**PRINTERSETUP** (command, FM-X BASIC)

Format: CALL PRINTERSETUP

Function: It allows printing hiragana and graphic characters on the printer connected to the Fujitsu FM-7 computer when this machine is connected to the micro FM-X using the MB22450 interface.

**QDFILES** (command, QuickDisk BASIC)

Format: CALL QDFILES [("QD [n]:")]

Function: Lists the contents of the specified Quick Disk device in long format, with filenames, attributes and file sizes. The listed attributes are as follows:

01 – MainRAM binary file

02 – BASIC in tokenized format

03 – BASIC or DATA in ASCII format

0B – VRAM binary file

QD [n] specifies the QuickDisk device to be used. It can range from 0 to 7, the default being 0.

**QDFORMAT** (command, QuickDisk BASIC)

Format: CALL QDFORMAT

Function: Formats a QuickDisk excluding all existing files. The data is recorded on a spiral track on a 2.8 inch disc. A QuickDisk can save a maximum of 20 files and has a capacity of 64 Kbytes each side, with a maximum capacity of 128 Kbytes.

**QDKEY** (command, QuickDisk BASIC)

Format: CALL QDKEY (&lt;parameter&gt;)

Function: Modifies the content of the function keys, except F7, when a QuickDisk unit is connected. When one MSX is started with a connected QuickDisk drive, the contents of most function keys are modified. CALL QDKEY allows you to switch between content.

| Key     | Content      | New Command             |
|---------|--------------|-------------------------|
| F1      | _RUN         | CALL RUN                |
| F2      | _LOAD        | CALL LOAD               |
| F3      | _BLOAD       | CALL BLOAD              |
| F4 (*)  | list         | LIST                    |
| F5 (*)  | run          | RUN + [RETURN]          |
| F6 (**) | color 15,4,7 | COLOR 15,4,7 + [RETURN] |
| F7      | _QDKEY       | CALL QDKEY              |
| F8      | _SAVE ("QD:  | CALL SAVE ("QD:         |
| F9      | _BSAVE ("QD: | CALL BSAVE ("QD:        |
| F10     | _QDFILES     | CALL QDFILES            |

(\*) Generally unchanged

(\*\*) Unchanged on Japanese, Korean machines, Philips VG-8000 and VG-8010 (not on version 8010F), Sanyo PHC-28S  
 <parameter> – If 0, the default key content will be reloaded (except F7). With any other number or without a parameter the QuickDisk content will be loaded.

### **QDKILL** (command, QuickDisk BASIC)

Format: CALL QDKEY ("QD [n]:<filename>")

Function: Deletes the last QuickDisk file. Attempting to delete another file will return an error message.

QD [n] specifies the QuickDisk device to be accessed. It can range from 0 to 7, the default being 0.

<filename> is the file to be deleted and must be in the format 8.3 characters.

### **RAMDISK** (command, Disk-BASIC 2nd version)

Format: CALL RAMDISK (<max size>, [<created size>])

Function: Creates a RAMDISK with <maximum size> and optionally returns the actual <created size>. RAMDISK is accessed via the H: drive.

### **RCANCEL** (command, SFG-BASIC)

Format: CALL RCANCEL

Function: Cancels the rhythm instruments. Using this instruction, the total of simultaneous voices goes from 6 to 8.

**RCVMAIL** (command, Network-BASIC)

Format: CALL RCVMAIL (<student number>)

Function: Receives data from a student's sending mailbox in the teacher's receiving mailbox. This instruction is only available to the teacher. Mailboxes are special 256-byte areas reserved in the teacher and student's NetRAM. <student number> can vary from 1 to 15. Short version: \_RCVM.

**REBOOT** (remote, Hangul-BASIC 4, Rookie Drive BASIC)

Format: CALL REBOOT

Function: Causes a "hot" restart of the system.

**REC** (command, Hitachi-BASIC version 2)

Format: CALL REC

Function: Puts the internal data reader of the Hitachi MB-H2 micro in recording mode. This instruction does not support files in ASCII mode (BASIC or data).

**REC MK** (remote, MSX-Audio)

Format: CALL REC MK (<matrix name>)

CALL REC MK (<start address>, <end address>)

Function: Records a file played by the musical keyboard.

**REC PCM** (command, MSX-Audio)

Format: CALL REC PCM (<file number> [, SYNC] [,<offset>  
[,<size>] [,<sampling frequency>])

Function: Record audio in memory through the microphone.

<file number> – Number of the file to be written (0 to 15)

SYNC – If 0, MSX-Audio waits until an audio signal is detected. If it is 1, recording starts immediately.

<offset> – Offset in units of 256 bytes

<size> – Size of the audio file

<sampling frequency> – Can range from 1,800 to 49,716 Hz for ADPCM and from 1,800 to 16,000 Hz for PCM.

**RECEIVE** (command, Network-BASIC)

Format: CALL RECEIVE ([[<drive letter>:] <filename>],  
<student number>)

**Function:** Receives the BASIC program from a student's computer. This instruction can be used by the teacher and students who have been authorized by the teacher with CALL ENACOM. <drive letter> can be "A:" or "B:" and can only be used by the teacher. <student number> can vary from 1 to 15. Short version: \_RECE.

**RECFILE** (command, New Modem BASIC)

**Format:** CALL RECFILE (<string variable>), <numeric variable>

**Function:** Receive a file using a specific protocol

<string variable> contains the name of the file to be received (may include the name of the drive, if omitted, the file will be saved to the current active drive). If a file with the same name already exists on the disk, the first letter will be replaced by "\$", which will occur up to the fourth character.

<numeric variable> stores the protocol:

0 – Xmodem or Xmodem-1K

3 – Ymodem (allows you to receive multiple files simultaneously)

Upon return, <numeric variable> will contain the status:

0 – Receipt was done correctly

1 – Signal dropped (usually the connection is broken)

2 – Timed out (download has not started)

3 – Aborted operation with CTRL + X

4 – Many breaks (waiting times)

5 – Not used (no effect)

6 – Disk full

7 – File not found

8 – Recording error (write-protected disc or there is no disc)

9 – Empty file

10 – Too many attempts

**RECMOD** (command, MSX-Audio)

**Format:** CALL RECMOD (<recording mode>)

**Function:** Sets the recording mode for the musical keyboard.

<recording mode> is a value from 0 to 3:

- 0 - Mute (does not record)
- 1 - Records the melody played on the keyboard (def)
- 2 - Records, in another area, the reproduction of a melody already recorded
- 3 - Records the performance and playback of a melody already recorded

**REMOTE** (remote, Pioneer-BASIC)

Format: CALL REMOTE (<device number>, <string>)

Function: Controls external devices.

<device number> can range from 0 to 15, but devices 0, 1 and 2 are already assigned (Commands 3 to 15 must be assigned with CALL DEF UNIV.):

0 – Pioneer Laser Vision Player LD-700

1 – Pioneer Laser Vision Player LD-1100

2 – Pioneer Component Display SD-26

<string> contains a character code of up to 16 commands according to the following table (the “+” character can be omitted):

*Functions of the LD-700 model (device 0):*

|       |                                  |       |                               |
|-------|----------------------------------|-------|-------------------------------|
| A+ 48 | Repeat A                         | M+ 58 | Multi-speed forward           |
| A- 44 | Repeat B                         | M- 55 | Multi-speed reverse           |
| C+ 47 | Inc. multi-speed                 | P+ 17 | Play                          |
| C- 46 | Dec. multi-speed                 | P@ 16 | Reject                        |
| D+ 43 | Presents number of frame/chapter | P/ 18 | Pause                         |
| F+ 10 | Quick search                     | S+ 54 | Pauses frame to frame forward |
| F- 11 | Rev. quick search                | S- 50 | Pauses frame to frame reverse |
| L+ 4B | Audio 1 / left                   | T+ 51 | Fast forward (3x)             |
| L- 49 | Audio 2 / right                  | T- 59 | Fast rew (3x)                 |
| L@ 4A | Estéreo                          | X+ 45 | Clear                         |

*Functions of the LD-1100 model (device 1):*

|                         |                                  |
|-------------------------|----------------------------------|
| D+ Displays frame num   | P+ Play                          |
| D- Displays chapter num | P@ Reject                        |
| F+ Quick search         | P/ Pausa                         |
| F- Rev. quick search    | S+ Pauses frame to frame forward |
| L+ Audio 1 / left       | S- Pauses frame to frame reverse |
| L- Audio 2 / right      | T+ Fast forward (3x)             |
| M+ Slow search ahead    | T- Fast rew (3x)                 |
| M- Slow search reverse  |                                  |

*Functions of the SD-26 model (device 2):*

|    |    |           |    |    |                       |
|----|----|-----------|----|----|-----------------------|
| 1  | 01 | Channel A | F+ | 10 | Increment channel (+) |
| 2  | 02 | Channel B | F- | 11 | Decrement channel (-) |
| 3  | 03 | Channel C | K1 | 0C | Input: TV             |
| 4  | 04 | Channel D | K2 | 0D | Input: Video-Disc     |
| 5  | 05 | Channel E | K3 | 0E | Input: Video 1        |
| 6  | 06 | Channel F | K4 | 0F | Input: Video 2        |
| 7  | 07 | Channel G | O@ | 1C | Turns on / off        |
| 8  | 08 | Channel H | V+ | 0A | Increases volume (+)  |
| 9  | 00 | Channel I | V- | 0B | Decrease volume (-)   |
| 0  | 00 | Channel J |    | 48 | Sleep                 |
| C- | 46 | Channel K |    | 49 | Mute                  |
| C+ | 47 | Channel L |    | 4A | Display call          |
|    |    | 4D        |    |    | Channel M             |
|    |    | 4E        |    |    | Channel N             |
|    |    | 4F        |    |    | Channel O             |
|    |    | 50        |    |    | Channel P             |

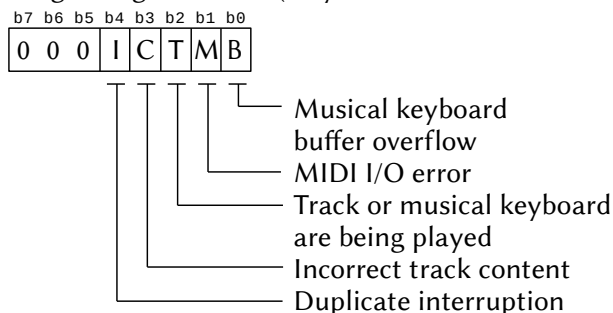
*Other functions:*

|    |    |                                      |
|----|----|--------------------------------------|
| M@ | ID | Turns on / off tape monitor          |
| P- | 15 | Reverse play (for tape-deck)         |
| W  |    | Hold video (for laser vision player) |
| R+ | 14 | Records (for tape-deck)              |
| R- | 12 | Mute record (for tape-deck)          |

**REPORT** (System variable, SFG-BASIC)

Format: CALL REPORT ([<error flag>] [, <mark number>]  
[, <number of repetitions remaining>])

Function: Returns the system variables. Short version: \_REPO.  
<error flag> integer variable (only the lowest 5 bits are valid):



<mark number> is an integer variable that returns the mark number of the last reproduced section.

<number of repetitions remaining> returns the number of times the rhythm will still be played.



- RESET** (command, RMSX BASIC)  
 Format: CALL RESET  
 Function: Restart the MSX1 or MSX2 computer emulated on a Turbo R machine by the rMSX emulator. It is a hot reset, as with DEFUSR=0: X=USR (0).
- REW** (command, Hitachi-BASIC version 2)  
 Format: CALL REW  
 Function: Causes the internal data reader of the Hitachi MB-H2 micro to rewind the tape.
- RHYTHM** (statement, SFG-BASIC)  
 Format: CALL RHYTHM (<repetition number> [,<brand number>])  
 Function: Reproduces the rhythm patterns selected by the CALL SELPATTERN command. Short version: \_RHYT.  
 <number of repetitions> specifies the number of repetitions in 1/4 note units.  
 <brand number> can range from 1 to 254. If omitted, the value 10 will be used.
- RMDIR** (command, Disk-BASIC 2nd version)  
 Format: CALL RMDIR (<subdirectory>)  
 Function: Removes the specified <subdirectory>.
- RSTOP** (declaration, SFG-BASIC)  
 Format: CALL RSTOP  
 Function: Stops rhythm playback. Short version: \_RSTO.
- RTCINI** (command, Hangul-BASIC 3)  
 Format: CALL RTCINI  
 Function: Resets the content of the RTC SRAM to the initial standard corresponding to MSX1.
- RTSOFF** (command, New Modem BASIC)  
 Format: CALL RTSOFF  
 Function: Turns off the carrier wave (RTS = Request To Send). This instruction works only when the DTR (Data Terminal Ready) signal is active (CALL DTRON).

**RTSON** (command, New Modem BASIC)

Format: CALL RTSON

Function: Turns on the carrier wave (RTS = Request To Send). This instruction works only when the DTR (Data Terminal Ready) signal is active (CALL DTRON).

**RUN** (command, Network-BASIC, QuickDisk-BASIC, X-BASIC)

Format: CALL RUN [ ( [<student number>], [<line number>] ) ]  
[Network-BASIC]

Function: Executes the BASIC program that is in the memory of a student's computer. This instruction is only available to the teacher. <student number> can vary from 0 to 15. If omitted or equal to 0, the programs of all computers will be executed. The program will run from line <line number>, if specified.

Format: CALL RUN ("[QD [n]:]<filename>") [QuickDisk-BASIC]

Function: Load a BASIC file from the specified Quick Disk device into MSX memory and execute it.

QD [n] specifies the QuickDisk device to be used. It can range from 0 to 7, the default being 0.

<filename> must be in the format 8.3 characters.

Format: CALL RUN [X-BASIC]

Function: Compiles and executes the BASIC program present in the MSX memory.

**SAVE** (command, DM-System2 BASIC, QuickDisk-BASIC)

Format: CALL SAVE (" [<device>:] [\ <path>] [\ <filename>],  
[@]<source address>, <size> [,<offset>])  
[DM-System2 BASIC]

Function: Saves data to a new file or somewhere in an existing file.

<device> can be drive A: to H: or COM: for computers connected with RS232C.

<path> specifies the location of the folder or file.

<filename> is the name of the file to be saved.

<destination address> is the source address of the data. If preceded by "@" it means VRAM.

<size> specifies the number of bytes to save.

<offset> specifies the offset in the target file.

Format: CALL SAVE ("[QD [n]:]" <filename> "[,A])  
[QuickDisk-BASIC]

Function: Saves data from memory or a BASIC program to a QuickDisk device. The data will always be saved in ASCII text. The BASIC program can be saved in ASCII or tokenized text.

QD [n] specifies the QuickDisk device to be used. It can range from 0 to 7, the default being 0.

<filename> must be in the format 8.3 characters.

[,A], if specified, saves the BASIC file as ASCII text.

### **SAVE PCM** (command, MSX-Audio)

Format: CALL SAVE PCM (<filename>, <file number>)

Function: Save audio file to disk.

<filename> is name of the file to be written to the disc

<file number> is the file number in the audio memory. It can range from 0 to 15.

### **SCLOAD** (command, Pioneer-BASIC)

Format: CALL SCLOAD [(<filename>)]

Function: Load data from the cassette to VRAM for display on the screen (only available for Screen 2)

### **SCOPY** (command, Hitachi-BASIC version 3)

Format: CALL SCOPY (<c1> [,<c2>, <c3>, <c4>.... <c15>])

Function: Sends to the printer a copy of a graphic screen in Screens 2, 4 or 5 using a formula based on the selected colors. The difference with CALL CSCOPY is unknown.

### **SCSAVE** (remote, Pioneer-BASIC)

Format: CALL SCSAVE (<filename>, [<baud rate>])

Function: Records VRAM data on the cassette. <baud rate> can be 1 (for 1200 baud) or 2 (for 2400 baud). If not specified, the baud rate defined in SCREEN will be used. Command available only for Screen 2.

### **SEARCH** (remote, Pioneer-BASIC)

Format: CALL SEARCH (<type>, {F | C}, <frame/chapter number>)

Function: Search the specified frame or chapter in the Laser Vision Player. <type> can be 0 for LD-700 or 1 for LD-1100. “F” search for a frame and “C” search for a chapter. <frame/chapter number> can vary between 0 and 54000.

### **SELPATTERN** (statement, SFG-BASIC)

Format: CALL SELPATTERN (<standard number>)

Function: Select the rhythm patterns for playback. Short version: \_SELP.

<pattern number> can vary from 1 to 8, with 1 to 6 being the ROM patterns and 7 and 8 being the patterns defined by the \_PATTERN command. The ROM defaults are:

|               |           |
|---------------|-----------|
| 1 – 16 beats  | 4 – Rock  |
| 2 – Slow rock | 5 – Disco |
| 3 – Waltz     | 6 – Swing |

### **SELVOICE** (statement, SFG-BASIC)

Format: CALL SELVOICE ([<voice 1>] [, <voice 2>]... [, <voice 8>])

Function: Select up to 8 voices chosen from the data loaded by the \_CLDVOICE command and execute them. <voice x> must correspond to the voice number created by the FM Voicing Program. The numbers from 49 to 56 are voices defined by the \_MODISNT command (these numbers will be used by default if the voice parameters are omitted).

### **SEND** (command, Network-BASIC)

Format: CALL SEND ([<unit name>:] <filename>] [, <student number>]])

Function: Sends the BASIC program to (other) students' computers. This instruction can be used by the teacher and students who have been authorized by the teacher with CALL ENACOM. <unit name> can be “A:” or “B:” and <student number> can range from 0 to 15. If <filename> is omitted, the BASIC program that is in the micro sender's memory will be sent.

**SENDFILE** (command, New Modem BASIC)

Format: CALL SENDFILE (<string variable>), <numeric variable>

Function: Send a file using a specific protocol.

<string variable> contains the name of the file to be sent  
(may include the name of the drive, if omitted, the  
file will be read from the current active drive).

<numeric variable> stores the protocol to be used:

1 – Xmodem.

2 – Ymodem-1K.

3 – Ymodem (allows only one file at a time).

Upon return, <numeric variable> will contain the status:

0 – Submission was successful.

1 – Signal dropped (usually the connection is broken).

2 – Timed out (upload has not started).

3 – Aborted operation with CTRL + X.

4 – Many breaks (waiting times).

5 – Not used (no effect).

6 – Disk full.

7 – File not found.

8 – Recording error (write-protected disc or no disc).

9 – Empty file.

10 – Too many attempts.

**SEOFF** (declaration, DM-System2 BASIC)

Format: CALL SEOFF

Function: Stops the playback of the sound effect. Requires SE driver.

**SEON** (declaration, DM-System2 BASIC)

Format: CALL SEON (<number>)

Function: Reproduces a sound effect from a table. Requires SE driver.  
<number> is the number of the sound effect to be played. It  
can range from 0 to 255, with 0 interrupting playback.

**SET PCM** (command, MSX-Audio)

Format: CALL SET PCM (<file number>, <device number>, <mode>,  
<parameter 1>, <parameter 2>, <sampling freq>)

Function: Defines parameters for the audio files. The parameters are  
defined for the following commands:

CONVA                      CONV                      COPY PCM  
 LOAD PCM                MK PCM                    PLAY PCM  
 REC PCM                    SAVE PCM

<file number> – File number in the audio memory. It can range from 0 to 15.

<device number> follows the table below:

| device | Device name  | Mode | Parameter 1 | Parameter 2 |
|--------|--------------|------|-------------|-------------|
| 0      | External RAM | 0/1  | –           | Size        |
| 5      | VRAM         | 0/1  | Address     | Size        |

The address and size are defined in units of 256 bytes.

<mode> can be: 0 – ADPCM, 1 – PCM

<sampling freq> can range from 1,800 to 49,716 Hz for ADPCM and from 1,800 to 16,000 Hz for PCM.

**SETBIN** (command, DM-System2 BASIC)

Format: CALL SETBIN (@<address>)

Function: Specifies the starting address of the binary table according to the binary system. <address> is the starting address of the binary table. The least significant bit is ignored. It is necessary to use "@" in front of <address> to put it in VRAM, otherwise an error will occur because the table cannot be placed in the Main RAM.

**SETPLT** (command, DM-System2 BASIC)

Format: CALL SETPLT (<address>)

Function: Defines the starting address of the color palette table. The table is 32 bytes long and the default address is C0000H.

**SETSE** (command, DM-System2 BASIC)

Format: CALL SETSE (<address>)

Function: Defines the starting address of the sound effects table. (Requires SE driver). <address> is the starting address of the table (0 to FFFFH), the initialization value being C000H.

**SIN** (function, DM-System2 BASIC)

Format: CALL SIN (<variable>, <angle>, <value>)

Function: Returns the sine of an angle. The result is obtained by multiplying the sine of the angle by a numerical value.

<variable> is a numeric variable that will receive the result.  
 <angle> is the angle value in degrees.  
 <value> is a number of two bytes (integer value).

**SJIS** (statement, Kanji-BASIC)

Format: CALL SJIS (<string variable>, <Kanji characters>)

Function: Converts a character in JIS code to a value of 4 hexadecimal digits.

<string variable> receives the 4 hexadecimal digits in ASCII

<Kanji characters> is a 2-byte Kanji character string where only the first one will be converted.

**SNDCMD** (command, Network-BASIC)

Format: CALL SNDCMD (<instruction>, [<student number>])

Function: Sends BASIC instructions to the student's computer and executes them. CHR\$(13) is sent at the end of the instruction and it is possible to send several by separating them with CHR\$(13). This instruction is only available to the teacher. <student number> can vary from 1 to 15. Short version: \_SNDC.

**SNDMAIL** (command, Network-BASIC)

Format: CALL SNDMAIL (<student number>)

Function: Sends data from the teacher's mailbox to a student's mailbox. This instruction is only available to the teacher. Mailboxes are special 256-byte areas reserved in the teacher and student NetRAM. <student number> can vary from 1 to 15. Short version: \_SNDM.

**SNDRUN** (command, Network-BASIC)

Format: CALL SNDRUN ([[<unit name>:]<filename>] [,<student number>])

Function: Send the BASIC program to the student's computer and execute it. This instruction is only available to the teacher. If a student already has a BASIC program in memory, it will be deleted and the student will receive a new one. <unit name> can be "A:" or "B:" and <student number> can range from 0 to 15. If <filename> is omitted, the BASIC program that is in the micro sender's memory will be sent. Short version \_SNDR.

**SOUND** (statement, SFG-BASIC)

Format: CALL SOUND (<instrument number>, <control mode>  
[,<pitch>] [,<fine tuning>] [,<speed>] [,<volume>])

Function: Controls instruments directly.

<instrument number> chooses the instrument from those defined by the \_INST instruction.

<control mode> can be:

0 – No key on / offline key

1 – Key on (note is audible)

2 – Key off (the note is at zero volume)

<pitch> can range from 25 to 120.

<fine adjustment> of the pitch. It can range from 0 to 100.

<volume> can range from 0 to 100, with 100 being the maximum volume (default).

**SPEAKEROFF** (remote, New Modem BASIC)

Format: CALL SPEAKEROFF

Function: Turns off the speaker.

**SPEAKERON** (remote, New Modem BASIC)

Format: CALL SPEAKERON

Function: Turn on the speaker.

**SPOLOFF** (command, Printer-BASIC)

Format: CALL SPOLOFF

Function: Disables the print spooler but does not empty the 32 Kbyte buffer. To clear the temporary buffer, it is necessary to use LPRINT or LLIST.

**SPOLON** (command, Printer-BASIC)

Format: CALL SPOLON

Function: Activates the print spooler, reserving a 32 Kbyte buffer for it.

**STANBY** (statement, SFG-BASIC)

Format: CALL STANDBY

Function: Temporarily stop playback. Short version: \_STAN.



- START** (command, Mega Assembler, SFG-BASIC)  
 Format: CALL START [Mega Assembler]  
 Function: Calls the Mega Assembler by initializing its variables. To call the MA without initializing, use \_ASM.  
 Format: CALL START [SFG-BASIC]  
 Function: Resumes playback interrupted by \_STANDBY. Short version: \_STAR.
- STATUS** (declaration, DM-System2 BASIC)  
 Format: CALL STATUS  
 Function: Displays the list of installed drivers for DM-System2.
- STDBY** (command, Hitachi-BASIC version 2)  
 Format: CALL STDBY  
 Function: Puts the internal data reader of the Hitachi MB-H2 micro in standby/suspend mode to save battery power.
- STOP** (command, Hitachi-BASIC, Network-BASIC, SFG-BASIC)  
 Format: CALL STOP [Hitachi-BASIC 2]  
 Function: Stops the tape movement in the internal data reader of the Hitachi MB-H2 micro.  
 Format: CALL STOP (<student number>) [Network-BASIC]  
 Function: Stops the BASIC program running on the student's computer. This instruction is only available to the teacher. <student number> can vary from 1 to 15. If omitted or equal to zero, execution will be interrupted on all student computers.  
 Format: CALL STOP (<instrument>) [SFG-BASIC]  
 Function: Suspend the playback of a specific instrument and, optionally, the digitization of the musical keyboard (when assigned to an instrument instead of a track by the CALL PLAY instruction). <instrument> must be a number between 1 and 4.
- STOPM** (statement, MSX-Audio, MSX-Music)  
 Format: CALL STOPM  
 Function: Stops the music played by MSX-Audio or MSX-Music.

**SYMBOL** (statement, Pioneer-BASIC)

Format: CALL SYMBOL (X, Y), CHR\$ (<character code>), [<hor>],  
[<vert>], [<color>], [<rotation>]

Function: Displays a character in Screen 2 in the coordinates

(X, Y). Optional parameters are as follows:

<character code> – ASCII character code

<hor> – Horizontal size multiplier. It can be between 1 and 32. If omitted, the value used will be 1.

<vert> – Vertical size multiplier. It can be between 1 and 24. If omitted, the value used will be 1.

<color> – Color code from 0 to 15. If omitted, the color defined by the COLOR command will be used.

<rotation> defines the character rotation.

0 – No rotation

1 – 90 degree rotation to the right

2 – 180 degree rotation to the right

3 – 270 degree rotation to the right

**SYNCOUT** (command, SFG-BASIC)

Format: CALL SYNCOUT

Function: Sends a synchronization signal to the cassette register.

Short version: \_SYNC.

**SYSOFF** (command, DM-System2 BASIC)

Format: CALL SYSOFF

Function: Uninstall DM-System2 BASIC and return to standard MSX-BASIC.

**SYSON** (command, DM-System2 BASIC)

Format: CALL SYSON

Function: Initializes the DM-System2 BASIC.

**SYSTEM** (command, Disk-BASIC, DM-System2 BASIC)

Format: CALL SYSTEM [Disk-BASIC version 1]

Function: Calls MSXDOS.

Format: CALL SYSTEM ["[<device>:] [\ <path>] [[\]<filename>"]]  
[Disk-BASIC version 2]

Function: Calls MSXDOS2, optionally executing the specified file or entering the subdirectory.

- Format: CALL SYSTEM [DM-System2 BASIC]  
 Function: Uninstall DM-System2 and return to the standard MSX-BASIC. If preceded by CALL SYSOFF, uninstall DM-System2 and call MSXDOS.
- TABOFF** (command, Hitachi-BASIC version 3)  
 Format: CALL TABOFF  
 Function: Disables the Drawing Tablet application on the Hitachi MB-H3 micro.
- TABON** (command, Hitachi-BASIC version 3)  
 Format: CALL TABON  
 Function: Launch the Drawing Tablet application on the Hitachi micro MB-H3.
- TALK** (statement, Network BASIC)  
 Format: CALL TALK (<message>, [<micro number>])  
 Function: Sends a message of up to 56 characters to the teacher or another student, when allowed by the teacher with CALL ENACOM. This instruction is only available to students. <micro number> can vary from 1 to 15 and can be obtained through CALL WHO. If it is 0, the message will be sent to the teacher. If, after sending the message, <micro number> contains 255, it failed, if it contains 0, the message was sent successfully.
- TDIAL** (remote, New Modem BASIC, SVI Modem BASIC)  
 Format: CALL TDIAL (<string variable>,<numeric variable> [New Modem BASIC])  
 Function: Calls a specific phone number via tone dialing. This instruction can only be used in a Terminal program. <string variable> stores the phone number to be called, where only the characters "0 123456789-AaBbCcDd\*#" are allowed (the character "-" corresponds to a 1 second wait). <numeric variable> returns the state: if it is 0, the input value is correct; otherwise not.  
 Format: CALL TDIAL ("<phone number>") [SVI Modem BASIC]  
 Function: Calls a specific phone number via tone dialing. <phone number> must be in quotation marks and only the characters "0 123456789AaBbCcDd" are allowed.

**TEMPER** (declaration, MSX-Music and MSX-Audio)

Format: CALL TEMPER (&lt;n&gt;)

Function: Sets the battery mode for OPLL. &lt;n&gt; can range from 0 to 21, the meaning of which is as follows:

- |                              |                              |
|------------------------------|------------------------------|
| 0 – Pythagoras               | 11 – Pure Rhythm Cis + (B-)  |
| 1 – Minton                   | 12 – Pure rhythm D + (H-)    |
| 2 – Welkmeyster              | 13 – Pure rhythm Es + (C-)   |
| 3 – Welkmeyster (adjusted)   | 14 – Pure rhythm E + (Cis-)  |
| 4 – Welkmeyster (separate)   | 15 – Pure rhythm F + (D-)    |
| 5 – Kilanbuger               | 16 – Pure rhythm Fis + (Es-) |
| 6 – Kilanbuger (adjusted)    | 17 – Pure rhythm G + (E-)    |
| 7 – Velotte Young            | 18 – Pure Rhythm Gis + (F-)  |
| 8 – Lamour                   | 19 – Pure rhythm A + (Fis-)  |
| 9 – Perfect rhythm (default) | 20 – Pure rhythm B- (G-)     |
| 10 – Pure rhythm C + (A-)    | 21 – Pure rhythm H- (Gis-)   |

**TEMPO** (statement, SFG-BASIC)

Format: CALL TEMPO (&lt;time value&gt;)

Function: Defines the “time” in quarter note units that will be played in one minute. It can range from 0 to 200, with 0 interrupting playback. Short version: \_TEMP.

**TERMINAL** (command, New Modem BASIC)

Format: CALL TERMINAL (&lt;numeric variable&gt;)

Function: Allows you to communicate with a BBS. Almost all keys pressed are sent over the phone line and what comes from the phone line is displayed on the screen. This instruction can only be used in a Terminal program. &lt;numeric variable&gt; stores the state:

- 1 – The signal has dropped (usually broken connection).
- 5 – The automatic login character (from the BBS) was received.
- 11 – The HOME key was pressed to return to BASIC. It can be used to return to the Terminal menu.
- 220 – GRAPH+I was pressed to return to BASIC. They can be used to manually send the name and password to a BBS without automatic login.

**TIMER** (command, SFG-BASIC)

Format: CALL TIMER (<period> [,<brand number>])

Function: Starts and sets the timer period.

<period> is defined in units of 1/100 seconds and can vary from 1 to 24,000.

<brand number> can be any number between 1 and 254. If omitted, the number 11 will be used.

**TRACE OFF** (command, MSX Aid BASIC)

Format: CALL TRACE OFF

Function: Stops program execution tracking.

**TRACE ON** (command, MSX Aid BASIC)

Format: CALL TRACE ON

Function: It starts tracking the execution of the program in the same way as the TRON instruction, but whenever the execution skips to another line, it sends the number of the executed line to the printer.

**TRACK** (declaration, SFG-BASIC)

Format: CALL TRACK (<number of tracks>)

Function: Defines the number of tracks used by \_PHRASE or \_PLAY. <number of tracks> for varying from 1 to 8; if omitted, the value will be 1. Short version: \_TRAC.

**TRANSPOSE** (declaration, MSX-Music and MSX-Audio, SFG-BASIC)

Format: CALL TRANSPOSE (<n>) [MSX Music /Audio]

Function: Changes the key. <n> can vary from -12799 to +12799, with 100 units corresponding to halftone. The default value is 0.

Format: CALL TRANSPOSE (<n>) [SFG-BASIC]

Function: Changes the key. <n> can range from -12 to +12 in halftone increments. The default value is 0.

**TSTOP** (command, SFG-BASIC)

Format: CALL TSTOP

Function: Stops the timer. The \_INIT statement also interrupts the timer. Short version: \_TSTO.

**TUNE** (command, SFG-BASIC)

Format: CALL TUNE (<numeric value>)

Function: Tunes the FM Tone Generation system with the other instruments. <numeric value> can range from -100 to +100, which corresponds to a semitone.

**UPPER** (function, DM-System2 BASIC)

Format: CALL UPPER (<variable>, <alphanumeric string>)

Function: Converts the alphabetic characters of the <alphanumeric string> to uppercase and returns it in the <variable>.

**USBCD** (remote, RookieDrive BASIC)

Format: CALL USBCD ("<directory>")

Function: Change the active directory on the USB device.

**USBERROR** (statement, RookieDrive BASIC)

Format: CALL USBERROR

Function: Displays the stored error code whenever a USB transaction fails for any reason. Only the error of the last executed USB transaction is stored.

**USBFILES** (remote, RookieDrive BASIC)

Format: CALL USBFILES

Function: Displays the list of disk images that are in the root directory of the USB virtual drive. The execution of this instruction takes the disk to the "offline" state. To return to the "online" state, use CALL INSERTDISK or CALL REBOOT.

**USBRESET** (remote, RookieDrive BASIC)

Format: CALL USBRESET

Function: Repeat the initialization procedure that is performed when a standard USB floppy drive is connected to a Rookie Drive interface.

**USERHYTHM** (statement, SFG-BASIC)

Format: CALL USERHYTHM

Function: Enables the rhythm instruments (drums) for use. These instruments use two FM voices; so the number of available voices drops from 8 to 6 with the rhythm enabled. Short version: \_USER.

**USR** (command, Nextor)

Format: CALL USR (<execution address>, [<registers>])

Function: Calls a routine in Assembler, optionally loading registers with specific values beforehand.

<execution address> is the starting address of the routine.

If “-1” is specified, the routine will only return without error (useful for detecting Nextor in BASIC).

<registradores> is a pointer to a 12-byte buffer where the values of the registers are specified in the sequence “F, A, C, B, E, D, L, H, IXl, IXh, IYl, IYh”.

**VARLIST** (command, MSX Aid BASIC)

Format: CALL VARLIST ([("<variable>") [, P]])

Function: Displays a list of all variables already used by the MSX-BASIC program that is in memory. If <variable> is specified (1 or 2 characters), line numbers with the variable in question will be listed. If the second character is an asterisk (\*), all variables that start with the first character are considered. With parameter P, the data will be sent to the printer. Without any parameters, the complete list will be displayed on the screen.

**VCOPY** (declaration, DM-System2 BASIC)

Format: CALL VCOPY (<X0>, <Y0>) - (<X1>, <Y1>) [, <PgF>] TO  
(<X2>, <Y2> - <X3>, <Y3>) [, <PgD>] [, <R>]  
[ON (<X4>, <Y4>)] [, <logical operator>]

Function: Copies a rectangular area of VRAM to another with zoom in / out and rotation.

<X0> - X coordinate of the first point in the source area.

<Y0> - Y coordinate of the first point in the source area.

<X1> - X coordinate of the second point in the source area.

<Y1> - Y coordinate of the second point in the source area.

<PgF> - VRAM source page.

<X2> - X coord of the first corner of the destination area.

<Y2> - Y coord of the first corner of the destination area.

<X3> - X coord of the opposite corner of the target area.

<Y3> - Y coord of the opposite corner of the target area.

<PgD> - VRAM's landing page.

<R> – Clockwise rotation in degrees.

<X4> – X coordinate of the rotation axis (X2 is standard).

<Y4> – Y coordinate of the rotation axis (Y2 is standard).

Note: <X> can vary from 0 to 511 and <Y> from 0 to 1023.

<LO> is the logical operator and can be [T]PSET, [T]PRESET, [T]XOR, [T]OR or [T]AND. The default is PSET.

**VDPWAIT** (command, DM-System2 BASIC)

Format: CALL VDPWAIT

Function: Wait until the VDP finishes executing the command.

**SEE** (statement, Hangul-BASIC 4)

Format: CALL VER

Function: Displays the version of Hangul-BASIC.

**VIDEO** (function, Pioneer-BASIC)

Format: CALL VIDEO (<variable>)

Function: Returns in the <variable> the type of video selection currently active. The returned value can be:

0 – Computer screen (internal synchronization)

1 – Superimpose

2 – External video

**VLIST** (declaration, SFG-BASIC)

Format: CALL VLIST

Function: Displays the instrument table on the screen.

**VMOFF** (control, DM-System2 BASIC)

Format: CALL VMOFF [(<segment number>)]

Function: Aborts the macro operation.

**VMON** (command, DM-System2 BASIC)

Format: CALL VMON (<start address>, [<start value>])

Function: VDP processing macro operation.

<start address> specifies the start of the macro code.

<initial value>, if specified, causes the macro operation to start only after being stored in the VDP macro variable.



**VMWAIT** (command, DM-System2 BASIC)

Format: CALL VMWAIT

Function: Puts the system on hold until the VDP macro operation is complete. CTRL+STOP can be used to exit this command.

**VOICE** (declaration, MSX-Music and MSX-Audio)

Format: CALL VOICE ([@ <n1>], [@ <n2>], ..... [@ <n9>])

Function: Specifies the instruments to be used in each voice.  
<nx> can range from 0 to 63. The default value is 0.

**VOICE COPY** (statement, MSX-Music and MSX-Audio)

Format: CALL VOICE COPY (@<n1>, - <n2>)

Function: Copies data related to the instruments to / from a matrix variable type DIM A%(16). <n1> is the source and <n2> the destination. <n1> can range from 0 to 63 and <n2> can only be 63, or <n1> and <n2> can be a matrix variable.

**WAIT** (command, DM-System2 BASIC, SFG-BASIC)

Format: CALL WAIT (<time>) [DM-System2 BASIC]

Function: Wait for a defined time. It can be aborted by CTRL+STOP.  
<time> is defined in 1/60 second units and can range from 0 to 32767.

Format: CALL WAIT (<event number>) [SFG-BASIC]

Function: Suspend the interruption when the melody is being played.  
<event number> can be:  
1~4 – Suspend while playing the respective instrument.  
5 – Suspend during rhythm playback.  
6 – Suspend until the timer time is zero.

**WHO** (statement, Network BASIC)

Format: CALL WHO (<micro number>)

Function: Returns the number of the computer in the MSX network.  
<micro number> can range from 0 to 15, where 0 is the teacher's micro.

**XREF** (statement, MSX Aid BASIC)

Format: CALL XREF [( <line number> ) [, P]]

**Function:** Displays a list with all the linked lines of an MSX-BASIC program that is in memory (GOSUB, GOTO, RESUME, RESTORE, RETURN instructions). <line number> is used to limit the list to a specified line number. With parameter P, the data will be sent to the printer. Without any parameters, the complete list will be displayed on the screen.

**XY** (command, DM-System2 BASIC)

**Format:** CALL XY (<X coordinate>, <Y coordinate>)

**Function:** Changes the coordinates of the graphic accumulator.

**YMMM** (declaration, DM-System2 BASIC)

**Format:** CALL YMMM (<X0>, <Y0>) – [STEP] (<X1>, <Y1>) TO  
(<X2>, <Y2>)

**Function:** Executes the YMMM command (quick copy in bytes in the Y direction) of the VDP. Available for Screens 5 to 12.

<X0> – X coordinate of the first point in the source area.

<Y0> – Y coordinate of the first point in the source area.

<X1> – X coordinate of the second point in the source area.

<Y1> – Y coordinate of the second point in the source area.

<X2> – Left X coordinate of the target area.

<Y2> – Upper Y coordinate of the target area.

STEP, if specified, indicates relative coordinates.

Note: <X> can vary from 0 to 511 and <Y> from 0 to 1023.

### 3.4 – MSX-BASIC ERROR CODES

- 01 NEXT without FOR
- 02 Syntax error
- 03 RETURN without GOSUB
- 04 Out of DATA
- 05 Illegal function call
- 06 Overflow
- 07 Out of memory
- 08 Undefined line number
- 09 Subscript out of range
- 10 Redimensioned array
- 11 Division by zero

- 12 Illegal direct
- 13 Type mismatch
- 14 Out of string space
- 15 String too long
- 16 String formula too complex
- 17 Can't CONTINUE
- 18 Undefined user function
- 19 Device I/O error
- 20 Verify error
- 21 No RESUME
- 22 RESUME without error
- 23 Unprintable error
- 24 Missing operand
- 25 Line buffer overflow
- 26~49 Unprintable error
- 50 FIELD overflow
- 51 Internal error
- 52 Bad file number
- 53 File not found
- 54 File already open
- 55 Input past end
- 56 Bad filename
- 57 Direct statement in file
- 58 Sequential I/O only
- 59 File not OPEN
- 60 Bad FAT
- 61 Bad file mode
- 62 Bad drive name
- 63 Bad sector
- 64 File still open
- 65 File already exists
- 66 Disk full
- 67 Too many files
- 68 Disk write protected
- 69 Disk I/O error
- 70 Disk offline
- 71 RENAME across disk
- 72 File write protected

- 73 Directory already exists
- 74 Directory not found
- 75 RAM disk already exists
- 76 Invalid device driver \*
- 77 Invalid device or LUN \*
- 78 Invalid partition number \*
- 79 Partition already in use \*
- 80~255 Unprintable error

Obs. The codes marked with “\*” (76 a 79) are for Nextor only.

## 4 – MSXDOS

COMMAND NAME (command type, COMMAND version)

Format: Valid formats for the command

Function: Command operation mode

Internal commands are commands executed directly by COMMAND.COM, and external commands are loaded from the disk.

The COMMAND version indicates the version for which the command is implemented. Values separated by “–” indicate that there are differences in syntax or behavior for different versions. Next there is a short description of the versions.

1 – MSXDOS version 1.0

2 – MSXDOS version 2.0 (Command up to version 2.3)

2.41 – MSXDOS version 2.0 (Command version 2.41)

K – Kanji-ROM required

### 4.1 – FORMAT NOTATION

<filename> – Filename in the form: A:\ dir1 \ dir2 \ file.ext

<compound filename> – Multiple filenames in the above format

<path> – Path in the form: A:\ dir1 \ dir2 \

[ ] delimits optional parameter.

| it means that only one of the items can be used (OR).

{ } delimits option.

Chars in parentheses after some options for some commands indicate the version of COMMAND for which that option is available.

A <device> can be:

CON Console (Keyboard)

CRT Video

PRN Printer

NULL Null

AUX Auxiliary

COM Serial port

Or whatever is installed.

### 4.1.1 – Description of filenames extensions

|     |  |
|-----|--|
| ACC | Music Creator accompaniment data files   |
| ARC | File(s) compressed in ARC format by System Enhancement Associates (SEA). Tools to extract are UNARC.COM (v1.6) and UNP.COM (v1.0 by Pierre Gielen).  |
| ARC | File(s) compressed in Russian ARC format, incompatible with SEA's ARC format. Tools to extract are XARC.COM (v1.01) and ARCDE.COM (v1.03).   |
| ARJ | File(s) compressed in ARJ format. Tool to extract are UNARC.COM (v1.10) and UNP.COM (v1.0 by Pierre Gielen).   |
| APT | Studio FM pattern data file.   |
| ASC | Plain text (ASCII format) that can contain a BASIC program or data.  |
| ASM | Assembler text file.   |
| ASN | Assignment files for MIDI Blaster  |
| BAS | BASIC program listing tokenized. These files can be executed from MSX-DOS with the BASIC name.bas command.   |
| BAT | Batch files (plain text) interpreted by MSX-DOS.   |
| BGM | MuSICA binary music file. MuSICA is a software developed by ASCII to create music on 17 voices with PSG, FM and Konami's SCC.  |
| BGM | MSX-FAN music file. Not to be confused with MuSICA files. Songs in this format were contained in all their disk magazines, and later a specific player was released that even supported playback on MIDI.      |
| BGM | Bloadable MSX-MUSIC file created by the BIT2BGM.COM utility of Uwe Schröder that converts Synth Saurus musical files.  |
| BIN | Binary file created with the BASIC BSAVE instruction. Loads with BLOAD. The header have a length of 7 bytes (FEh + Start address + End address + execution address). It can contain machine language and data. |

|     |  |
|-----|--|
| BMP | Image file in format . Viewable in SCREEN 7 or 8 with BMP.COM (v1.01 by SEIGA).  |
| BOK | MSX View Picture book.   |
| BTM | Batch files supported by MSX-DOS 2 v.2.40 or later.  |
| CAS | BIOS level cassette image for emulators, needs a separate tool to run or write to cassette. SofaCas allows to convert software on tape to CAS file and also play it using a homemade cable PC sound output to MSX cassette input. On MSX turbo R, we can use TRCAS (by Martos) to run CAS played by SofaCas. |
| CMP | Compressed screen 5 image, including palette, created with DD-Graph (Dot Designer's Graph) (aka DD-Graph).   |
| CMP | Compressed image, including palette, created with GIOS. GIOS, aka Graphical Input/Output System  |
| COM | Command containing a binary executable under MSX-DOS. Can be also an executable file compressed with POPCOM.COM (v1.0 by Perpermint-Star).   |
| CPM | .COM file renamed to .CPM, either to be used in some CPM emulators, or to be able to workaround GMail's nanny protection against executable files. Just rename those back to .COM to be able to run them.  |
| DRM | File for the drums editor of the First Rate Music Hall tracker.  |
| DAT | Synthesizer configuration file for MIDI Blaster  |
| DSK | Disk image for emulators, needs a separate tool to run or write to normal disk. Can be launched on real MSX with SofaRunit or using Nextor's EMUFILE command.  |
| DUA | Music-BOX dual data file (melody + sample)   |
| EDI | File for the song editor of the First Rate Music Hall tracker.   |
| EMx | Disk image for the floppy disk emulator (HDDEMU.COM) for MSX Turbo R by Tsuyoshi. Internal structure is same as in DSK-files. Protected disks have additional information stored to files with HED-extension.  |

|     |   |
|-----|---|
| EVA | Video file in EVA format.   |
| EVG | Yamaha SFG-05 event data file.  |
| FM  | MSX-MUSIC BASIC file.   |
| FMP | MSX-MUSIC BASIC file.   |
| FMS | Synth Saurus sound file.  |
| FNT | Font file for the Scroll Power utility.   |
| FON | MSX View font.  |
| G9B | Library graphic format for GFX-9000.  |
| GE5 | Synonym for .SC5. See the .SCx file description.  |
| GEN | Plain text that contain Z80 assembly source code, used with GEN80 compiler.   |
| GIF | Graphics Interchange Format. They can be viewed with GIFL.COM (by Kakami Hiroyuki) and converted in MSX format with ENGIF.COM (v1.2 by Pierre Gielen), SHOWEM.COM (by Steven van Loef) and GIFDUMP.COM (by Francesco Duranti) |
| GLx | Graph Saurus image file like BASIC instruction COPY. Can be used under BASIC.   |
| GRA | Image file in QLD format. The viewer BLS.COM (v2.00 by SEIGA) support it.   |
| GRP | Synonym for .SC2. See the .SCx file description. Can also be a compressed image for Graph Saurus.   |
| GZ  | File compressed in GZIP format by PC gzippers. Tool to extract is GUNZIP.COM.   |
| HLP | MSX-DOS 2 help file (plain text).   |
| INS | File for the instruments editor of the First Rate Music Hall tracker.   |
| IPS | Patch for file. Needs IPS patcher.  |
| ISH | Compressed file.  |



- JPG Compressed image file in format JPEG. Some viewer can show image until 1024x1024: JPD.COM (v0.23 by APi), JLD.COM (v1.11 by SEIGA) or BLS.COM (v2.00 by SEIGA). JPEG file can be produced on MSX from SCREEN 12 images with JSV.COM (v0.1 by SEIGA).
- KSS MSX music file that contains also player code. Use KSS-PLAY.COM (by NYRIKKI) to play it.
- LDR Tokenized Basic file usually BASIC program LoadR used to load and run a program consisting of several BAS files.
- LHA File(s) compressed in LHA format. Tools to create a LHA archive are LHPACK.COM (v1.03 by H.Saito) or LHA.COM (v1.05a by Kyouju). Tools to extract are PMM.COM (v1.20 by Iita), LHARC.COM and LHEXT.COM (v1.33 by Kyouju).
- LPF Loop file for the Scroll Power utility.
- LZH Synonym for LHA.
- MAG Maki-chan V2 image file mainly used on PC-9801 and Sharp X68000. Viewable with BLS.COM (v2.00 by SEIGA)
- MAX Synonym for MAG.
- MBK Sample kit file for the music tracker MoonBlaster.
- MBM Music file for the music tracker MoonBlaster.
- MBS Sample file for the music tracker MoonBlaster.
- MBV Voice file for the music tracker MoonBlaster.
- MBW Wave song for the music tracker MoonBlaster.
- MCM Micro Cabin music file. Played by MCDRV.EXE.
- MDT MSX Music-System music data file.
- MDX Music file in a format designed for Sharp X68000. These files can be played by MPX2.COM (when driver installed with MXDRV.COM). Optional PDX files are PCM samples. Require the YAMAHA SFG-01/05 cartridge or the MFP PCM cartridge.

|     |   |
|-----|---|
| MEG | Plain text that contain Z80 assembly MegaAssembler source code. Extension also used for Mega-Rom images.                                  |
| MEL | Music-BOX melody data file.   |
| MFM | FM song for MoonBlaster.  |
| MGS | Music file in format developed by AIN. Played by MGSEL.COM (when driver installed with MGSDRV.COM).                                       |
| MID | Standard MIDI file (can be played using MIDI-interface or MoonSound software)   |
| MIF | Compressed image file (MSX Image Format). Can be viewed with MIFVIEW.COM.   |
| MIO | MIODRV Music file. Played by MIODRV Player.   |
| MKI | Maki-chan V1 image file mainly used on Sharp X68000. Viewable with BLS.COM (v2.00 by SEIGA).  |
| MOD | Amiga MOD file. Can be played on MSX turbo R or MoonSound.  |
| MP3 | MPEG Audio Layer III file. MP3s can be played with Sunrise MP3 player, MPX Cartridge r1.1 by Junsoft or SE-ONE by TMT logic.              |
| MPK | Music Player K-kaz song. Require WAMPK Player.  |
| MSx | Synth Saurus score file.  |
| MSD | MuSICA source music file (MML). We can also use KINROU4 (by Masarun), an alternative compiler.  |
| MUE | HAL Music Editor MUE music file. There's a patch to add mouse support here.   |
| MUS | FAC Soundtracker music file.  |
| MUS | MGSDRV source MML file. Needs to be compiled to a MGS file with MGSC.COM. OTOH, MGSCR.COM can decompile MGS files back to the MUS source. |
| MUS | Studio FM music file (not recommended).   |

|     |  |
|-----|--|
| MWK | MoonSound Wave sample kit.   |
| MWM | MoonBlaster for MoonSound Wave song.   |
| OPX | OPLL driver music format.  |
| PAC | Dump of SRAM contents (save games) of PAC or FM-PAC cartridge.   |
| PAT | Studio FM pattern file.  |
| PCM | Sound sample file for MSX-Turbo R.   |
| PCK | Packaged file for First Rate Music Hall tracker. Includes 4 songs with all instruments and drums data.   |
| PCT | Dynamic Publisher page files.  |
| PDX | Optional PCM sample file used with an MDX file. You can play a PDX with PDXLOAD.COM by AIN. See also MDX extension.  |
| PIC | Phillips Video Graphics image. Synonym for .SC8, so check the .SCx description. Also specific image format used of X68000, it is viewable with BLS.COM (v2.00 by SEIGA)  |
| PLx | Graph Saurus colors palette file in Raw format (contains 8 sets of palettes with two bytes by color RG 0B). It's a companion for the respective .SRx file, so both files must always be copied together.   |
| PMA | File(s) compressed in PMARC format. Tools to create an archive are PMARC.COM, PMARC2.COM (v2.0 by Sybex) and UNP.COM (v1.0 by Pierre Gielen). Tools to extract are PMM.COM (v1.20 by Iita), PMEXE.COM (v2.0) and PMEXT.COM (v2.22). PMEXT has been ported on Windows (v1.21 by Yoshihiko Mino). To extract a PMA file on a Mac use The Unarchiver. |
| PRO | Music file for Pro-Tracker (by Typhoon Soft).  |
| PSG | PSG Sampler sample file.   |
| RDT | MSX Music-System rhythm data file.   |
| RLT | Music Creator real time data files.  |

|     |   |
|-----|---|
| ROM | Raw ROM image dump. Used by ROM loaders or emulators.   |
| RTM | Synth Saurus rhythm file.   |
| S1x | Contains the odd lines of an interlace image. For more info, see the SCx file.  |
| S3M | See MOD file.   |
| SAM | Music-BOX sample data file (used as drumkit file in Music Creator)  |
| SBM | Music data for SCC and PSG soundchips.  |
| SBS | Instruments data for SCC and PSG soundchips.  |
| SCx | Screen-x binary image file. Can have a companion .S1x file that will contain the extra interlaced lines to double the vertical resolution. Used by image editors or BLOAD instruction with the parameter S. SC2 images can be viewed under MSX-DOS with SC2VIEW.COM (by GDX), and SC5 to SCC files with BLS.COM (by SEIGA). |
| SCR | Screen-2 image created with Graphos III. It's an executable file with a loader that produces an effect. Loadable on MSX-BASIC with BLOAD"file",R.   |
| SDT | MSX Music-System sound data file (= voice data)   |
| SDT | SCMD Music file for MSX made a MML compiler for Windows. The player is SC.COM.  |
| SEE | Sound Effect data, for use in Sound Effect Editor (Shareware by Fuzzy Logic)  |
| SEQ | Music Creator sequence data files.  |
| SFM | Studio FM music file.   |
| SMx | FAC Soundtracker sample file.   |
| SMP | Sample file for Covox/SIMPL or MSX Turbo R.   |
| SNG | Music file for the music editor SCC-Musixx by Tyfoon Soft.  |

|     |  |
|-----|--|
| SPT | Music Creator step time data files.  |
| SPT | Text file for the Scroll Power utility.  |
| SRx | Graph Saurus Image file. Requires the respective .PLx file. Can be optionally compressed with run-length. Uncompressed files can be loaded on MSX-BASIC with a BLOAD"FILE.SRx",S, but the external palette will have to be loaded with OPEN#1. |
| STP | Dynamic Publisher stamp files. Contains an image that can be loaded on a page (.STP).  |
| TIx | Graph Saurus tile file.  |
| TSR | Terminate and Stay Resident programs to be used with MemMan 2.0 and higher.  |
| TXT | Plain text file generally coded in ASCII, Ank or JIS.  |
| VCD | Voice file for the MSX Voice Recorder (HAL Laboratory).  |
| VCD | MuSICA voice file.   |
| VGM | Music file that supports many sound chips, playable by VGMPLOY.COM (by Laurens Holst)  |
| VOC | Music Creator voice data files.  |
| VOC | Studio FM voice data file.   |
| VOG | Yamaha SFG-05 voice data file.   |
| WAV | Sound sample file. Can be played with the MPX Cartridge r1.1 by Junsoft.   |
| WB  | Assembler Project file. For the The WBASS2 Z80 Assembler.  |
| XM  | See MOD file.  |
| XPC | ROM Patch file for EXECROM.COM (A&L Software)  |
| ZIP | File(s) compressed in ZIP format by PC zipers. The best tool to extract is SUZ.COM (v1.3 by Loutrax).  |

## 4.2 – DESCRIPTION OF COMMANDS

### **ALIAS** (internal, 2.41)

Format: ALIAS [/P] [name] [=] [value] | /R | {/L | /S} <filename>

Function: Displays or sets the alias command.

[/P] Pauses the listing when completing a screen.

[/R] Removes all defined aliases.

[/L] Loads an alias defined in <filename>.

[/S] Saves the current alias to file <filename>.

[name] is the name of the new command.

[value] is the command or string that will be assigned to  
[name]

<filename> is the file on disk to which it will be written or  
where the defined alias will be retrieved.

### **ASSIGN** (internal, 2)

Format: ASSIGN [d1: [d2:]]

Function: Redirects access to drive d1: to drive d2:.

### **ATDIR** (internal, 2)

Format: ATDIR + | -H [/H] [/P] <filename>

Function: Enables / disables hidden directory attributes.

[/P] pauses error messages when completing a screen.

+H marks file as hidden.

-H turns off the hidden file attribute, and must be followed  
by /H.

### **ATTRIB** (internal, 2-2.41)

Format: ATTRIB {+ | -H | + | -R | + | -S | + | -A} [/H] [/P] <filename>

Function: Change attributes of hidden file (H) read-only (R), system  
file (S, 2.41 only) or archived (A, 2.41 only). "-H" must be  
used with "/H".

[/P] pauses error messages when completing a screen.

### **BASIC** (internal, 1)

Format: BASIC [<prog name>]

Function: It transfers control to the BASIC interpreter and optionally  
loads and executes the program <prog name>.

**BEEP** (internal, 2.41)

Format: BEEP

Function: Generates a beep.

**BOOT** (internal, 2.41)

Format: BOOT [drive]

Function: Exchanges the MSXDOS boot drive from BASIC.

**BUFFERS** (internal, 2)

Format: BUFFERS [number]

Function: Displays or sets the number of system I/O buffers.

**CD** (internal, 2)

Format: CD [[d:] [path] | -]

CHDIR [[d:] [path] | -]

Function: Display or change the current subdirectory. If “-” is specified, returns to the previous directory.

**CDD** (internal, 2.41)

Format: CDD [[d:] [path] | -]

Function: Displays or changes the current subdirectory and drive. If “-” is specified, it returns to the previous drive / directory.

**CDPATH** (internal, 2.41)

Format: CDPATH [[+ | -] [d:] path [[d:] path ...]]

Function: Displays or sets the search path.

**CHDIR** (internal, 2)

Format: The same as the CD command.

Function: The same as the CD command.

**CHKDSK** (internal, 2)

Format: CHKDSK [d:] [/F]

Function: Checks the integrity of the files on the disk. If [/F] is specified, files will not be corrected; only information about the integrity fault will be shown.

**CLS** (internal, 2)

Format: CLS

Function: Clears the screen.

**COLOR** (internal, 2.41)

Format: COLOR <front color> [<backgrd color> [<border color>]]

Function: Change the colors of the screen.

**COMMAND2** (internal, 2)

Format: **COMMAND2** [command]

Function: Executes a *MSXDOS2* command.

**CONCAT** (internal, 2-2.41)

Format: `CONCAT [/H] [/S] [/P] [/A] [/B] [/V] <source files>`

<destination files>

Function: Concatenates all source files into a single file.

[/H] Hidden files will also be concatenated.

[/S] System files will also be concatenated (only 2.41).

[/P] Pause messages when completing a screen.

[/B] Concatenates without interpretation.

**[/A]** Reverses the effect of **[/B]**.

[/V] Check concatenated file created.

**COPY** (internal, 1-2-2.41)

Format: COPY [/H] [/S] [/P] [/A] [/B] [/V] [/T] <source files>

<destination files>

Function: Copies files.

[/H] Hidden files will also be copied (2).

[/S] System files will also be copied (2.41).

[/P] Pause messages when completing a screen.

[/A] Makes an ASCII copy (adds Ctrl+Z to the end of the file).

**[/B]** Reverses the effect of **[/A]**.

[/V] Checks copied file.

[/T] Changes the date and time of the copied file to the current one.

CPU (internal, 2.41)

Format: CPU [number]

Function: Display or change the CPU for the MSX turbo R  
(0 = Z80; 1 = R800 ROM; 2 = R800 DRAM).

**DATE** (internal, 1-2.41)

Format: DATE [date]

Function: Displays or changes the system date. [date] must be in format “mm-dd-yyyy” or in format defined by SET DATE.



**DEL** (internal, 1)

Format: DEL [/S] [/H] [/P] <compound filename>  
 ERA [/S] [/H] [/P] <compound filename>  
 ERASE [/S] [/H] [/P] <compound filename>

Function: Delete one or more files.  
 [/S] System files will also be deleted (2.41).  
 [/H] Hidden files will also be deleted.  
 [/P] Pause messages when completing a screen.

**DELALL** (external, Nextor)

Format: DELALL <drive letter>:  
 Function: Quick format for a drive unit.

**DEVINFO** (external, Nextor)

Format: DEVINFO <driver slot> – [<driver subslot>]  
 Function: Displays information about devices controlled by Nextor.

**DIR** (internal, 1-2-2.41)

Format: DIR [/S] [/H] [/W] [/P] [/2] [<compound filename>]  
 Function: Displays the filenames of the disk.  
 [/S] System files will also be listed (2.41)  
 [/H] Hidden files will also be listed  
 [/W] List filenames only  
 [/P] Pauses the listing when completing a screen  
 [/2] List in two columns (2.41)

**DISKCOPY** (external, 2)

Format: DISKCOPY [d1: [d2:]] [/X]  
 Function: Copies an entire disk (d1:) to another (d2:)  
 [/X] Suppress messages during copying.

**DRIVERS** (external, Nextor)

Format: DRIVERS  
 Function: Displays information about the drivers available for Nextor and MSXDOS.

**DRVINFO** (external, Nextor)

Format: DRVINFO  
 Function: Displays information about all available drive letters.

**DSKCHK** (internal, 2.41)

Format: DSKCHK [ON | OFF]

Function: Displays or sets the check status of the disc.

**ECHO** (internal, 1)

Format: ECHO [text]

Function: Prints a text while executing a batch file with line feed at the end.

**ECHOS** (internal, 1)

Format: ECHOS [text]

Function: Prints a text during the execution of a batch file without line feed at the end.

**ELSE** (internal, 2.41)

Format: ELSE [command]

Function: Conditional command execution. Without the [command] parameter, toggle Command Mode between ON/OFF.

**END** (internal, 2.41)

Format: END

Function: Ends a batch file (batch).

**ENDIFF** (internal, 2.41)

Format: ENDIFF [command]

Function: Increase a level and restore Command Mode.

**ERA** (internal, 1)

Format: The same as the DEL command.

Function: The same as the DEL command.

**ERASE** (internal, 1)

Format: The same as the DEL command.

Function: The same as the DEL command.

**EXIT** (internal, 2)

Format: EXIT [number]

Function: Exits the program executed by the COMMAND2 command. [number] is the user's error code (the default is 0).

**FASTOUT** (external, Nextor)

Format: FASTOUT [ON | OFF]

Function: Turns on/off the quick output for the STROUT routine, or displays the current STROUT status.

**FIXDISK** (external, 2)

Format: FIXDISK [d:] [/S]

Function: Updates a disk to the MSXDOS2 format.  
[/S] Update complete.**FORMAT** (internal, 1-2.41)

Format: FORMAT [d:] (1)

FORMAT [d: [option [/X]]] (2.41)

Function: Formats a disc. If [option] is specified, it formats with that option, without displaying a list of options.  
[/X] Starts immediate formatting, without displaying a message.**FREE** (internal, 2.41)

Format: FREE [d:]

Function: Displays the total, free and used space of the disk.

**GOSUB** (internal, 2.41)

Format: GOSUB~label

Function: Executes a subroutine within a batch file.

**GOTO** (internal, 2.41)

Format: GOTO~label

Function: Jump to the label within a batch file.

**HELP** (internal, 2)

Format: HELP [&lt;filename&gt;]

Function: Displays the help file &lt;filename&gt;.HLP or lists all if there is no argument.

**HISTORY** (internal, 2.41)

Format: HISTORY [/P]

Function: Displays the command history.  
[/P] pauses history when completing a screen.

**IF** (internal, 2.41)

Format: IF [NOT] EXIST [d:] [<path>] <filename> [THEN] <command>  
 or  
 IF [NOT] <expr1> == | EQ | LT | GT <expr2> [AND | OR |  
 XOR [NOT] <expr3> == | EQ | LT | GT <expr4>  
 [AND | OR | XOR ...]] [THEN] <command>

Function: Executes a command if the given equation is true.

EQ → Equivalence (equality)

LT → Less than

GT → Greater than

**IFF** (internal, 2.41)

Format: IFF [NOT] EXIST [d:] [<path>] <filename> [THEN] <command>  
 :  
 ENDIFF [<command>]  
 or  
 IFF [NOT] <expr1> == | EQ | LT | GT <expr2> [AND | OR |  
 XOR [NOT] <expr3> == | EQ | LT | GT <expr4>  
 [AND | OR | XOR ...]] [THEN] <command>  
 :  
 ENDIFF [<command>]

Function: Turn on Command Mode if the given equation is true and turn off otherwise.

EQ → Equivalence (equality)

LT → Less than

GT → Greater than

**INKEY** (internal, 2.41)

Format: INKEY [<string>] %%<environment variable>

Function: Reads the value of a key pressed and stores the value read in the <environment variable>.

**INPUT** (internal, 2.41)

Format: INPUT [<string>] %%<environment variable>

Function: Reads a string from the keyboard or device and stores the value read in the <environment variable>.

**KMODE** (external, 2-K)

Format: KMODE [mode | OFF] [/S] [d:]

Function: Select or turn off the Kanji mode or update the boot to automatically install the Kanji driver.

[/S] Updates the boot code for the [d:] drive.

**LOCK** (external, Nextor)

Format: LOCK [<drive letter>: [ON | OFF]]

Function: Lock or unlock drive letters, or display the list of locked drives.

**MAPDRV** (external, Nextor)

Format: MAPDRV [/L] <drive>: <partition> | d | u [<disp index> – <LUN index>] [<driver slot> [– <subslot of the driver>]]]

Function: Maps a drive unit in the Nextor system.

[/L] Locks the unit right after mapping

<drive> drive letter to be mapped

<partition>: 0 – the drive will be mapped from the device's absolute zero sector.

1 – First primary partition

2 to 4 – Refer to extended partitions 2.1 to 2.4 if partition 2 is extended; otherwise, they refer to primary partitions.

5 or more refer to extended partitions.

d – The default unit will be mapped

u – The drive will not be mapped

**MD** (internal, 2)

Format: MD [d:] <path>

MKDIR [d:] <path>

Function: Create a subdirectory

**MEMORY** (internal, 2.41)

Format: MEMORY [/K] [/P]

Function: Displays information about the system's RAM.

[/K] Displays in Kbytes.

[/P] Pause messages when completing a screen.

**MKDIR** (internal, 2)

Format: The same as the MD command.

Function: The same as the MD command.

**MODE** (internal, 1-2.41)

Format: MODE <number of characters> [<lines>]

Function: Changes the number of characters per horizontal line (1, 2 and 2.41) and the number of screen lines (only 2.41).

**MORE** (external, 1-2.41)

Format: <command> | MORE

Function: Display command. The output of the <command> is redirected to the MORE command. At the end of the screen, the display is paused with the message MORE until a key is pressed. <ESC> or <N> abort the command.

**MOVE** (internal, 2)

Format: MOVE [/H] [/P] [/S] <filename> <path>

Function: Moves files to another part of the disk.

[/H] Hidden files will also be moved.

[/S] System files will also be moved (2.41).

[/P] Pause messages when completing a screen.

**MVDIR** (internal, 2)

Format: MVDIR [/H] [/P] <filename> <path>

Function: Moves directories to another part of the disk.

[/H] Hidden directories will also be moved.

[/P] Pause messages when completing a screen.

**NSYSVER** (external, Nextor)

Format: NSYSVER <major version>. <Minor version>

Function: Changes the version number of DOS returned by the system.

**PATH** (internal, 2)

Format: PATH [[+ | -] [d:] <path> [[d:] <path> ...]]

Function: Displays or sets the search path for .COM and .BAT execution files.

+ Delete paths with the same name and recreate them

- Delete the specified paths

Note: without +/-, delete all existing paths and create the specified path.

**PAUSE** (internal, 2)

Format: PAUSE [comment]

Function: Stops the execution of a batch file (batch) until a key is pressed.

**POPD** (internal, 2.41)

Format: POPD [/N]

Function: Retrieves the current drive and directory.  
 [/N] only the last drive/directory are removed from the list.

**PUSHD** (internal, 2.41)

Format: PUSHD [d:] [<path>]

Function: Change the default directory and drive, saving the chains.

**RAMDISK** (internal, 2)

Format: RAMDISK [=] [<size> [K]] [/D]

Function: Displays the size or creates a RAMDISK.  
 [/D] Delete the existing RAMDISK and create another one.

**RALLOC** (external, Nextor)

Format: RALLOC [<drive letter>: [ON | OFF]]

Function: Enables or disables the reduction of allocation space for a drive unit, or displays the list of drives with reduced allocation.

**RD** (internal, 2)

Format: RD [/H] [/P] <filename>

RMDIR [/H] [/P] <filename>

Function: Removes one or more subdirectories.  
 [/H] Hidden files will also be moved  
 [/P] Pause messages when completing a screen

**REM** (internal, 1)

Format: REM [comments]

Function: Insert comments in a batch file.

**REN** (internal, 1-2.41)

Format: REN [/H] [/P] [/S] <filename 1> <filename 2>

RENAME [/H] [/P] [/S] <filename 1> <filename 2>

Function: Rename the file <filename 1> to <filename 2>.  
 [/H] Hidden files will also be renamed  
 [/S] System files will also be renamed (2.41)  
 [/P] Pause messages when completing a screen

**RENAME** (internal, 1)

Format: The same as the REN command.

Function: The same as the REN command.

**RESET** (internal, 2.41)

Format: RESET

Function: Reset the system.

**RETURN** (internal, 2.41)

Format: RETURN [~label]

Function: Returns from a subroutine in a batch file.

**RMDIR** (internal, 2)

Format: The same as the RD command.

Function: The same as the RD command.

**RNDIR** (internal, 2)

Format: RNDIR [/H] [/P] <directory name 1> <directory name 2>

Function: Renames the subdirectory <directory name 1> with <directory name 2>.

[/H] Hidden files will also be renamed.

[/P] Pause messages when completing a screen.

**SET** (internal, 2-2.41)

Format: SET [/P] [name] [=] [value]

Function: Defines or displays environment items.

[/P] Pause messages when completing a screen.

The default values are as follows:

EXPAND = ON (2.41)

SEPAR = ON (2.41)

ALIAS = ON (2.41)

REDE = ON

LOWER = ON (2.41)

UPPER = OFF

ECHO = OFF

EXPERT = ON (2.41)

PROMPT = %\_CWD%> (modified in 2.41)

CDPATH =; (2.41)

PATH =;



TIME = 24  
 DATE = yy-mm-dd  
 TEMP = A:\  
 HELP = A:\ HELP  
 SHELL = A:\ COMMAND2.COM

**SHIFT** (internal, 2.41)

Format: SHIFT [/<number>]

Function: Shifts the arguments of the batch file one position to the left. If /<number> is specified, the argument in this position will be the first to be moved; previous arguments will not be affected.

**THEN** (internal, 2.41)

Format: THEN [<command>]

Function: Executes a command (THEN is ignored).

**TIME** (internal, 1)

Format: TIME [<time>]

Function: Displays or changes the system time.

**TO** (internal, 2.41)

Format: TO <part\_name\_subdirectory> [/N] [/X | F | P | L]

TO [d:] /S [/H]

TO [d:] ...

TO [d:] -n

TO [d:] \

TO [d:] <directory\_name> /M | /C [/H]

TO [d:] <directory\_name> /D

TO [d:] <old name> <new name> /R

TO [d:] <source\_dir> <dest\_dir> /V

Function: Change, create, delete, rename or remove a directory.

[/N] Lists the directories containing <part\_name\_subdir>.

[/C] Create a new directory and enter it.

[/D] Remove directory.

[/F] Searches only at the beginning of the name.

[/H] Makes /S also search for hidden files  
or /M or /C create a hidden directory.

[/L] Searches only at the end of the name.

- [/M] Create a new directory.
- [/P] Searches for the entire name.
- [/R] Rename first directory.
- [/S] Searches all directories and creates the TO.LST file.
- [/V] Moves subdirectory.
- [/X] Only exact names are searched.
- n Level of subdirectories.
- \ Go to the root directory.

### **TREE** (internal, 2.41)

Format: TREE [d:] [<path>] [/P] [/?]

Function: Displays the directory tree on the disk.

- [/P] Pauses the listing when completing a screen.
- [/?] Displays a help screen.

### **TYPE** (internal, 1-2.41)

Format: TYPE [/S] [/H] [/P] [/B] <filename> | ">" <device>  
TYPE <device> <filename>

Function: Displays data from a file / device or create a file from specified device. To end <device> to <filename> entry, press CTRL+Z and RETURN.

- [/S] System files will also be displayed (only 2.41).
- [/H] Hidden files will also be displayed.
- [/P] Pauses the presentation when completing a screen.
- [/B] Disables checking of control codes.

### **TYPEWW** (external, 1-2.41)

Format: TYPEWW <filename> [/S] [/H] [/B]

Function: Displays data from a file. Unlike the TYPE command, <filename> cannot be ambiguous.

- [/S] System files will also be displayed (only 2.41).
- [/H] Hidden files will also be displayed.
- [/P] Pauses the presentation when completing a screen.

### **UNDEL** (external, 2)

Format: UNDEL [<filename>]

Function: Recovers deleted files.

**SEE** (internal, 2)

Format: SEE

Function: Displays the system version.

**VERIFY** (internal, 2)

Format: VERIFY [ON | OFF]

Function: Displays or changes the writing verification status.

**VOL** (internal, 2)

Format: VOL [d:] [<volume name>]

Function: Displays or changes the volume name of the disk.

**XCOPY** (external, 2)

Format: XCOPY [<source filename> [<dest filename>]] [/T] [/A]  
[ /M] [/S] [/E] [/P] [/W] [/V]

Function: Copies files and directories. The options are:

[/T] Changes the date of the copied file to the current one

[/A] Only files with the “file” attribute set are copied.

[/M] Similar to /A, but the “file” attribute is reset after copying.

[/S] Subdirectories are also copied.

[/E] Makes /S create all subdirectories, even empty ones.

[/P] Pause after copying each file.

[/W] Pause after copying some files.

[/V] Checks copied files.

**XDIR** (external, 2)

Format: XDIR [<filename>] [/H]

Function: Lists all files in the current subdirectory, in a tree.

[/H] Hidden files will also be listed.

**Z80MODE** (external, Nextor)

Format: Z80MODE <slot driver> [- <subslot driver>]] [ON | OFF]

Function: Enables or disables Z80 access mode for the specified MSXDOS driver.

## 4.3 – BDOS CALLS

### 4.3.1 – I/O Handling

#### **CONIN** (01H)

Function: Keyboard input.

Setup: None.

Output: A – keyboard character code.

Note: Character input with wait and echo on screen. The following control sequences are checked by this routine:  
CTRL+C → Return the system to the command level.  
CTRL+P → Turn on echo for the printer. Anything written on the screen will be output to the printer.  
CTRL+N → Turns off echo for the printer.  
CTRL+S → Stops displaying characters until a key is pressed.

#### **CONOUT** (02H)

Function: Displays the character contained in the E-register on the screen. The control sequences described above are checked.

Input: E – Character code.

Output: None.

#### **AUXIN** (03H)

Function: External auxiliary device input (modem, for example). The four control sequences are checked.

Input: None.

Output: A – Auxiliary device character code.

#### **AUXOUT** (04H)

Function: Output to external device. This function checks the four control sequences.

Input: E – Code of the character to be sent.

Output: None.

#### **LSTOUT** (05H)

Function: Character output to printer. This function checks the four control sequences.

Input: E – Code of the character to be sent.

Output: None.

**DIRIO** (06H)

Function: String input or output. Does not support control characters, but checks all four control sequences.

Input: E – Character code to be printed on the screen.  
If it is FFH, the character will be received.

Output: If E is FFH on input, the ASCII code of the key will return in A. If A returns 00H, no key was pressed.

**DIRIN** (07H)

Function: Reads a character from keyboard (with wait) and prints to screen. This function does not support control characters.

Input: None.

Output: A – ASCII code of the character.

**INNOE** (08H)

Function: Reads a character from the keyboard (with wait) but does not print to the screen. This function does not support control characters.

Input: None.

Output: A – ASCII code of the character.

**STROUT** (09H)

Function: String output. The 24H ASCII character (\$) marks the end of the string and will not print to the screen. This function checks the four control sequences.

Input: DE – Starting address of the string to be sent.

Output: None.

**BUFIN** (0AH)

Function: String input. The reading of characters ends when the RETURN key is pressed. If the number of characters exceeds the maximum indicated by DE, these will be ignored and a "beep" will be emitted for each extra character. This function checks the four control sequences.

Input: DE must point to a buffer with the following structure:  
DE+0 → number of characters to read.  
DE+1 → number of characters actually read.  
DE+2 onwards: codes of the characters read.

Output: The second byte of the buffer pointed to by DE contains the number of characters actually read and from the third byte onwards are the ASCII codes of the characters read.

**CONST** (0BH)

Function: Check keyboard status. This function checks the four control sequences.

Input: None.

Output: If any key was pressed, register A returns with FFH, otherwise, it returns with the value 00H.

### 4.3.2 - Definition and reading of parameters

**TERM0** (00H)

Function: System reset. When this function is called under DOS, it will cause MSXDOS to reload. When called under DISK-BASIC, it will cause a full reset.

Input: None.

Output: None.

**CPMVER** (0CH)

Function: Reading the system version. In the case of MSX, it will always return the value 0022H, indicating compatibility with CP/M 2.2.

Input: None.

Output: HL – System version.

**DSKRST** (0DH)

Function: Disk reset. All buffers are cleared (FCB, DPB, etc.), the current drive will be A: and DTA will be 0080H.

Input: None.

Output: None.

**SELDSK** (0EH)

Function: Select current drive. The current drive number is stored in address 0004H.

Input: E – Drive number (A:=00H, B:=01H, etc.).

Output: A – Number of available drives (1 to 8).

**LOGIN** (18H)

Function: Reading of drives connected to the computer.

Input: None.

Output: HL – Connected drives

H → Always “00 000 000”

L → 1b71b61b51b41b31b21b11b01

drive: 1H:1G:1F:1E:1D:1C:1B:1A:1

The bit will contain 0 if the drive is not connected and 1 if it is. If B: = 1 and A: = 0 (b1=1 and b0=0), it means that there is just one physical drive connected as A: and B:

**CURDRV** (19H)

Function: Reading the current (current) drive.

Input: None.

Output: A – Current drive number (A:=00H; B:=01H, etc.).

**SETDTA** (1AH)

Function: Sets the address for data transfer.

Input: DE – Start address of DTA (Disk Transfer Address).

Output: None.

Note: At system reset, DTA is set to 0080H.

**ALLOC** (1BH)

Function: Reading information about the disk.

Input: E – Desired drive number (0=current; 1=A:; etc.).

Output: A = FFH → Drive specification is invalid; otherwise:

A – Number of logical sectors per cluster;

BC – Sector size in bytes (typically 512);

DE – Total number of clusters on disk;

HL – Number of free (unused) clusters;

IX – Starting address of DPB in RAM;

IY – Starting FAT address in RAM.

**GDATE** (2AH)

Function: Returns the system date.

Input: None.

Output: HL – Year (1980 to 2079);

D – Month (1=January, 2=February, etc.);

E – Day of the month (1 to 31)

A – Day of the week (0=Sunday, 1=Monday, etc.).

**SDATE** (2BH)

Function: Modify system date.

Input: HL – Year (1980 to 2079);

D – Month (1=January, 2=February, etc.);

E – Day of the month (1 to 31).

Output: A = 00H → Date specification is valid;  
FFH → The specification is invalid.

**GTIME** (2CH)

Function: Returns system time.

Input: None.

Output: H – Hours;

L – Minutes;

D – Seconds;

E – Hundredths of a second.

**TIME** (2DH)

Function: Modify system time.

Input: H – Hours;

L – Minutes;

D – Seconds;

E – Hundredths of a second.

Output: A = 00H → The time specification is valid;  
FFH → The specification is invalid.

**VERIFY** (2EH)

Function: Disk write check.

Input: E = 0 → Disables disk write verification mode.

E ≠ 0 → Enable disk write check.

Output: None.

**4.3.3 – Absolute reading/writing of sectors****RDABS** (2FH)

Function: Read logical sectors from disk. The sectors read are placed from the DTA.

Input: DE – Number of the first logical sector to read;

H – Number of sectors to read;

L – Drive number (0=A:, 1=B:, etc.).

Output: A = 0 → The reading was successful;  
A ≠ 0 → Error code.



**WRABS** (30H)

Function: Writing of logical sectors to disk. The data to be written to disk will be read into RAM from DTA.

Input: DE – Number of the first logical sector to be written;  
H – Number of sectors to write;  
L – Drive number (0=A:, 1=B:, etc.).

Output: A = 0 → The writing was successful;  
A ≠ 0 → Error code.

**4.3.4 – Accessing files by using FCB****FOPEN** (0FH)

Function: Open file (FCB).

Input: DE – Start address of an unopened FCB.

Output: A = 0 → The operation was successful;  
A ≠ 0 → Error code.

**FCLOSE** (10H)

Function: Close file (FCB).

Input: DE – Start address of an open FCB.

Output: A = 0 → The operation was successful;  
A ≠ 0 → Error code.

**SFIRST** (11H)

Function: Search for the first file. This function accepts wildcard characters (\* and ?).

Input: DE – Start address of an unopened FCB.

Output: A = 0 → The file was found;  
A ≠ 0 → The file was not found.

**SNEXT** (12H)

Function: Search for the next file. This function accepts wildcard characters (\* and ?).

Input: None.

Output: A = 0 → The file was found;  
A ≠ 0 → The file was not found.

**FDEL** (13H)

Function: Delete files. Wildcard characters (\* and ?) can be used.

Input: DE – Start address of an open FCB.

Output: A = 0 → The operation was successful;  
A ≠ 0 → Error code.

**RDSEQ** (14H)

Function: Sequential reading.

Input: DE – Start address of an open FCB.

Current block in FCB – Initial block for reading.

Current record in FCB – Initial record for reading.

Output: A = 0 → The reading was successful;  
A ≠ 0 → Error code.

**WRSEQ** (15H)

Function: Sequential writing.

Input: DE – Start address of an open FCB.

Current block in FCB – Initial block for writing.

Current record in FCB – Initial record for writing.

Initial 128 bytes of DTA – Data to be written.

Output: A = 0 → The writing was successful;  
A ≠ 0 → Error code.

**FMAKE** (16H)

Function: Create files.

Input: DE – Start address of an unopened FCB.

Output: A = 0 → The operation was successful;  
A ≠ 0 → Error code.

**FREN** (17H)

Function: Rename files. The wildcard character "?" can be used to rename multiple files simultaneously.

Input: DE – FCB start address with the name of the file to be renamed. In the first position of the FCB, the drive number must be placed followed by the name of the file to be renamed. From the 18th byte (FCB+11H) to the 28th, the new filename must be entered.

Output: A = 0 → The renaming was successful;  
A ≠ 0 → Error code.

**RDRND** (21H)

Function: Random reading. The read record will be placed in the area indicated by the DTA and has a fixed size of 128 bytes.

Input: DE – Start address of an open FCB.  
Random register in FCB – number of the register to read.

Output: A = 0 → The reading was successful;  
A ≠ 0 → Error code.

**WRRND** (22H)

Function: Random writing.

Input: DE – Start address of an open FCB.  
Random register in FCB – Register number to be written.  
128 bytes from DTA – Data to be written.

Output: A = 0 → The writing was successful;  
A ≠ 0 → Error code.

**FSIZE** (23H)

Function: Read file size. The size returns in the first three bytes in the FCB's random file size field in 128-byte increments. So, if a file contains 1 to 128 bytes, the returned value will be 1; if it contains 129 to 256 bytes, the value is 2, and so on.

Input: DE – Start address of an open FCB.

Output: A = 0 → The operation was successful;  
A ≠ 0 → Error code.

**SETRND** (24H)

Function: Set field of random record.

Input: DE – Start address of an open FCB.  
Current block in FCB – Number of desired block.  
Current FCB record – Number of desired record.

Output: The desired current register position, calculated from the register and block contained in the FCB, is placed in the random register field. The first three random record bytes contain valid values.

**WRBLK** (26H)

Function: Block random writing. The random record number is automatically incremented after writing, and its size can range from 1 to 65 535 bytes.

Input: DE – Start address of an open FCB.  
 HL – Number of records to be written.  
 DTA – Data to be written.  
 FCB record size – Record size to be written.  
 FCB random record – First record number to be written.

Output: A = 0 → The operation was successful;  
 A ≠ 0 → Error code.

**RDBLK** (27H)

Function: Block random access.

Input: DE – Start address of an open FCB.  
 HL – Number of records to read.  
 DTA – Starting address for the read data.  
 FCB record size – Record size to be read.  
 FCB random record – First record number to be read.

Output: A = 0 → The reading was successful;  
 A ≠ 0 → Error code.  
 HL – Number of records actually read if end of file is reached before all records are read.

**WRZER** (28H)

Function: Random writing with 00H bytes. This function is the same as 22H (WRRND), except that it fills the remaining records of the file with 00H bytes if the specified record is not the last one in the file.

Input: DE – Start address of an open FCB.  
 Random record in FCB – Record to be written.  
 128 bytes from DTA – Data to be written.

Output: A = 0 → The writing was successful;  
 A ≠ 0 → Error code.

**4.3.5 – Functions added by MSXDOS2****DPARAM** (31H)

Function: Read parameters from disk.

Input: DE – Start address of a 32-byte buffer.  
 L – Drive number (0=current, 1=A:, etc).

Output: A – Error code (if it is 0, there was no error).  
 DE – Start address of the parameter buffer.

|        |                                     |
|--------|-------------------------------------|
| +0     | Physical drive number (1=A:, etc.). |
| +1~2   | Sector size in bytes (usually 512). |
| +3     | Number of sectors per cluster.      |
| +4~5   | Number of reserved sectors.         |
| +6     | Number of FATs (usually 2).         |
| +7~8   | Number of directory entries.        |
| +9~10  | Total number of logical sectors.    |
| +11    | Disk ID.                            |
| +12    | Number of sectors per FAT.          |
| +13~14 | First sector of the directory.      |
| +15~16 | First sector of data area.          |
| +17~18 | Maximum number of clusters.         |
| +19    | Dirty disk flag.                    |
| +20~23 | Volume ID (-1 = No volume ID).      |
| +24~31 | Reserved (usually 0).               |

### **FFIRST** (40H)

Function: Search for first entry in directory.

Input: DE – Initial address of the FIB or an ASCII string "drive/path/file", which may contain the wildcard characters "?" and "\*".

HL – Starting address of filename (only when DE points to FIB).

B – Attributes to search (same as directory).

IX – Starting address of a new FIB.

Output: A – Error code (if it is 0, there was no error).

IX – Starting address of the new filled FIB.

### **FNEXT** (41H)

Function: Searches for next directory entry. This function should only be used after the 40H function. It accepts the wildcard characters "?" and "\*" set to 40H and searches for all files that have equal parts of their name specified by wildcard characters, one after another.

Input: IX – FIB start address.

Output: A – Error code (if it is 0, there was no error).

IX – Starting address of the new filled FIB.

**FNEW** (42H)

Function: Search for new entry.

- Input: DE – Starting address of the FIB or an ASCII string "drive/path/file". If there are wildcard characters in the filename, they will be replaced with appropriate characters. If the "directory" bit is set on the input (register B), a subdirectory will be created. The other bits will be copied.
- HL – Starting address of a filename (only if DE points to FIB).
- B – b0~b6 → Attributes;  
b7 → Create new flag.
- IX – Start address of new FIB containing default filename.
- Output: A – Error code (if 0, there was no error)
- IX – Starting address of the FIB filled with the new entry.

**OPEN** (43H)

Function: Open handle file. If the "inheritable" bit of A is set, the handle file must be opened by another process (see function 60H).

- Input: DE – FIB start address or ASCII string "drive/path/file".
- A – Open mode:  
b0=1 – Not written;  
b1=1 – Not reading;  
b2=1 – Inheritable;  
b3~b7 – Must be "0".
- Output: A – Error code (if it is 0, there was no error).
- B – New handle file.

**CREATE** (44H)

Function: Create handle file. The file created by this function will automatically open (function 43H)

- Input: DE – Drive/path/file or ASCII string.
- A – Open mode:  
b0=1 – Not written;  
b1=1 – Not reading;  
b2=1 – Inheritable;  
b3~b7 – Must be "0".
- B – b0~b6 = Attributes;  
b7 = Create new flag.

Output: A – Error code (if it is 0, there was no error).  
 B – New handle file.

### **CLOSE** (45H)

Function: Close file handle.

Input: B – File handle to close.

Output: A – Error code (if it is 0, there was no error).

### **ENSURE** (46H)

Function: Protect file handle (the current file pointer cannot be modified).

Input: B – Handle file to be protected.

Output: A – Error code (if it is 0, there was no error).

### **DUP** (47H)

Function: Duplicate handle file.

Input: B – Handle file.

Output: A – Error code (if it is 0, there was no error).  
 B – New handle file.

### **READ** (48H)

Function: Read from handle file. The four control sequences (Ctrl+P, Ctrl+N, Ctrl+S and Ctrl+C) are checked.

Input: B – Handle file.

DE – Start address of the buffer.

HL – Number of bytes to read.

Output: A – Error code (if it is 0, there was no error).  
 HL – Number of bytes actually read.

### **WRITE** (49H)

Function: Write by a handle file. If the end of file is found, it will be extended up to the required value.

Input: B – Handle file.

DE – Start address of the buffer.

HL – Number of bytes to write.

Output: A – Error code (if it is 0, there was no error).  
 HL – Number of bytes actually written.

**SEEK** (4AH)

Function: Move handle file pointer.

Input: B – Handle file.

A – Method code:

0 – Relative to the beginning of the file;

1 – Relative to the current position;

2 – Relative to the end of the file.

DE:HL – Offset signalling.

Output: A – Error code (if it is 0, there was no error).

DE:HL – New file pointer.

**IOCTL** (4BH)

Function: Control for I/O devices.

Input: B – Handle file.

A – Subfunction code:

00H – Read status from handle file;

01H – Set ASCII/binary mode;

02H – Tests if device. is ready for entry;

03H – Tests if device. is ready for exit;

04H – Calculates screen size.

DE – Other parameters.

Output: A – Error code (if it is 0, there was no error).

DE – Other return values.

Note: If A equals 0 on input, then the DE register must be loaded with the following parameters:

→ For devices:

b0=1 – Input device;

b1=1 – Output device;

b2~b4 – Reserved;

b5=1 – ASCII mode;

=0 – Binary mode;

b6=1 – End of file;

b7=1 – Device (always 1);

b8~b15 – Reserved.

→ For files:

b0~b5 – Drive number (0=A; etc.);

b6=1 – End of file;

b7=0 – Disk file (always 0);

b8~b15 – Reserved.



On return, DE will have the same values. If A=1, only bit 5 of DE must be specified; other bits will be ignored. If A is equal to 2 or 3, register E will return with the value 00H if the device is not ready and with FFH if the device is ready. If A equals 4, DE returns the logical screen size value for the handle file (D=number of rows and E=number of columns). For devices other than the screen, DE will return with the value 0000H.

### **HTEST (4CH)**

Function: Test handle file.

Input: B – Handle file.

DE – Pointer to FIB or to ASCII string "drive/path/file".

Output: A – Error code (if it is 0, there was no error).

B – 00H = not the same file;

FFH = Is the same file.

### **DELETE (4DH)**

Function: Delete file or subdirectory. A subdirectory can only be deleted if it does not contain any files. If a device name is specified it will not return an error, but of course the device will not be "erased".

Input: DE – Pointer to FIB or to ASCII string "drive/path/file".

Output: A – Error code (if it is 0, there was no error).

### **RENAME (4EH)**

Function: Rename file or subdirectory.

Input: DE – Pointer to FIB or to ASCII string "drive/path/file".

HL – Pointer to the new ASCII name.

Output: A – Error code (if it is 0, there was no error).

### **MOVE (4FH)**

Function: Move file or subdirectory. A file cannot be moved if the respective file handle is open. The FIB of the moved file will not be updated.

Input: DE – Pointer to FIB or to ASCII string "drive/path/file".

HL – Pointer to new ASCII string path.

Output: A – Error code (if it is 0, there was no error).

**ATTR** (50H)

Function: Set or read attributes of a file. The attributes of a file cannot be modified if the corresponding handle file is open.

Input: DE – Pointer to FIB or to ASCII string "drive/path/file".

A = 0 – Read attributes;

1 – Write attributes.

L – New attribute byte (if A = 1).

Output: A – Error code (if it is 0, there was no error).

L – Current attribute byte.

**FTIME** (51H)

Function: Read or set date and time in a file.

Input: DE – Pointer to FIB or to ASCII string "drive/path/file".

A = 0 – Read date and time;

1 – set date and time.

IX – New time (if A=1).

HL – New date (if A=1).

Output: A – Error code (if it is 0, there was no error).

DE – Time of current file.

HL – Current file date.

**HDELET** (52H)

Function: Delete handle file. If there is another handle file open for the same file, it cannot be deleted.

Input: B – Handle file.

Output: A – Error code (if it is 0, there was no error).

**HRENAM** (53H)

Function: Rename handle file. The file cannot be renamed if there is another file handle open for the same file. This function is identical to function 4EH, except that the HL register cannot point to a FIB.

Input: B – Handle file.

HL – New ASCII filename.

Output: A – Error code (if it is 0, there was no error).

**HMOVE** (54H)

Function: Move handle file. The file cannot be moved if there is another file handle open for the same file. This function is identical to function 4FH, except that the HL register cannot point to a FIB.

Input: B – Handle file.  
 HL – New path in ASCII.  
 Output: A – Error code (if it is 0, there was no error).

**HATTR** (55H)

Function: Read or set handle file attributes. Attribute byte cannot be modified if there is another handle file opened for the same file.

Input: B – Handle file.  
 A = 0 – read attributes;  
       1 – set attributes.  
 L – New attribute byte (if A=1).  
 Output: A – Error code (if it is 0, there was no error).  
 L – Current attribute byte.

**HFTIME** (56H)

Function: Read or change time and date from handle file. If there is another handle file open for the same file, the date and time cannot be changed. This function is identical to function 51H, except that there is no pointer; only the handle file.

Input: B – Handle file.  
 A = 0 – Read date and time;  
       1 – Set date and time.  
 IX – New time (if A=1).  
 HL – New date (if A=1).  
 Output: A – Error code (if it is 0, there was no error).  
 DE – Current file time.  
 HL – Current file date.

**GETDTA** (57H)

Function: Read start address of the DTA (Disk Transfer Area).

Input: None.  
 Output: DE – DTA start address.

**GETVFY** (58H)

Function: Read write verification flag.  
 Input: None.  
 Output: B = 0 – Write check disabled;  
       1 – Write check enabled.

**GETCD** (59H)

Function: Read current directory or subdirectory.

Input: B – Drive number (0=current; 1=A:, etc.).

DE – Start address of a 64-byte buffer.

Output: A – Error code (if it is 0, there was no error).

DE – Points to the filled buffer. The drive name and "\" character are not included. If there is no current directory, the buffer will be filled with 00H bytes.

**CHDIR** (5AH)

Function: Change current subdirectory.

Input: DE – ASCII string "drive/path/name".

Output: A – Error code (if it is 0, there was no error).

**PARSIS** (5BH)

Function: Parses pathname (path name).

Input: B – Volume name flag (bit 4).

bit4 = 0 → string "drive/path/file"

1 → string "drive/volume"

DE – ASCII string for analysis.

Output: A – Error code (if it is 0, there was no error).

DE – Pointer to the ending character.

HL – Pointer to the beginning of the last item.

B – Analysis flags.

b0=1 if any character points to another drive name;

b1=1 if any path directory is specified;

b2=1 if drive name is specified;

b3=1 if master file is specified in the last item;

b4=1 if filename extension is specified in the last item;

b5=1 if the last item is ambiguous;

b6=1 if the last item is "." or "..";

b7=1 if the last item is "...".

C – Logical drive (1=A:, etc.).

Note: The value returned in HL will point to the first character of the last item in the string. For example, for the string "A:\XYZ\P.Q /F", DE will point to the white space before "/F" and HL will point to "P".

**PFILE** (5CH)

Function: Parse filename.

Input: DE – ASCII string to be parsed, no drive specification.  
 Wildcard characters (? and \*) can be used.

HL – Pointer to an 11-byte buffer.

Output: A – Always 00H.

DE – Pointer to the final character.

HL – Pointer to the filled buffer.

B – Analysis flags. The values are identical to the 5BH function, except that bits 0, 1 and 2 will always be 0.

**CHKCHR** (5DH)

Function: Check character. 16-bit characters are also checked.

Input: D – Character flags.

b0=1 to suppress the character. In this case, the character returned in E will always be the same.

b1=1 if it is the first byte of a 16-bit character;

b2=1 if it is the second byte of a 16-bit character;

b3=1 if it is volume name or preferably filename;

b4=1 if it is invalid file/volume character;

b5~b7 are reserved (always 0).

E – Character to be checked.

Output: A – Always 00H.

D – Updated character flags.

E – Checked character.

**WPATH** (5EH)

Function: Read complete string path, without drive specification and “\” character. For greater reliability, call function 40H or 41H first and then call WPATH twice, as other functions may change the data.

Input: DE – Pointer to a 64-byte buffer.

Output: A – Error code (if it is 0, there was no error).

DE – Buffer filled with the complete path string.

HL – Pointer to the beginning of the last item.

**FLUSH** (5FH)

Function: Unload disk buffers.

Input: B – Drive specification (0=current; 1=A; etc. If FFH, unload all drives).  
 D = 00H – Unload only;  
 FFH – Unload and invalidate.  
 Output: A – Error code (if it is 0, there was no error).

**FORK** (60H)

Function: Branch files into tree.

Input: None.

Output: A – Error code (if it is 0, there was no error).  
 B – Branch process ID.

**JOIN** (61H)

Function: Join files in tree. This function returns to the original handle file the handle file copied by the previous function. The copied file is automatically closed and the original handle file is reactivated.

Input: B – Branch process ID (or 0).

Output: A – Error code (if it is 0, there was no error).  
 B – Branch primary error code.  
 C – Branch secondary error code.

**TERM** (62H)

Function: End with error code.

Input: B – Error code for termination.

Output: None.

**DEFAB** (63H)

Function: Set abort routine. Only available if called by 0005H.

Input: DE – Starting address of the abort routine; the default address is 0000H.

Output: A – Always 00H.

**DEFER** (64H)

Function: Set user routine for disk error.

Input: DE – Start address of disk error routine. The default value is 0000H.

Output: A – Always 00H.

Note: The specification of parameters and results of the created routine are as follows:

Input: A – Error code;  
 B – Physical drive number;  
 C – b0=1 if writing error;  
       b1=1 if you ignore the error (not recommended);  
       b2=1 if automatic abort is suggested;  
       b3=1 if the sector number is valid.  
 DE – Disk sector number (if b3 of C is 1).

Output: A = 0 – Call system error routine;  
        1 – Abort;  
        2 = Try again;  
        3 = Ignore.

### **ERROR** (65H)

Function: Catch error code in advance to prevent the kind of error that might occur on the next function call.

Input: None.

Output: A – Always 00H.  
 B – Function error code.

### **EXPLN** (66H)

Function: Return error code message.

Input: B – Error code.  
 DE – Pointer to a 64-byte buffer.

Output: A – Always 00H.  
 B – Error code or 00H.  
 DE – Buffer filled with error message in ASCII format.

### **FORMAT** (67H)

Function: Format a disk.

Setup: B – Drive number (0=current; 1=A:, etc.).  
 A = 00H – Return choice message;  
       01H~09H – Formats with this choice;  
       0AH~0DH – Illegal;  
       FEH – Update parameters for MSXDOS2;  
       FFH – Full update for MDXDOS2.  
 HL – Pointer to the buffer (if A = 1~9).  
 DE – Buffer size (if A = 1~9).

Output: A – Error code (if it is 0, there was no error).  
 B – Chosen message slot (only if A=0 on input).  
 HL – Address of the chosen message (only if A=0).

### **RAMD** (68H)

Function: Create or delete ramdisk on drive “H”.

Input: B – 00H = Clear the ramdisk;  
           01H~FEH = Create new ramdisk with xxH 16K logical segments.  
           FFH = Returns ramdisk size in 16K segments.

Output: A – Error code (if it is 0, there was no error).  
 B – Ramdisk size.

### **BUFFER** (69H)

Function: Allocate buffers (each is 16K).

Input: B = 0 – Returns number of allocated buffers;  
           1 to 20 – allocates the specified number of buffers.  
           21 or more (more than 15H) – Invalid

Output: A – Error code (if it is 0, there was no error).  
 B – Total number of allocated buffers

### **ASSIGN** (6AH)

Function: Assign logical drive to a physical drive.

Input: B – Logical drive number (1=A:, 2=B:, etc.):  
           0 – Cancel all assignments (for D = 0);  
           1 to 7 – Assign/cancel respective logical drive;  
 D – Physical drive number (1=A:, 2=B:, etc.).  
           0 – Cancel assignment (for B = 1 to 7):  
           1 to 7 – Assign respective physical drive;  
           FFH – Only return logical drive on D.

Output: A – Error code (if it is 0, there was no error).  
 D – Physical drive number.

### **GENV** (6BH)

Function: Read external item.

Input: HL – Pointer to ASCII string name.  
 DE – Buffer pointer to value string.  
 B – Buffer size. If the buffer is small, the return value will be truncated and terminated in 00H. A 255-byte buffer will always sufficient.



Output: A – Error code (if it is 0, there was no error).  
 DE – Pointer to the filled buffer.

### **SENV** (6CH)

Function: Set external item.

Input: HL – Pointer to ASCII name.  
 DE – Pointer to the value string to be set. Must be up to 255 characters long and end in 00H. If the string is null, the outer item will be removed.

Output: A – Error code (if it is 0, there was no error).

### **FENV** (6DH)

Function: Search for external item.

Input: DE – External item number.  
 HL – Buffer pointer to ASCII name.

Output: A – Error code (if it is 0, there was no error).  
 HL – Pointer to the filled buffer, the end of which is marked with a 00H byte.

### **DSKCHK** (6EH)

Function: Enable or disable disk checking. When the check is active, the system will reload boot, FAT, FIB, FCB, etc. From the disk every time it is changed.

Input: A = 00H – Read check value from disk;  
           01H – Set disk check value.  
 B = 00H – Active (if A=01H);  
           01H – Disables (if A=01H).

Output: A – Error code (if it is 0, there was no error).  
 B – Current disk check value.

### **DOSVER** (6FH)

Function: Read MSXDOS version number. Values returned in registers BC and DE will be in BCD. So if the version is 2.34, the returned value will be 0234H. For compatibility with MSXDOS1 check first if there was any error (A≠0).

Input: None.

Output: A – Error code (if it is 0, there was no error).  
 BC – DOS Kernel version.  
 DE – MSXDOS2.SYS version.

**REDIR** (70H)

Function: Read or set redirection state. The effect of this function is temporary, in the case of A=01H and B=00H at the input.

Input: A = 00H – Read redirection status;  
01H – Set redirection status.

B – New state:  
b0 – Standard input;  
b1 – Standard output.

**4.3.6 – Functions added by NEXTOR****FOUT** (71H)

Function: Enables or disables quick STROUT mode. When enabled, only the first 511 characters will be printed.

Input: A = 00H – Get fast STROUT mode;  
01H – Set fast STROUT mode.  
B = 00H – Disable (only if A = 01H);  
FFH – enable (only if A = 01H).

Output: A – Error code (if it is 0, there was no error).  
B – Current fast STROUT mode.

**ZSTROUT** (72H)

Function: Print a zero-terminated string. This function is affected by the quick STROUT mode.

Input: DE – String address.

Output: A = 0 (never returns an error).

**RDDR** (73H)

Function: Read the absolute sectors of the unit. This function is able to read sectors regardless of the file system viewed on the drive (FAT12, FAT16 or an unknown file system), and even when there is no file system. The read sectors will be placed from the current disk's DTA.

Input: A – Unit number (0 = A:, 1=B:, etc.).  
B – Number of sectors to read.  
HL:DE – Sector number.

Output: A – Error code (if it is 0, there was no error).

**WRDRV** (74h)

Function: Write absolute sectors to disk. This function is able to record sectors regardless of the file system viewed on the drive (FAT12, FAT16 or an unknown file system), and even when there is no file system. The sectors will be written from the current disk's DTA.

Input: A – Unit number (0 = A; 1=B; etc.).  
 B – Number of sectors to read.  
 HL:DE – Sector number.

Output: A – Error code (if it is 0, there was no error).

**RALLOC** (75H)

Function: Get or set reduced allocation information mode vector. The vector assigns a bit to each drive; bit 0 of L is for A ; bit 1 of L is for B ; etc. This bit is 1 if the reduced allocation mode is currently enabled (when getting the vector) or to be enabled (when setting the vector) for the drive, 0 when the mode is disabled or to be disabled.

Input: A = 00H – Get current vector;  
 01H – Define vector.  
 HL – New vector (only if A = 01H).

Output: A – 0 (never returns an error).  
 HL – Current vector.

**DSPACE** (76H)

Function: Get disk space information. The extra value in BC will be nonzero only when the unit's minimum allocation unit is not an integer in Kbytes.

Input: E – Unit number (0 = Standard, 1 = A ; etc)  
 A = 00H – Get free space;  
 01H – Get full space.

Output: A – Error code (if it is 0, there was no error).  
 HL:DE – Space in Kbytes.  
 BC – Extra space in bytes.

**LOCK** (77H)

Function: Lock / unlock a unit or get the lock status of a unit. When a drive is locked, Nextor will assume that the media on that drive will never change and therefore will never ask the associated driver for media change status; thus resulting in

an overall increase in media access speed. This is useful when using removable devices as the primary storage device.

- Input: E – Physical unit (0 = A ;, 1 = B ;, etc)  
 A = 00H – Get lock status;  
       01H – Set lock status.  
 B = 00H – Unlock unit (only if A = 01H);  
       FFH – Unit lock (only if A = 01H).
- Output: A – Error code (if 0, no error).  
 B – Current blocking state, same as input.

### **GDRVR (78H)**

Function: Get information about a device driver.

- Input: A – Driver index )0 to specify slot and segment) D driver slot number (only if A = 0).  
 E – Driver segment number, FFH for drivers in ROM (only if A = 0).  
 HL – Pointer to 64-byte data buffer.
- Output: A – Error code (if it is 0, there was no error).  
 HL – Points to buffer filled with driver data.
- +0: Driver slot number.
  - +1: Driver segment number, FFh if the driver is built into a Nextor or MSX-DOS kernel ROM (always FFH in the current version).
  - +2: Number of drive letters assigned to this driver at boot time.
  - +3: First drive letter assigned to this driver at boot time (A: = 0, etc). Not used if no drive is assigned at boot time.
  - +4: Driver Flags:
    - bit 7: 1 → Nextor driver;  
       0 → MSX-DOS driver (built into MSX-DOS kernel ROM).
    - bits 6-3: Not used, always “0000”.
    - bit 2: 1 → Driver implements DRV\_CONFIG routine.
    - bit 1: Not used, always zero.
    - bit 0: 1 → Device-based driver;  
       0 → Drive-based driver.
  - +5: Driver version number (MSB).

- +6: Driver version number (LSB).
- +7: Driver revision number.
- +8: Driver name, left-justified, complete with spaces (32 bytes).
- +40 ~ +63: Reserved (currently always zero).

## **GDLI** (79H)

Function: Get information about a drive letter

Input: A – Physical unit (0 = A :, 1 = B :, etc)

HL – Pointer to 64-byte data buffer

Output: A – Error code (if 0, no error).

HL – Pointer to filled buffer

- +0: Unit Status
    - 0 – Not assigned
    - 1 – Assigned to a storage device connected to a Nextor or MSX-DOS driver
    - 2 – Not used
    - 3 – A file is mounted on the drive
    - 4 – Assigned RAMdisk (other fields will be zero)
  - +1: Driver slot number
  - +2: Driver segment number (FFH if the driver is built into a Nextor or MSX-DOS kernel ROM)
  - +3: Relative drive number within driver (for drive-based drivers only FFH if driver is device-based)
  - +4: Device index (only for device-based drivers; 0 for MSX-DOS drivers)
  - +5: Logical unit index (only for device-based drivers; 0 for MSX-DOS drivers)
  - +6 ~ +9: Device's first sector number (only for device-based drivers; 0 for MSX-DOS drivers)
  - +10 ~ +63: Reserved (currently always zero)
- If a file is mounted on the drive, the information returned in the data buffer will be inserted as follows:
- +1: Drive where mounted file is located: (0 = A:, 1 = B:, etc)
  - +2: Flags:
    - bit0 = 0 – Read and write, 1 – Read only
  - +3: Always 0
  - +4: filename in print format (up to 12 characters, plus a terminating zero)

**GPART** (7AH)

Function: Get information about a device partition. This function only works on device-based drivers.

- Input:
- A – Driver slot number.
  - B – Driver segment number, FFH for drivers in ROM.
  - D – Device index.
  - E – Logical unit index.
  - H – Primary partition number (1 to 4).
  - HL:7 = 0 – Get partition information;
  - 1 – Get the sector number of the device containing the partition table entry.
  - L – Extended partition number (0 for an entry in the primary partition table).
- Output:
- A – Error code (if 0, no error).
  - If partition information is requested:
  - B – Partition type code:
    - 0 – None (specified partition does not exist).
    - 1 – FAT12.
    - 4 – FAT16, less than 32 MB (obsolete).
    - 5 – Extended (handles more than 4 partitions).
    - 6 – FAT16 (CHS).
    - 14 – FAT16 (LBA).
  - C – Partition status byte.
  - HL:DE – Partition absolute sector starting number
  - IX:IY – Partition size in sectors.
  - If the sector number of the partition table entry is requested:
  - HL:DE – Sector number of the device containing the partition table entry.

**CDRVR** (7BH)

Function: Call a routine in a device driver. This function works in MSX-DOS 1 mode.

- Input:
- A – Driver slot number.
  - B – Driver Segment No., FFH for drivers in ROM.
  - DE – Routine address.
  - HL – Pointer to an 8-byte buffer with input values. The order of registers is as follows: F, A, C, B, E, D, L, H.

Output: A – Error code (if 0, no error).  
 BC, DE, HL – Routine results.  
 IX – AF value returned by the routine.

### MAPDRV (7CH)

Function: Map a drive letter to a device driver

Input: A – Physical unit (0 = A :, 1 = B :, etc)  
 B – Action to be taken:  
     0 – Remove drive mapping.  
     1 – Map the drive to its default state.  
     2 – Map the drive using specific mapping data.  
     3 – Mount a file on the drive.

HL – If B=2:

Address of an 8-byte buffer with mapping data  
 with the following structure:

+0 – Driver slot number  
 +1 – Driver segment number (FFH if the driver is  
     embedded in a Nextor kernel ROM)  
 +2 – Device number  
 +3 – Logical unit number  
 +4 ~ +7 – Home sector

If B=3:

Pointer to filename or FIB address.

D – File mount type (if B = 3).  
     0 – Automatic (read only if file has this attribute set,  
     read and write otherwise).  
     1 – Read only.

Output: A – Error code (if 0, no error).

### Z80MODE (7DH)

Function: Enable or disable the Z80's access to a driver. This function  
 works only on MSX Turbo R computers.

Input: A – Driver slot number.  
 B = 00H – Get current Z80 access mode;  
     01H – Set Z80 access mode.  
 D = 00H – Disable Z80 access mode (only if B = 01H);  
     FFH – Enable Z80 access mode (only if B = 01H).

Output: A – Error code (if 0, no error).  
 D – Current Z80 access mode for specified driver (same  
 as input).

**GETCLUS (7EH)**

Function: Get information for a cluster on a FAT drive.

Input: A – Unit number (0 = Standard, 1 = A: etc.).

DE – Cluster number.

HL – Pointer to a 16-byte buffer.

Output: A – Error code (if 0, no error).

HL – Pointer to the filled buffer:

+0 – FAT sector number containing the entry for the cluster (2 bytes).

+2 – Offset in the FAT sector where the entry for the cluster is located (0-511).

+4 – First number of the data sector to which the cluster refers (4 bytes).

+8 – FAT input value for cluster (2 bytes).

+10 – Size of a cluster in sectors for the unit (1 byte).

+11 – Flags (1 byte):

bit 0 = 1 if the drive is FAT12.

bit 1 = 1 if the drive is FAT16.

bit 2 = 1 if the FAT entry to the cluster is an odd entry (FAT12 only).

bit 3 = 1 if the cluster is the last of a file.

bit 4 = 1 if the cluster is free.

bits 5-7: Unused, always zero.

+12 ~ +15: Unused, always zero.

The FAT entry value for cluster has the following meanings:

0 → Free cluster.

0FF8H-0FFFH for FAT12 and FFF8H-FFFFH for FAT16 →

→ Cluster is the last of a file.

Other value → Number of the next cluster where data for a file continues.

**4.4 – MSXDOS ERROR CODES**

50 FIELD overflow

51 Internal error

52 Bad file number

53 File not found

54 File open



- 55 End of file
- 56 Bad filename
- 57 Direct statement in file
- 58 Sequential I/O only
- 59 File not OPEN
- 60 Disk error
- 61 Bad file mode
- 62 Bad drive name
- 63 Bad sector
- 64 File still open
- 65 File already exists
- 66 Disk full
- 67 Too many files
- 68 Write protected disk
- 69 Disk I/O error
- 70 Disk offline
- 71 RENAME across disk

## **4.5 – MSXDOS2 ERROR CODES**

### **4.5.1 – Disk Errors**

- FFH Incompatible disk
- FEH Write error
- FDH Disk error
- FCH Not ready
- FBH Verify error
- FAH Data error
- F9H Sector not found
- F8H Write protected disk
- F7H Unformatted disk
- F6H Not a DOS disk
- F5H Wrong disk
- F4H Wrong disk for file
- F3H Seek error
- F2H Bad file allocation table
- F1H No message
- F0H Cannot format this drive

#### 4.5.2 – MSXDOS Functions Errors

|     |                             |
|-----|-----------------------------|
| DFH | Internal error              |
| DEH | Not enough memory           |
| DDH | -                           |
| DCH | Invalid MSX-DOS call        |
| DBH | Invalid drive               |
| DAH | Invalid filename            |
| D9H | Invalid pathname            |
| D8H | Pathname too long           |
| D7H | File not found              |
| D6H | Directory not found         |
| D5H | Root directory full         |
| D4H | Disk full                   |
| D3H | Duplicate filename          |
| D2H | Invalid directory move      |
| D1H | Read only file              |
| D0H | Directory not empty         |
| CFH | Invalid attributes          |
| CEH | Invalid . or .. operation   |
| CDH | System file exists          |
| CCH | Directory exists            |
| CBH | File exists                 |
| CAH | File already in use         |
| C9H | Cannot transfer above 64K   |
| C8H | File allocation error       |
| C7H | End of file                 |
| C6H | File access violation       |
| C5H | Invalid process id          |
| C4H | No spare file handles       |
| C3H | Invalid file handle         |
| C2H | File handle not open        |
| C1H | Invalid device operation    |
| C0H | Invalid environment string  |
| BFH | Environment string too long |
| BEH | Invalid date                |
| BDH | Invalid time                |
| BCH | RAM disk already exists     |
| BBH | RAM disk does not exist     |

BAH File handle has been deleted  
B9H Internal error  
B8H Invalid sub-function number

#### **4.5.3 – Errors Added by Nextor**

B6H Invalid device driver  
B5H Invalid device or LUN  
B4H Invalid partition number  
B3H Partition is already in use  
B2H File is mounted  
B1H Bad file size  
B0H Invalid cluster number

#### **4.5.4 – End Programs Errors**

9FH Ctrl-STOP pressed  
9EH Ctrl-C pressed  
9DH Disk operation aborted  
9CH Error on standard output  
9BH Error on standard input

#### **4.5.5 – Command Errors**

8FH Wrong version off COMMAND  
8EH Unrecognized command  
8DH Command too long  
8CH Internal error  
8BH Invalid parameter  
8AH Too many parameters  
89H Missing parameter  
88H Invalid option  
87H Invalid number  
86H File for HELP not found  
85H Wrong version of MSX-DOS  
84H Cannot concatenate destination file  
83H Cannot create destination file  
82H File cannot be copied onto itself  
81H Cannot overwrite previous destination file

## 5 – SYMBOLS

### 5.1 – KERNEL ROUTINES

#### 5.1.1 – Kernel Restarts

##### **RST 08H (MSGSLP)** – Message\_Sleep\_And\_Receive

Description: Checks for a new message from another process. If there is no message, the process will be switched into sleep mode, until a message is available. For more information about receiving message, see RST 18H (MSGGET).

Input:      IXl – Receiver process ID (your own one).  
              IXh – Sender process ID (–1 to check any process).  
              IY – Pointer to message buffer (14 bytes).  
 Output:    IXl – 0 → no message available, 1 → msg received.  
              IXh – Sender process ID (if IXl=1).  
 Registers:  AF, BC, DE, HL.

##### **RST 10H (MSGSEND)** – Message\_Send

Description: Sends a message to another process. IXl must contain your own process ID and IXh the ID of the receiver. If the message queue is full or the receiver does not exist, it will not be sent. The message must always be placed between C000H and FFFFH (transfer RAM area) and can have a maximum size of 14 bytes.

Input:      IXl – Sender process ID (your own one).  
              IXh – Receiver process ID.  
              IY – Pointer to the message (1–14 bytes).  
 Output:    IXl – 0 → message queue is full.  
              1 → message has been sent successfully.  
              2 → receiver process does not exist.  
 Registers:  AF, BC, DE, HL.

##### **RST 18H (MSGGET)** – Message\_Receive

Description: Checks for a new message from another process. The message buffer must have a size of 14 bytes and always be placed between C000H and FFFFH (transfer RAM area).

Input: IXI – Receiver process ID (your own one).  
 IXh – Sender process ID (–1 to check any process).  
 IY – Pointer to message buffer (14 bytes).  
 Output: IXI – 0 → no message available, 1 → msg received.  
 IXh – Sender process ID (if IXI=1).  
 Registers: AF, BC, DE, HL.

### **0H (BNKSCL) – Banking\_SlowCall**

Description: Calls a routine, which is placed in the first RAM bank.  
 All registers will be transfered unmodified to and from the routine. The address of the routine has to be specified  
 Input: (SP+0) – Destination address.  
 AF, BC, DE, HL, IX, IY – Registers for the dest routine.  
 Output: AF, BC, DE, HL, IX, IY – Registers from the dest routine.  
 Registers: –  
 Example: rst 20H : dw 8130H  
 → Calls the routine at 8130H in the first RAM bank.

### **RST 28H (BNKFCL) – Banking\_FastCall**

Description: Calls a routine, which is placed in the first RAM bank.  
 DE, IX and IY will be transfered unmodified to and from the routine. It is faster than RST 20H (BNKSCL). Don't use this function, if the routine does make bank switching or requires more registers than DE, IX, IY.  
 Input: HL – Destination address.  
 DE, IX, IY – Registers for the destination routine.  
 Output: DE, IX, IY – Registers from the destination routine.  
 Registers: AF, BC, HL.  
 Example: ld hl, 08109H : rst 028H  
 → Calls the routine at 8109H in the first RAM bank.

### **RST 30H (MTSOFT) – Multitasking\_SoftInterrupt**

Description: Releases the CPU time for the operating system. If the process currently has nothing to do (it is waiting for something, you should call this function, so that other processes can get CPU time, too. A process, which called this function, is marked as "idle".  
 Input: –  
 Output: –  
 Registers: –

**RST 38H (MTHARD) – Multitasking\_HardInterrupt**

Description: You shouldn't call this function by yourself. It is called by the hardware interrupt, which comes 50 or 300 times per second, depending on the computer system.

Input: –  
 Output: –  
 Registers: –

**5.1.2 – Kernel Commands (Multitasking Management)**

Kernel commands are triggered via a message, which has to be sent with RST 10H (MSGSEND) to the kernel process. The kernel process always has the ID 1.

**ID: 001 (MSC\_KRL\_MTADDP) – Multitasking\_Add\_Process\_Command**

Description: Adds a new process with a given priority and starts it immediately. Application processes usually will be started with priority 4.

Library: SyKernel\_MTADDP

Message: 00 1B 001  
 01 1W Stack address (see notes below).  
 03 1B Priority (1=highest, 7=lowest).  
 04 1B RAM bank (0~15).

Example (stack):

```

        ds 128      ;Stack buffer
stack_ptr:
        dw 0        ;initial value for IY
        dw 0        ;initial value for IX
        dw 0        ;initial value for HL
        dw 0        ;initial value for DE
        dw 0        ;initial value for BC
        dw 0        ;initial value for AF
        dw process_start ;process start adr
        process_id: db 0 ;kernel writes the ID here
  
```

Response: See MSR\_KRL\_MTADDP

**ID: 002 (MSC\_KRL\_MTDelp) – Multitasking\_Delete\_Process\_Command**

Description: Stops an existing process and deletes it.

Library: SyKernel\_MTDelp.

Message: 00 1B 002.  
 01 1B Process ID.  
 Response: See MSR\_KRL\_MTDelp.

**ID: 003 (MSC\_KRL\_MTADDT) – Multitasking\_Add\_Timer\_Command**

Description: Adds a new timer and starts it immediately. Timers will be called 50 or 60 times per second, depending on the screen vsync frequency. Please see MSC\_KRL\_MTADDP for information about the stack.

Library: SyKernel\_MTADDT.  
 Message: 00 1B 003.  
 01 1W Stack address.  
 04 1B RAM bank (0~15).  
 Response: See MSR\_KRL\_MTADDT.

**ID: 004 (MSC\_KRL\_MTDelt) – Multitasking\_Delete\_Timer\_Command**

Description: Stops an existing timer and deletes it.

Library: SyKernel\_MTDelt.  
 Message: 00 1B 004.  
 01 1B Timer ID.  
 Response: See MSR\_KRL\_MTDelt.

**ID: 005 (MSC\_KRL\_MTSLPP) – Multitasking\_Sleep\_Process\_Command**

Description: Puts an existing process into the sleep mode. It is stopped and does not run anymore, until it receives a message, or until it will be wacked up again (see MSC\_KRL\_MTWAKP).

Library: SyKernel\_MTSLPP.  
 Message: 00 1B 005.  
 01 1B Process ID.  
 Response: See MSR\_KRL\_MTSLPP.

**ID: 006 (MSC\_KRL\_MTWAKP) – Multitasking\_WakeUp\_Process\_Command**

Description: Wakes up a process, which was sleeping before. A process will be wacked up, too, when another process is sending a message to it.

Library: SyKernel\_MTWAKP.  
 Message: 00 1B 006.  
 01 1B Process ID.  
 Response: See MSR\_KRL\_MTWAKP

**ID: 007 (MSC\_KRL\_TMADDT) – Timer\_Add\_Counter\_Command**

Description: Adds a counter for a process. You need to specify a byte anywhere in the memory. This byte then will be increased every [P5]/50 seconds. This is much easier and faster than setting up an own timer.

Library: SyKernel\_TMADDT.

Message: 00 1B 007.  
 01 1W Counter byte address.  
 03 1B Counter byte RAM bank (0~15).  
 04 1B Process ID.  
 05 1B Speed (counter will be increased every x/50 secs).

Response: See MSR\_KRL\_TMADDT.

**ID: 008 (MSC\_KRL\_TMDELT) – Timer\_Delete\_Counter\_Command**

Description: Stops the specified counter. Please note, that this will be done automatically, if the process should be deleted.

Library: SyKernel\_TMDELT.

Message: 00 1B 008.  
 01 1W Counter byte address.  
 03 1B Counter byte RAM bank (0~15).

Response: See MSR\_KRL\_TMDELT.

**ID: 009 (MSC\_KRL\_TMDELP) –**

– Timer\_Delete\_AllProcessCounters\_Command

Description: Stops all counters of one process. Please note, that this will be done automatically, if the process should be deleted.

Library: SyKernel\_TMDELP.

Message: 00 1B 009.  
 01 1B Process ID.

Response: See MSR\_KRL\_TMDELP.

**ID: 010 (MSC\_KRL\_MTPRIO) –**

– Multitasking\_Process\_Priority\_Command

Description: Changes the priority of a process. A process is able to change its own priority.

Library: SyKernel\_MTPRIO.

Message: 00 1B 010.  
 01 1B Process ID.  
 01 1B New Priority (1=highest, 7=lowest).

Response: See MSR\_KRL\_MTPRIO.



### 5.1.3 – Kernel Responses (Multitasking Mangement)

Kernel responses are coming as a message, which has to be received with RST 18H (MSGSEND) or RST 08H (MSGSLP) from the Kernel process. which always has the ID 1.

#### **ID: 129 (MSR\_KRL\_MTADDP) – Multitasking\_Add\_Process\_Response**

Description: The kernel sends this message after trying to add a new process (see MSC\_KRL\_MTADDP). You shouldn't add another process until you receive this message.

Message: 00 1B 129.  
 01 1B Error status (0=successful, 1=failed).  
 02 1B Process ID (if P1=0).

#### **ID: 130 (MSR\_KRL\_MTDELP) – Multitasking\_Delete\_Process\_Response**

Description: The kernel sends this message after deleting an existing process (see MSC\_KRL\_MTDELP).

Message: 00 1B 130.

#### **ID: 131 (MSR\_KRL\_MTADDT) – Multitasking\_Add\_Timer\_Response**

Description: The kernel sends this message after trying to add a new timer (see MSC\_KRL\_MTADDT). You shouldn't add another timer until you receive this message.

Message: 00 1B 131.  
 01 1B Error status (0=successful, 1=failed).  
 02 1B Timer ID (if P1=0).

#### **ID: 132 (MSR\_KRL\_MTDELT) – Multitasking\_Delete\_Timer\_Response**

Description: The kernel sends this message after deleting an existing timer (see MSC\_KRL\_MTDELT).

Message: 00 1B 132.

#### **ID: 133 (MSR\_KRL\_MTSLPP) – Multitasking\_Sleep\_Process\_Response**

Description: The kernel sends this message after putting a process into sleep mode (see MSC\_KRL\_MTSLPP).

Message: 00 1B 133.

**ID: 134 (MSR\_KRL\_MTWAKP) –**

– Multitasking\_WakeUp\_Process\_Response

Description: The kernel sends this message after wacking up a process (see MSC\_KRL\_MTWAKP).

Message: 00 1B 134.

**ID: 135 (MSR\_KRL\_TMADDT) – Timer\_Add\_Counter\_Response**

Description: The kernel sends this message after trying to add a new counter (see MSC\_KRL\_TMADDT).

Message: 00 1B 135.

01 1B Error status (0=successful, 1=failed).

**ID: 136 (MSR\_KRL\_TMDELT) – Timer\_Delete\_Counter\_Response**

Description: The kernel sends this message after deleting a counter (see MSC\_KRL\_TMDELT).

Message: 00 1B 136.

**ID: 137 (MSR\_KRL\_TMDELP) –**

– Timer\_Delete\_AllProcessCounters\_Response

Description: The kernel sends this message after deleting all counters of one process (see MSC\_KRL\_TMDELP).

Message: 00 1B 137.

**ID: 138 (MSR\_KRL\_MTPRIO) – Multitasking\_Process\_Priority\_Response**

Description: The kernel sends this message after changing the priority of a process (see MSC\_KRL\_MTPRIO).

Message: 00 1B 138.

**5.1.4 – Kernel Functions (Memory Management)**

All kernel memory functions have to be called with RST 20H (BNKSCL) or RST 28H (BNKFCL).

**MEMSUM (8100H) – Memory\_Summary**

Description: Gives back the size of the total existing memory (=D\*65 536+65 536) and the amount of bytes (=E\*65 536+IX), which are still available.

How to call: ld hl,8100H : rst 028H

Input: –

Output: E, IX – Free memory in bytes.  
 D – Number of existing 64K extended RAM banks.  
 Registers: A, BC, IY.

### **MEMINF** (8121H) – Memory\_Information

Description: Searches for the largest free area inside a 64K bank. If you don't specify the RAM bank (A=0) the system is searching for the largest area inside the whole memory.

How to call: rst 20H : dw 8121H

Input: A – RAM bank (1–15, 0 means search in any bank)  
 E – Memory type:  
     0 – Total (code area).  
     1 – Within a 16K block (data area).  
     2 – Within the last 16K block (transfer area).

Output: BC – Length of the largest free area.  
 A, HL – Total free memory in bytes.

Registers: F, DE

### **MEMGET** (8118H) – Memory\_Get

Description: Reserves the requested amount of memory in any or a special RAM bank. If the memory type is 1, it will be reserved inside a 16k block, if it is 2, inside the last 16K block of the RAM bank.

How to call: rst 20H : dw 8118H.

Input: A – RAM bank (1–15, 0 means search in any bank).  
 E – Memory type:  
     0 – Total (code area).  
     1 – Within a 16K block (data area).  
     2 – Within the last 16K block (transfer area).

Output: BC – Length in bytes.  
 A – RAM bank (1–15).  
 HL – Address.  
 CY – Error state (CY=1 → Not enough memory free).

Registers: BC, DE.

### **MEMFRE** (811BH) – Memory\_Free

Description: Frees the specified memory. Please note, that because of performance and resources reasons the system will free it in any way, so be sure, that you really free only the memory you reserved by yourself.

How to call: rst 20H : dw 811BH.

Input: A – RAM bank (1–15).

HL – Address.

BC – Length in bytes.

Output: –

Registers: AF, BC, E, HL.

### **MEMSIZ (811EH) – Memory\_Resize**

Description: Changes the length of a reserved memory area. You will always have success, if the new length is smaller than the old one.

How to call: rst 20H : dw 811EH.

Input: A – RAM bank (1–15).

HL – Address.

BC – Old length in bytes.

DE – New length in bytes.

Output: CY – Error state (CY=1 → not enough free memory).

Registers: AF, BC, DE, HL.

## **5.1.5 – Kernel Functions (Banking Management)**

Most kernel banking functions have to be called with RST 20H (BNKSCL) or RST 28H (BNKFCL). The interbank functions have to be called directly.

### **BNKRWD (8124H) – Banking\_ReadWord**

Description: Reads a word from an address in any RAM bank.

How to call: rst 20H : dw 8124H.

Input: A – RAM bank (0~15).

HL – Address.

Output: BC – Content of A, HL.

HL – Address+2.

Registers: –

### **BNKWWD (8127H) – Banking\_WriteWord**

Description: Writes a word to an address in any RAM bank.

How to call: rst 20H : dw 8127H.

Input: A – RAM bank (0~15).

HL – Address.

BC – Word.

Output: HL – Address+2.  
 Registers: BC

### **BNKRBT** (812AH) – Banking\_ReadByte

Description: Reads a byte from an address in any RAM bank.

How to call: rst 20H : dw 812AH.

Input: A – RAM bank (0~15).  
 HL – Address.

Output: B – Content of A, HL.  
 HL – Address+1.

Registers: –

### **BNKWBT** (812DH) – Banking\_WriteByte

Description: Writes a byte to an address in any RAM bank.

How to call: rst 20H : dw 812DH.

Input: A – RAM bank (0~15).  
 HL – Address.

B – Byte.

Output: HL – Address+1.

Registers: BC.

### **BNKCOP** (8130H) – Banking\_Copy

Description: Copies a memory area from an address in any RAM bank to any other place in memory. The low nibble of the A register (bit 0–3) specifies the source bank, the high nibble (bit 4–7) the destination bank.

How to call: rst 20H : dw 8130H

Input: A – bit0–3 → Source RAM bank (0~15)  
 bit4–7 → Destination RAM bank (0~15)  
 HL – Source address  
 DE – Destination address  
 BC – Length

Output: –

Registers: AF, BC, DE, HL

### **BNKGET** (8133H) – Banking\_GetBank

Description: Gives back the number of the RAM bank, where the process is running.

How to call: rst 20H : dw 8133H.

Input: –

Output: A – RAM bank (1–15).

Registers: F.

### **BNK16C** (8142H) – Banking\_Call\_Application16KRoutine

Description: Allows you to execute an application routine in the first RAM bank. The routine must be placed inside a 16K block (data RAM area), that will be switched to 4000H–7FFFH, and the routine will be called. After this the old memory configuration will be restored. The application has to relocate the routine by itself first, by setting bit15=0 and bit14=1 for every address pointer. The routine needs an own temporary stack during its execution, that's must be placed in the same 16K block.

How to call: ld hl,8142H : rst 28H.

Input: IX – Pointer to data structure (between C000H–FFFFH).

00 1B Routine RAM bank (0~15).

01 1W Routine address.

03 1W Address of the temporary stack.

DE, IY – Will be handed over unmodified to the routine.

Output: DE, IX, IY – Will be received unmodified by the routine.

Registers: AF, BC, HL.

### **BNKCLL** (FF03H) – Banking\_Interbank\_Call

Description: Switches to a routine into another 64K RAM bank. This allows to have code areas placed in multiple 64K RAM banks and to jump easily between them. The code must be relocated and its stack and transfer area must be placed between C000H and FFFFH as usual.

How to call: call FF03H.

Input: IX – Routine address.

B – Routine RAM bank (0~15).

IY – Address of the routines stack.

DE, HL – Will be handed over unmodified to the routine.

Registers: AF, BC, IY.

**BNKRET (FF00H) – Banking\_Interbank\_Return**

Description: Returns from a routine inside another 64K RAM bank to the caller in the primary bank. See BNKCLL.

How to call: `jp FF00H`.

Input: C, DE, HL, IX – Will be handed over unmodified to the caller.

Registers: AF, B, IY.

**5.1.6 – Kernel Functions (Miscellaneous)**

All miscellaneous kernel functions have to be called with `RST 20H` (BNKSCL) or `RST 28H` (BNKFCL). For more information see KERNEL FUNCTIONS (MEMORY MANAGEMENT).

**MTGCNT (8109H) – Multitasking\_GetCounter**

Description: Gives back the system counter ( $=IY*65536+IX$ ) and the counter of the idle process. The system counter is increased 50 or 60 times per second. The idle process increases its counter every 64 microseconds, when it owns the CPU time.

How to call: `ld hl,8109H : rst 28H`

Input: –

Output: IY, IX – System counter  
DE – Idle counter

Registers: –

**5.2 – DESKTOP MANAGER COMMANDS****ID: 032 (MSC\_DSK\_WINOPN) – Window\_Open\_Command**

Description: Opens a new window. Its data record must be placed in the transfer RAM area (between C000H and FFFFH).

Library: SyDesktop\_WINOPN.

Message: 00 1B 32.

01 1B Window data record RAM bank (0–8).

02 1W Window data record address (C000H–FFFFH).

Response: See MSR\_DSK\_WOPNER and MSR\_DSK\_WOPNOK.

**ID: 033 (MSC\_DSK\_WINMEN)** – Window\_Redraw\_Menu\_Command

Description: Redraws the menu bar of a window. If you changed your menus you should call this command to update the screen display. Works only if window has focus.

Library: SyDesktop\_WINMEN.

Message: 00 1B 033.  
01 1B Window ID.

**ID: 034 (MSC\_DSK\_WININH)** – Window\_Redraw\_Content\_Command

Description: Redraws one, all or a specified number of controls inside the window content. Works only if window has focus.

Library: SyDesktop\_WININH.

Message: 00 1B 034.  
01 1B Window ID.  
02 1B -1 → Control ID or negative number of controls.  
000~239 → The control with the specified ID will be redrawed.  
240~254 → Redraws -P2 controls, starting from control P3. As an example, if P2 is -3 (253) and P3 is 5, the controls 5, 6 and 7 will be redrawed.  
255 → Redraws all controls inside the window content.  
→ If P2 is between 240 and 254:  
03 1B ID of the first control, which should be redrawed.

**ID: 035 (MSC\_DSK\_WINTOL)** – Window\_Redraw\_Toolbar\_Command

Description: Redraws one, all or a specified number of controls inside the window toolbar. Use this command to update the screen display, if you made changes in the toolbar. Works only if window has focus.

Library: SyDesktop\_WINTOL

Message: 00 1B 035  
01 1B Window ID  
02 1B -1 → Control ID or negative number of controls.  
000~239 → The control with the specified ID will be redrawed.



240~254 → Redraws –P2 controls, starting from control P3. As an example, if P2 is –3 (253) and P3 is 5, the controls 5, 6 and 7 will be redrawn.

255 → Redraws all controls inside the window content.

→ If P2 is between 240 and 254:

03 1B ID of the first control, which should be redrawn.

#### **ID: 036 (MSC\_DSK\_WINTIT) – Window\_Redraw\_Title\_Command**

Description: Redraws the title bar of a window. Use this command to update the screen display, if you changed the text of the window title. Works only if window has focus.

Library: SyDesktop\_WINTIT.

Message: 00 1B 036.

01 1B Window ID.

#### **ID: 037 (MSC\_DSK\_WINSTA) – Window\_Redraw\_Statusbar\_Command**

Description: Redraws the status bar of a window. Use this command to update the screen display, if you changed the text of the status bar. Works only if window has focus.

Library: SyDesktop\_WINSTA.

Message: 00 1B 037.

01 1B Window ID.

#### **ID: 038 (MSC\_DSK\_WINMVX) – Window\_Set\_ContentX\_Command**

Description: If the size of the window content is larger than the visible part, you can scroll its X offset with this command. The command works also, if the window is not resizable by the user. Works only if window has focus.

Library: SyDesktop\_WINMVX.

Message: 00 1B 038.

01 1B Window ID.

02 1W New X offset of the visible window content.

#### **ID: 039 (MSC\_DSK\_WINMVY) – Window\_Set\_ContentY\_Command**

Description: If the size of the window content is larger than the visible part, you can scroll its Y offset with this command.

The command works also, if the window is not resizable by the user. Works only if window has focus.

Library: SyDesktop\_WINMVY.

Message: 00 1B 039.

01 1B Window ID.

02 1W New Y offset of the visible window content.

#### **ID: 040 (MSC\_DSK\_WINTOP) – Window\_Focus\_Command**

Description: Takes the window to the front position on the screen.  
Works in all conditions.

Library: SyDesktop\_WINTOP.

Message: 00 1B 040.

01 1B Window ID.

#### **ID: 041 (MSC\_DSK\_WINMAX) – Window\_Size\_Maximize\_Command**

Description: Maximizes a window. A maximized window has a special status, where it can't be moved to another screen position.  
Works only if the window is minimized or restored.

Library: SyDesktop\_WINMAX.

Message: 00 1B 041.

01 1B Window ID.

#### **ID: 042 (MSC\_DSK\_WINMIN) – Window\_Size\_Minimize\_Command**

Description: Minimizes a window. It will disappear from the screen and can only be accessed by the user via the task bar.  
Works only if the window is minimized or restored.

Library: SyDesktop\_WINMIN.

Message: 00 1B 042.

01 1B Window ID.

#### **ID: 043 (MSC\_DSK\_WINMID) – Window\_Size\_Restore\_Command**

Description: Restores the window or the size of the window, if it was minimized or maximized before. Works only if the window is maximized or minimized.

Library: SyDesktop\_WINMID.

Message: 00 1B 043.

01 1B Window ID.

**ID: 044 (MSC\_DSK\_WINMOV) – Window\_Set\_Position\_Command**

Description: Moves the window to another position on the screen.  
Works only, if the window is not maximized.

Library: SyDesktop\_WINMOV.

Message: 00 1B 044.  
01 1B Window ID.  
02 1W New X window position.  
04 1W New Y window position.

**ID: 045 (MSC\_DSK\_WINSIZ) – Window\_Set\_Size\_Command**

Description: Resizes a window. This command will always work, even if the window is not resizeable by the user. Please note, that the size always refers to the visible content of the window, not to the whole window including the control elements. So with title bar, scroll bars etc. a window can have a bigger size on the screen. Works always.

Library: SyDesktop\_WINSIZ.

Message: 00 1B 045.  
01 1B Window ID.  
02 1W New window width.  
04 1W New window height.

**ID: 046 (MSC\_DSK\_WINCLS) – Window\_Close\_Command**

Description: Closes the window. The desktop manager will remove it from the screen. Works in all conditions.

Library: SyDesktop\_WINCLS.

Message: 00 1B 046.  
01 1B Window ID.

**ID: 047 (MSC\_DSK\_WINDIN) –**

– Window\_Redraw\_ContentExtended\_Command

Description: Redraws one, all or a specified number of controls inside the window content. This command is identical with MSC\_DSK\_WININH with the exception, that it always works but with less speed. See MSC\_DSK\_WININH.

Library: SyDesktop\_WINDIN

Message: 00 1B 047  
01 1B Window ID

- 02 1B Control ID, -1 (all) or negative number of controls.  
 000~239 → The control with the specified ID will be redrawn.  
 240~254 → Redraws -P2 controls, starting from control P3. As an example, if P2 is -3 (253) and P3 is 5, the controls 5, 6 and 7 will be redrawn.  
 255 → Redraws all controls inside the window content.  
 → If P2 is between 240 and 254:  
 03 1B ID of the first control, which should be redrawn.

#### **ID: 048 (MSC\_DSK\_DSKSRV) – Desktop\_Service\_Command**

Description: Please read the desktop manager service description below for more information.

Library: See DESKTOP MANAGER SERVICES.

Message: 00 1B 048.

01 1B Service ID.

02-05 See desktop manager service description below.

Response: See MSC\_DSK\_DSKSRV.

#### **ID: 049 (MSC\_DSK\_WINSLD) – Window\_Redraw\_Slider\_Command**

Description: Redraws the two slider of the window, with which the user can scroll the content. Sliders will only be displayed, if the window is resizable. Works if window has focus.

Library: SyDesktop\_WINSLD.

Message: 00 1B 049.

01 1B Window ID.

#### **ID: 050 (MSC\_DSK\_WINPIN) –**

– Window\_Redraw\_ContentArea\_Command

Description: This command works like MSC\_DSK\_WINDIN, but it updates only a specified area inside the window content. Changes outside the area won't be updated. For more information see MSC\_DSK\_WINDIN and MSC\_DSK\_WININH. This command works in all conditions.

Library: SyDesktop\_WINPIN

Message: 00 1B 050

- 01 1B Window ID
- 02 1B Control ID, -1 (all) or negative number of controls  
 000~239 → The control with the specified ID  
 will be redrawn.  
 240~254 → Redraws -P2 controls, starting from  
 control P3. As an example, if P2 is -3  
 (253) and P3 is 5, the controls 5, 6 and 7  
 will be redrawn.  
 255 → Redraws all controls inside the window  
 content.
- 04 1W Area X begin inside the window content.
- 06 1W Area Y begin.
- 08 1W Area X length.
- 10 1W Area Y length.
- If P2 is between 240 and 254:
- 03 1B ID of the first control, which should be redrawn.

#### **ID: 051 (MSC\_DSK\_WINSIN) –**

– Window\_Redraw\_SubControl\_Command

Description: This command works like MSC\_DSK\_WINDIN, but it updates only one sub control inside a control collection. This command currently doesn't support the redrawing of multiple sub controls. For additional information see also MSC\_DSK\_WINDIN. This command works always.

Library: SyDesktop\_WINSIN.

Message: 00 1B 051.  
 01 1B Window ID.  
 02 1B Control collection ID.  
 03 1B ID of the sub control inside the control collection.

### **5.2.1 – Desktop Manager Responses**

#### **ID: 160 (MSR\_DSK\_WOPNER) – Window\_OpenError\_Response**

Description: The window couldn't be opened, because the maximum number of windows (32) has already been reached.

Message: 00 1B 160.

**ID: 161 (MSR\_DSK\_WOPNOK) – Window\_OpenOK\_Response**

Description: The window has been opened. The desktop manager sends back its ID. For all following commands regarding the new window you will need this ID.

Message: 00 1B 161.  
04 1B Window ID.

**ID: 162 (MSR\_DSK\_WCLICK) – Window\_UserAction\_Response**

Description: The desktop manager is sending this message to the application, if the user has done an interaction with the window or the controls inside the window.

Message: 00 1B 162  
01 1B Window ID  
02 1B Action type

- 05 – Close button has been clicked or ALT+F4 has been pressed (DSK\_ACT\_CLOSE).
- 06 – Menu entry has been clicked (DSK\_ACT\_MENU). P8 will contain the menu entry value.
- 14 – A control of the window content has been clicked and/or modified with the keyboard or mouse (DSK\_ACT\_CONTENT). P8 will contain the control value, P4/6 the mouse position, if the user used the mouse.
- 15 – A control of the window toolbar has been clicked and/or modified with the keyboard or mouse (DSK\_ACT\_TOOLBAR). P8 will contain the control value, P4/6 the mouse position, if the user used the mouse.
- 16 – User has pressed a key without modifying any control (DSK\_ACT\_KEY). P4 will contain the ASCII code.

→ If P2 is 14 or 15:

03 1B Action sub specification

- 00 – Left mousebutton clicked (DSK\_SUB\_MLCLICK).
- 01 – Right mousebutton clicked (DSK\_SUB\_MRCLICK).

- 02 – Left mousebutton double clicked (DSK\_SUB\_MDCLICK).
- 03 – Middle mousebutton clicked (DSK\_SUB\_MMCLICK).
- 07 – Key has been pressed (DSK\_SUB\_KEY)
- If P2 is 14 or 15 and P3 is between 0 and 3:
- 04 1W Mouse X position (inside the window content/toolbar).
- 06 1W Mouse Y position.
- If P2 is 14 or 15 and P3 is 7, or if P2 is 16:
- 04 1B ASCII code of the pressed key. For information about extended ASCII codes, see the chapter “device manager”, EXTENDED ASCII CODES.
- If P2 is 6, 14 or 15:
- 08 1W Menu entry value or control value.

#### **ID: 163 (MSR\_DSK\_DSKSRV) – Desktop\_Service\_Response**

Description: Please read the desktop manager service description below for more information.

Message: 00 1B 163.  
 01 1B Service ID.  
 02 -05 See desktop manager service description below.

#### **ID: 164 (MSR\_DSK\_WFOCUS) – Window\_Focus\_Response**

Description: The desktop manager is sending this message to the application, if the focus status of a window changed.

Message: 00 1B 164.  
 01 1B Window ID.  
 02 1B Status: 0 → Window lost focus position.  
 1 → Window received focus position.

#### **ID: 165 (MSR\_DSK\_CFOCUS) – Control\_Focus\_Response**

Description: The desktop manager is sending this message to the application, if another control inside a window got the focus. Please note, that the control ID is not the value of the control but its number inside the control group (starting with 1).

Message: 00 1B 165.  
 01 1B Window ID.

- 02 1B ID of the new focus control (starting with 1).
- 03 1B Reason for focus change:
  - 0 → User clicked the control via mouse or used the mouse wheel.
  - 1 → User pressed the tab key.

#### **ID: 166 (MSR\_DSK\_WRESIZ) – Window\_Resize\_Response**

Description: The desktop manager is sending this message to the application, if the user resized the window. This may happen when it has been maximized, restored or resized by keyboard or mouse. Please note, that this message will also be sent, if the user maximizes or restores a window, which was minimized before.

Message: 00 1B 166.  
01 1B Window ID.

#### **ID: 167 (MSR\_DSK\_WSCROL) – Window\_Scroll\_Response**

Description: The desktop manager is sending this message to the application, if the user scrolled the content of the window.

Message: 00 1B 167  
01 1B Window ID.

### **5.2.2 – Desktop Manager Services**

Most parts of the device manager can't be accessed by an application directly. All video screen related things will be handled by the desktop manager. Because of this there are the desktop services, which allow an application to change some video screen parameters. Also some more services are offered.

#### **ID: 001 (DSK\_SRV\_MODGET) – DesktopService\_ScreenModeGet**

Description: Returns the current screen resolution and number of possible colours.

Library: SyDesktop\_MODGET.

Message: 00 1B 048.  
01 1B 001.

Response: 00 1B 163.  
01 1B 001.



- 02 1B Screen mode; the available modes depend on the computer platform.
- PCW 0 – 720 X 255, 2 colours (PCW standard mode).
- CPC 1 – 320 X 200, 4 colours (CPC standard mode).  
2 – 640 X 200, 2 colours.
- EP 1 – 320 X 200, 4 colours (EP standard mode).  
2 – 640 X 200, 2 colours.
- MSX 5 – 256 X 212, 16 colours.  
6 – 512 X 212, 4 colours.  
7 – 512 X 212, 16 colours (MSX standard mode).
- G9K 8 – 384 X 240, 16 colours.  
9 – 512 X 212, 16 colours (G9K standard mode).  
10 – 768 X 240, 16 colours.  
11 – 1024x 212, 16 colours.
- If G9K:
- 03 1B Virtual desktop width.
- 0 – No virtual desktop.  
1 – 512.  
2 – 1000.

**ID: 002 (DSK\_SRV\_MODSET) – DesktopService\_ScreenModeSet**

Description: Sets the screen resolution and number of possible colors.

Library: SyDesktop\_MODSET

Message: 00 1B 048

01 1B 002

02 1B Bit0~6 Screen mode (the available modes depend on the computer platform):

PCW 0 – 720 X 255, 2 colours (PCW standard mode).

CPC 1 – 320 X 200, 4 colours (CPC standard mode).  
2 – 640 X 200, 2 colours.

EP 1 – 320 X 200, 4 colours (EP standard mode).  
2 – 640 X 200, 2 colours.

MSX 5 – 256 X 212, 16 colours.  
6 – 512 X 212, 4 colours.  
7 – 512 X 212, 16 colours (MSX standard mode).

G9K 8 – 384 X 240, 16 colours.  
9 – 512 X 212, 16 colours (G9K standard mode).  
10 – 768 X 240, 16 colours.  
11 – 1024x 212, 16 colours.

→ If G9K:

- 03 1B Virtual desktop width.
  - 0 – No virtual desktop.
  - 1 – 512.
  - 2 – 1000.

Response: The desktop manager does not send a response message.

#### **ID: 003 (DSK\_SRV\_COLGET) – DesktopService\_ColourGet**

Description: Returns the definition of a colours. Please note, that you always have a range of 4096, even if the computer is not a CPC PLUS, as the system recalculates the colour for the other machines.

Library: SyDesktop\_COLGET.

Message: 00 1B 048.  
 01 1B 003.  
 02 1B Colour number (0~15).

Response: 00 1B 163.  
 01 1B 003.  
 02 1B Colour number (0~15).  
 03 1B bit0~3 → Blue component (0~15).  
           bit4~7 → Green component (0~15).  
 04 1B bit0~3 → Red component (0~15).

#### **ID: 004 (DSK\_SRV\_COLSET) – DesktopService\_ColourSet**

Description: Defines one colour. Please note, that you always have a range of 4096, even if the computer is not a CPC PLUS, as the system recalculates the colour for other machines.

Library: SyDesktop\_COLSET.

Message: 00 1B 048.  
 01 1B 004.  
 02 1B Colour number (0~15)  
 03 1B bit0~3 → Blue component (0~15).  
           bit4~7 → Green component (0~15).  
 04 1B bit0~3 → Red component (0~15).

Response: The desktop manager does not send a response message.

#### **ID: 008 (DSK\_SRV\_DSKBGR) – DesktopService\_RedrawBackground**

Description: Reinitialize and redraws the desktop background.

Library: SyDesktop\_DSKBGR.

Message: 00 1B 048.

01 1B 008.

Response: The desktop manager does not send a response message.

### **ID: 009 (DSK\_SRV\_DSKPLT) – DesktopService\_RedrawComplete**

Description: Reinitialize the desktop background and redraws the complete screen. The background, the task bar and all windows will be updated.

Library: SyDesktop\_DSKPLT

Message: 00 1B 048

01 1B 009

Response: The desktop manager does not send a response message.

## **5.2.3 – Desktop Manager Functions**

The desktop manager functions have to be called with RST 20H (BNKSCL).

### **BUFPUT (814EH) – Clipboard\_Put**

Description: Copies data into the clipboard. If the clipboard already contained data, it will be deleted first.

How to call: rst 20H : dw 814EH

Input:

- IX – Source data address.
- E – Source data RAM bank (0~15).
- IY – Length of source data.
- D – Type of source data.
  - 1 – Text.
  - 2 – Graphic (extended).
  - 3 – Item list (format not yet defined).
  - 4 – Desktop icon shortcut.

Output: CY – Error state (0 → Ok, 1 → Memory full)

Registers: AF, BC, DE, HL.

### **BUFGET (8151H) – Clipboard\_Get**

Description: Copies data from the clipboard to the destination memory area. This will only be done, if the clipboard contains data of the requested type and if the data inside the clipboard is not larger than the destination area.

How to call: rst 20H : dw 8151H

Input: IX – Destination address.  
 E – Destination RAM bank (0~15).  
 IY – Maximum length of destination area.  
 D – Type of required data.

Output: CY – Error state.  
 0 → Ok (IY – Length of copied data).  
 1 → Error: A – 0 → Clipboard is empty,  
 1 → Wrongdata type,  
 2 → Data is too large.

Registers: AF, BC, DE, HL.

#### BUFSTA (8154H) – Clipboard\_Status

Description: Reads the status of the clipboard (data type and length).

The address and bank of the data is returned as well,  
 though an application shouldn't access it directly, as it  
 may be changed by another process in the meantime.

How to call: rst 20H : dw 8154H

Input: –

Output: D – Data type (0 – clipboard is empty).  
 IY – Data length.  
 IX – Data address.  
 E – Data RAM bank (0~15).

Registers: –

### 5.2.4 – Desktop Manager Data Records

If "recalculation" for a control group is activated every coordinate and size value of a control will be recalculated, if the user changes the size of the window. The calculation is:

position or size – Static\_part + window\_size \* multiplier / divider

#### 5.2.4.1 – Window Data Record

00 1B Status (0=closed, 1=normal, 2=maximized, 3=minimized,  
 +128=open window centered, will be always reset after opening).

- 01 1B bit0: Display 8x8 pixel application icon (in the upper left edge).  
 bit1: Window is resizable.  
 bit2: Display close button.  
 bit3: Display tool bar (below the menu bar).  
 bit4: Display title bar.  
 bit5: Display menu bar (below the title bar).  
 bit6: Display status bar (at the lower side of the window).  
 bit7: Used internally (set to 0).
- 02 1B bit0: Adjust X size of the window content to the X size of the window.  
 bit1: Adjust Y size of the window content to the Y size of the window.  
 bit2: Window will not be displayed in the task bar.  
 bit3: Window is not moveable.  
 bit4: Window is a modal window: other windows, who point on it (see byte 51), can't get the focus position.  
 bit5: Reserved (set to 0).  
 bit6: Used internally (set to 0).  
 bit7: Used internally (set to 0).
- 03 1B Process ID of the windows owner
- 04 2W X/Y position, if window is not maximized
- 08 2W X/Y size, if window is not maximized.
- 12 2W X/Y offset of the displayed window content.
- 16 2W Full X/Y length of the total window content.
- 20 2W Minimal possible X/Y size of the window.
- 24 2W Maximal possible X/Y size of the window.
- 28 1W Address of the application icon (graphic object).
- 30 1W Address of the title line text (terminated by 0).
- 32 1W Address of the status line text (terminated by 0).
- 34 1W Address of the MENU DATA RECORD.
- 36 1W Address of the CONTROL GROUP DATA RECORD of the window content.
- 38 1W Address of the CONTROL GROUP DATA RECORD of the tool bar content.
- 40 1W Height of the tool bar.
- 42 9B Used during runtime, so it has to be reserved.
- 51 1B "0" or number of modal window + 1.
- 52 140B Used during runtime, so it has to be reserved.

#### 5.2.4.2 – Control Group Data Record

- 00 1B Number of controls (has to be >0; notice that you have to fill the background of the form by yourself, too!)
- 01 1B Process ID of the control group owner
- 02 1W Address of the CONTROL DATA RECORDS
- 04 1W Address of the position/size CALCULATION RULE DATA RECORD (0 means, no re-calculation)
- 06 2B Not used, set to 0.
- 08 1B Object to click, when user hits return (1~255, 0=not defined; works only for window content, not for the toolbar)
- 09 1B Object to click, when user hits escape (1~255, 0=not defined; works only for window content, not for the toolbar)
- 10 4B Reserved, set to 0.
- 14 1B Focus object (1~255, 0=no focus on any object; only for window content)
- 15 1B Not used, set to 0.

#### 5.2.4.3 – Control Data Records

[Number of controls] \* [

- 00 1W Control ID/value; this will be sent to the application, if the user clicks or modifies the control. As an example you could store the address of a sub routine here, which you call, if the user clicks the control.
- 02 1B CONTROL TYPE; for the type IDs see below. The IDs are between 0 and 63. IDs > 63 will be ignored, so you can set bit 6 and/or 7 to 1, if you want to hide an object, and reset it to 0 if you want to show it again.
- 03 1B Bank number, where the extended control data record is located (0~15); “-1” means, that the control is placed in the same bank like the window data record.
- 04 1W Either a parameter to specify the control properties or, if one word is not enough, a pointer to the extended control data record; this depends on the control, so see the control description for information, what to write here.
- 06 2W X/Y position of the control (related to the upper left edge of the content or tool bar); if the window is using a CALCULATION RULE DATA RECORD, you can write 0 here.

- 10 2W X/Y size of the control (related to the upper left edge of the content or tool bar); if the window is using a CALCULATION RULE DATA RECORD, you can write 0 here
- 14 2B Not used, set to 0.
- ]

#### 5.2.4.4 – Calculation Rule Data Record

- 00 1W X position (static part).
- 02 1B Window X size multiplier.
- 03 1B Window X size divider.
- 04 1W Y position (static part).
- 06 1B Window Y size multiplier.
- 07 1B Window Y size divider.
- 08 1W X size (static part).
- 10 1B Window X size multiplier.
- 12 1B Window X size divider.
- 13 1W Y size (static part).
- 14 1B Window Y size multiplier.
- 15 1B Window Y size divider.

### 5.3 – CONTROL TYPES

#### 5.3.1 – Paint

##### ID: 00 (PLF) – paint\_area

Description: Fills an area with a specified colour.

Parameter: bit0–3: Pen.

bit7: Colour mode:

0 → 4 colour indexed, 1 → 16 colour.

Data record: –

Size: Not limited.

##### ID: 01 (PLT) – paint\_text

Parameter: Pointer to data record.

Data record: 00 1W Text address (terminated by 0).

03 1B bit0–1: Alignment (0=left, 1=right, 2=center).

bit5: If 1, don't prepare background (MSX only).

bit7: Colour mode:

0 → 4 colour indexed, 1 → 16 colour.

→ If 4 colour mode:

02 1B bit0–1: Paper, bit2–3: Pen,  
bit7: If 1, fill background.

→ If 16 colour mode:

02 1B bit0–3: Paper, bit4–7: Pen.  
03 1B bit6: If 1, fill background.

Size: Width is not limited, height must be equal like the height of the current font; if the text is larger than the control width, it will only be cut, if the "fill background" option is activated.

### **ID: 02 (PLR) – paint\_frame**

Description: Plots a text with the standard system font with 4 or 16 colours for background and foreground. If "fill background" is activated first the whole area of the control will be filled with the paper-colour, and the text will be clipped to the defined area. Otherwise it would exceed the area, if it's too long. If the background has already been filled with the paper colour before, bit 5 of byte 3 can be used to increase the performance on the MSX platform.

Parameter: bit7: Colour mode:  
0 → 4 colour indexed, 1 → 16 colour.  
bit6: If 1, fill area inside frame.  
→ If 4 colour mode:  
bit4–5: Pen of area inside frame (only used, if bit6=1).  
bit0–1: Pen of upper and left line.  
bit2–3: Pen of lower and right line.  
→ If 16 Colour mode:  
bit0–3: Pen of area inside frame (only used, if bit6=1).  
bit8–11: Pen of upper and left line.  
bit12–15: Pen of lower and right line.

Data record: –

Size: Equal or greater than 3x3.

### **ID: 03 (PLX) – paint\_frame\_with\_title**

Description: Plots a frame with a text title. Notice, that the lines have a distance of 3 pixels to the border of the control. The area inside the frame will not be filled.



Parameter: Pointer to data record.

Data record: 00 1W Text address (terminated by 0)

02 1B bit7: Colour mode:

0 → 4 colour indexed, 1 → 16 colour.

→ If 4 colour mode:

02 1B bit0–1: Indexed paper of text;

bit2–3: Indexed pen of text and line.

→ If 16 colour mode:

02 1B bit0–3: Pen of line.

03 1B bit0–3: Paper of text;

bit4–7: Pen of text.

Size: Equal or greater than 16x16.

#### **ID: 04 (PLP) – paint\_progress**

Description: Plots a progress bar. The second byte of the parameter specifies the progress in 1/255 steps.

Parameter: bit0–1: Indexed colour of upper and left line.

bit2–3: Indexed colour of lower and right line.

Bit4–5: Indexed colour of filled area inside frame.

bit6–7: Indexed colour of empty area inside frame.

bit8–15: Progress (0=0%, 255=100%).

Data record: –

Size: Equal or greater than 3x3.

#### **ID: 05 (PLA) – paint\_text\_with\_alternative\_font**

Description: Plots a text with an self specified alternative font. The font must be placed in the same 16K area and RAM bank like the text. For the description how a font is stored in the memory see below (FONTS). If "fill background" is activated first the whole area of the ontlrol will be filled with the paper-colour.

Parameter: Pointer to data record.

Data record: 00 1W Text address (terminated by 0).

02 1B bit0–1: Paper, bit2–3: Pen (if 4 colour mode).

bit0–3: Paper, bit4–7: Pen (if 16 colour mode).

03 1B bit0–1: Alignment (0=left, 1=right, 2=center).

bit7: Colour mode:

0 → 4 colour indexed, 1 → 16 colour.

04 1W Font address.

Size: Width is not limited, height must be equal like the height of the current font; if the text is larger than the control width, it will only be cut, if the "fill background" option is activated.

**ID: 06 (PLC) – paint\_text\_with\_control\_codes**

Description: Plots a text, which can include control codes (0–31). The following control codes are currently accepted:

00 – End of text

01 – Set text colour

Parameters: 1byte (bit0–3=paper, bit4–7Pen)

02 – Set font

Parameters: 1word (font address; must be placed in the same 16K area and RAM bank like the text; if the address is –1, the standard font will be used)

03 – Switch underline mode on

04 – Switch underline mode off

05 – Insert additional space between the current and the next char

Parameters: 1byte (amount of pixels)

06 to 07 – \*not yet supported\* (will be ignored)

08 to 11 – Skip next bytes ((code–8)\*2+1 bytes)

12 to 31 – Insert additional space between the current and the next char (code–8 pixels)

Parameter: Pointer to data record

Data record: 00 1W Text address (terminated by 0)

02 1W Maximum number of bytes (control codes included)

04 1W Font address (–1=Standard)

06 1B bit0–3: paper, bit4–7: Pen

07 1B [bit0] =if 1, underlined

Size: Not limited

### 5.3.2 – Graphics

**ID: 08 (ICN) – Graphic\_simple**

Description: Plots a graphic. For the description how a graphic object is stored in the memory see below (GRAPHICS, "Standard graphics"). The control must have the same size like the graphic.

Parameter: Graphic address.

Data record: –

Size: Same as the graphic object.

### **ID: 09 (ICT) – Graphic\_with\_text**

Description: Plots a graphic with one or two textlines below. It is used for displaying icons. When there is a 0 instead of a text address, the line will stay empty. The graphic itself must have a size of 24x24.

Parameter: Pointer to data record.

Data record: 00 1W Graphic address (standard graphic) or address of the graphic header (extended graphic).

02 1W "0" or address of text for line 1 (terminated by 0).

04 1W "0" or address of text for line 2 (terminated by 0).

06 1B bit4: Graphic mode (0 – Standard, 1 – extended).

bit5: Text colour mode:

0 → 4 colour indexed, 1 → 16 colour.

bit6: Flag, if extended options.

bit7: Flag, if icon can be moved by the user.

→ If 4 colour text mode:

06 1B bit0–1: Paper, bit2–3: Pen.

→ If 16 colour text mode:

07 1B bit0–3: Paper, bit4–7: Pen.

→ If extended options:

08 1B bit0: Flag, if this icon can be marked.

bit1: Flag, if this icon is marked.

Size: 48x40.

### **ID: 10 (ICX) – Graphic\_extended**

Description: Plots a graphic with an extended header. For the description how a graphic object is stored in the memory see below (GRAPHICS, "Graphics with extended header"). The control must have the same size like the graphic.

Parameter: Address of the graphic header.

Data record: –

Size: Same as the graphic object.

### 5.3.3 – Buttons

#### **ID: 16 (BTN)** – button\_simple

Description: Plots a button with a centered text inside. Indexed colour 2 is used for the background, indexed colour 1 for text colour and right/lower lines, indexed colour 3 for left/upper lines.

Parameter: Text address (terminated by 0).

Data record: –

Size: Width is not limited, height must always be 12.

#### **ID: 17 (BTC)** – button\_check

Description: Plots a check box followed by a textline. The status byte contains 1, if the box is checked, otherwise it contains 0.

Parameter: Pointer to data record.

Data record: 00 1W Address of status byte (this byte can be 0 or 1)

02 1W text address (terminated by 0)

04 1B bit0–1: Indexed text paper;  
bit2–3: Indexed text pen.

Size: Width is not limited, height must always be 8.

#### **ID: 8 (BTR)** – button\_radio

Description: Plots a radio button followed by a textline. If the global status byte has the same value as the own status, this radio button is checked. The 4byte coordinate buffer has to contain –1,–1,–1,–1 at the beginning. It stores the coordinates of the current checked radio button. Radio buttons, which are connected to each other, have to point to the same global status byte and the same coordinate buffer.

Parameter: Pointer to data record.

Data record: 00 1W Address of global status byte.

02 1W Text address (terminated by 0).

04 1B bit0–1: Indexed text paper,  
bit2–3: Indexed text pen.

05 1B Value of the own status.

06 1W Pointer to a global 4-byte coordinate buffer.

Size: Width is not limited, height must always be 8.

**ID: 19 (BTP) – button\_hidden**

Description: This just defines an area on which the user can click.  
Nothing will be displayed.

Parameter: –

Data record: –

Size: Not limited.

**ID: 20 (BTT) – button\_tabs**

Description: Plots a tab line. If –1 is set as the width of one tab title the system will calculate the needed width by itself and overwrites the –1 with the correct value. As soon as the text of a tab is changed the application has to set the value to –1 again.

Parameter: Pointer to data record.

Data record: 00 1B Number of tabs

01 1B bit0–1: Indexed paper, bit2–3: Indexed pen,  
bit4–5: Indexed colour of left/upper lines,  
bit6–7: Indexed colour of right/lower lines.

02 1B Selected tab.

03 1W Text address of tab 1 title (terminated by 0).

05 1B –1 or width of tab 1 title.

06 1W Text address of tab 2 title (terminated by 0).

08 1B –1 or width of tab 2 title.

⋮

?? 1W Text address of tab n title.

?? 1B –1 or width of tab n title.

Size: Width is not limited, height must always be 11.

**5.3.4 – Miscellaneous****ID: 24 (SLD) – Slider\_simple**

Description: Plots a slider. It can be used to control a value or to move inside a window or list.

Parameter: Pointer to data record.

Data record: 00 1B bit0: Alignment (0=vertical, 1=horizontal).

bit1: 0=value control, 1=window section control.

bit7: Reserved for internal use, set to 0.

- 01 1B Not used, set to 0.
- 02 1W Current value/position
- 04 1W Maximum value/position (range is 0 – maximum)
- 06 1B Value increase, if the user clicks the down/left button
- 07 1B Value decrease, if the user clicks the up/right button

Size: Depending on the alignment, one component must have a minimum of 24 pixels; the other one must be always 8.

### **ID: 25 (SUP) – control\_collection**

Description: Plots a collection of sub controls. A control collection behaves like a sub content inside the content of a window.

Parameter: Pointer to data record.

Data record: 00 1W Pointer to sub control group data record.

02 1W Full width of the control collection area.

04 1W Full height of the control collection area.

06 1W Current X offset.

08 1W Current Y offset.

10 1B bit0: Flag, if X slider should be displayed,  
bit1: Flag, if Y slider should be displayed.

Size: If sliders are activated, the size must be more than 32x32; there are no other limitations.

## **5.3.5 – Textinput**

### **ID: 32 (TXL) – textinput\_line**

Description: Plots a textinput line. The user can use several key functions for editing the text (see below) as well as a context menu, which opens on right mouseclick. If the user modified the text, bit 7 of byte 12 of the data record will be set to 1.

Parameter: Pointer to data record.

Data record: 00 1W Address of text (has to be large enough, see below;text has to be placed anywhere inside a 16K aligned data area).

02 1W First displayed character.

04 1W Cursor position.

- 06 1W Number of selected characters (0 → no selection,  
     <0 → cursor is placed at the end of the selection,  
     >0 → cursor is placed at the beginning of the  
     selection).
- 08 1W Length of the current text.
- 10 1W Possible maximum text length (doesn't include  
     the 0 terminator at the end of the text).
- 12 1B bit0: Flag, if Password (all chars will be  
     displayed as '\*').  
     bit1: Text is read only.  
     bit2: Use alternative colours.  
     bit7: Will be set to 1, if text has been modified.
- If usage of alternative colours:
- 13 1B bit0–3: Text paper.  
     bit4–7: Text pen.
- 14 1B bit0–3: Pen of upper and left line.  
     bit4–7: Pen of lower and right line.

Size: Width is not limited, height must always be 12.

Key functions: SHFT+LEFT/RIGHT (De)select parts of the text.  
 CTRL+LEFT/RIGHT Jump word wise left/right.  
 CTRL+UP/DOWN Jump to line begin/end.  
 CTRL+A Select the complete text.  
 CTRL+C Copy selected text.  
 CTRL+X Cut selected text (copy and delete).  
 CTRL+V Paste copied text.

### ID: 33 (TXB) – textinput\_box

Description: Plots a textinput box. If the user modified the text, bit 7 of byte 12 of the data record will be set to 1.

Parameter: Pointer to data record.

Data record: 00 1W Address of text (has to be large enough and has to be placed anywhere inside a 16K aligned data area). See below:

- 02 1W Not used.
- 04 1W Cursor position (inside the complete text)
- 06 1W Number of selected characters (0 → no selection,  
     <0 → cursor is placed at the end of the selection,  
     >0 → cursor is placed at the beginning of the  
     selection)

- 08 1W Length of the current text
- 10 1W Possible maximum text length (doesn't include the 0 terminator at the end of the text)
- 12 1B bit1: Text is read only  
bit2: Use alternative colours  
bit3: Use alternative font  
bit7: Will be set to 1, if text has been modified
- 13 1B bit0–3: Text paper, bit4–7: Text pen  
(only when using alternative colours)
- 14 1B Not used.
- 15 1W Font address (only when using alternative font)
- 17 1B Reserved, set to 0.
- 18 1W Current number of lines
- 20 1W Maximum pixel width of one line for word wrapping (–1 → unlimited).
- 22 1W Maximum number of lines;
- 24 1W Used internally: X size of visible area.  
(–8=force reformatting)
- 26 1W Used internally: Y size of visible area.
- 28 1W Address of this data record
- 30 1W Used internally: Total X size.
- 32 1W Used internally: Total Y size.
- 34 1W Used internally: X offset of visible area.
- 36 1W Used internally: Y offset of visible area.
- 38 1B bit0: Word wrapping (0=at window border, 1=at maximum pixel position, see byte20)  
bit1: 1 (has to be set always)
- 39 1B Tab stop width (1–255; 0=no tab stop)
- 40 4B Message buffer for additional control commands
- 44 4B Reserved, set to 0.
- 48 [maximum number of lines]W  
Line length table; this table contains a word for each line with the length in chars; that may also include potential carriage return/line feed (CR+LF) codes at the end of a line; bit15 is set, if a line contains the CR+LF codes.

Key functions: SHFT+LEFT/RIGHT (De)select parts of the text.  
CTRL+LEFT/RIGHT Jump word wise left/right.



CTRL+UP/DOWN    Jump to line begin/end.  
 CTRL+A        Select the complete text.  
 CTRL+C        Copy selected text.  
 CTRL+X        Cut selected text (copy and delete).  
 CTRL+V        Paste copied text.

**Commands:** The textinput box control provides additional functions, which can be accessed by sending special keyboard codes to the control. This is done by using the KEYPUT function (see Device Manager documentation) while the control has focus position. If a command requires additional parameters, they have to be stored at byte 40 in the data record before sending the code. Here you will also find the results, if the command returns. The following commands are available:

Code 29: Get cursor position; this command returns the current cursor position

Output: (buffer+0)=column (starting at 0)  
           (buffer+2)=line (starting at 0)

Code 30: Text has been modified; this command forces the control to reformat and update the text.

Code 31: Set cursor position and text selection; the visible area of the textinput box will be scrolled to the new position, if necessary.

Input: (buffer+0)=new cursor position  
           (buffer+2)=new number of selected chars

### 5.3.6 – Lists

#### **ID: 40 (LST)** – List\_title

**Description:** Plots the title line of a list.

**Parameter:** Pointer to data record.

**Data record:** 00 1W Number of lines

02 1W First displayed line of the list

04 1W Pointer to data record for the list content

06 2B Not used, set to 0.

08 1B Number of columns (1~64).

09 1B bit0~5: Index of sorted column.

bit6: Sort list on start.

bit7: Sort order (0=ascending, 1=descending).

10 1W Pointer to data record for the columns.  
 12 1W Last clicked line.  
 14 1B bit0: Flag, if list slider will be displayed.  
 bit1: Flag, if multiselections are possible.  
 15 1B Not used, set to 0.  
 Column record: [Number of columns] \* [  
 00 1B bit0-1: Allignment (0=left, 1=right, 2=center)  
 bit2-3: Type (0=text, 1=graphic, 2=16-bit number,  
 3=32-bit number)  
 01 1B Not used, set to 0.  
 02 1W Width of this column in pixel.  
 04 1W Text address of the title (terminated by 0).  
 06 2B Not used, set to 0.  
 ]  
 List record: [Number of lines] \* [  
 1W bit0-12: Value of this line  
 bit13: Colour of the first row (1=alternative)  
 bit14: Set to 0, it is internally used for "selection  
 update".  
 bit15: Flag, if this line is marked.  
 [Number of columns] \*  
 1W Text/data address or value for this cell.  
 ]  
 Size: Width is not limited, height must always be 10.

#### **ID: 41 (LSI) – List\_content**

Description: Plots the list itself without the title.

Parameter: Pointer to data record

Data record: See ID 40.

Size: Width must be equal or larger than 11, height must be equal or larger than 16

#### **ID: 42 (LSP) – List\_dropdown**

Description: Plots a dropdown list. Only one line of the list will be displayed. If the user clicks on this control, the complete list will drop down and the user can choose one of the entries.

Parameter: Pointer to data record.

Data record: See ID 40.

12 1W Last clicked line (this always represents the selected line).

14 1B bit0: Flag, if list slider will be displayed (should be set to 1, if list has more than 10 entries).

bit1: Flag, if multiselections are possible (always set to 0)

Size: Width must be equal or larger than 11, height must always be 10.

### **ID: 43 (LSC) – List\_complete**

Description: Plots the list title and the list itself together. This is the combination of ID 40 and ID 41.

Parameter: Pointer to data record.

Data record: See ID 40.

Size: Width must be equal or larger than 11, height must be equal or larger than 26

### **5.3.7 – Pulldown Menus**

You can define up to 8 sub menu levels. The WINDOW DATA RECORD points to the highest menu level. These are the entries you see in the menu bar of a window. These entries usually point to their sub menus, which contain entries, too, which are clickable or which point to an additional sub menu again.

00 1W Number of entries

[Number of entries] \* [

00 1W bit0: Flag, if the menu entry is active. Deactivated entries can't be clicked by the user and will appear in a different colour.

bit1: Flag, if there is a check mark behind the entry.

bit2: Flag, if the entry opens a sub menu.

bit3: Flag, if there is no entry but a separator line.

02 1W Text address (terminated by 0). If bit3 of the previous word is set, you have to use 0 here.

04 1W Value, if the entry is clickable, or address of the sub menu data record, if bit2 of the first word is set.

06 1W Reserved, set to 0.

]

## 5.4 – FONTS AND GRAPHICS

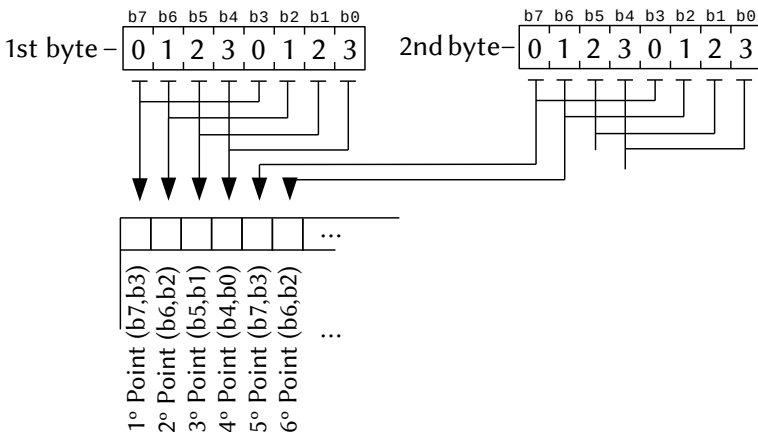
### 5.4.1 – Standard graphics

A SymbOS standard graphic has 4 colours and can have a maximum size of 255x255 pixel. Each graphic object starts with a 3 byte header:

- 00 1B bit0–6: Width of the graphic in bytes.  
bit7: Encoding type (0=CPC, 1=MSX).
- 01 1B Width of the graphic in pixel.
- 02 1B Height of the graphic in pixel.

Directly behind the header the amount of  $[\text{Width\_in\_bytes}] * [\text{Height\_in\_pixel}]$  bytes is following containing the graphic data. Every graphic is stored line by line like a sprite. The pixels have to be encoded in CPC format (Mode 1). Graphics on a MSX system will automatically be converted to the MSX format, when they are displayed the first time. bit 7 of header byte 0 contains the current encoding format. Please note, that it is not allowed to store an original graphic in MSX format, as a CPC system is not able to handle such graphics!

The following is a description of the CPC encoding format. Each byte contains 4 pixels:



Only applications, which have to modify a graphic after it has been displayed the first time, should take care about the encoding type and the MSX format.

## 5.4.2 – Graphics with extended header

As the width of a graphic is limited to 255 pixel, it wouldn't be possible to store a complete screen (like 320 X 200 in CPC Mode 1) in one piece. Such a screen needs to be splitted in two pieces (eg. 2 X 160 X 200), which makes it very difficult to write graphic modification routines.

Extended graphics do not have this limitation and also allow more than 4 colours. They can only be used for control ID 10, "graphic\_extended". A graphic can be stored in one piece with a width of up to 1020 pixel. The control "graphic\_extended" then is able to display a part of such a big linear stored graphic.

The extended header is build like this:

- 00 1B Width of the complete graphic in bytes (this has to be an even value!).
- 01 1B Width of the graphic area, which should be displayed, in pixel.
- 02 1B Height of the graphic area, which should be displayed, in pixel.
- 03 1W Address of the graphic data, including the area offset.
- 05 1W Address of the encoding information byte (see below). Please note: This single byte has ALWAYS to be placed directly in front of the complete graphic data!
- 07 1W Size of the complete graphic.
- ?? 1B Encoding information:
  - bit0–1: Colour encoding (0 → CPC, 1 → MSX).
  - bit2–3: Colour depth (0 → 4 colours, 1 → 16 colours).
 Only the following two initial values are allowed:
  - 0 → 4 colours, CPC format; an MSX system will convert the graphic to MSX format, when it is displayed the first time.
  - 5 → 16 colours, MSX format; a CPC and PCW system will render down the complete graphic to 4 colours (CPC format), when it is displayed the first time.
- ??+1 X Graphic data.

The graphic header doesn't need to be stored directly in front of the graphic, it just needs to be located in the same 16K data area like the graphic itself. You can use this type of graphic:

- If your graphic is larger than 255 pixel
- If you only want to display a part of the graphic
- If you don't want to store the header directly in front of the graphic
- If you want to use 16 colour graphics

In any other case you should use standard graphics, as they are a little bit faster. The graphic itself ("graphic\_data") is stored in one piece in memory (without header). Then we have two headers ("graphic\_header\_for\_area\_1" and "graphic\_header\_for\_area\_2") which are pointing to two different areas of the big graphic.

### 5.4.3 – Fonts

A font defines the appearance of the characters used for printing texts in SymbOS. A font starts with a simple 2 byte header:

- 00 1B Height of each character in pixel. This value can be between 1 and 15. The usual value is 8.
- 00 1B First character in the font. This value can be between 0 and 255. To save memory the usual value is 32 ("space", the first printable ASCII char), as the first 32 chars normally won't be printed. Please note, that the SymbOS system font always starts with 32 and consists of 98 chars (32–129).

After the header the char definitions follow (1568 bytes):

- 00 1B Width of the first char in pixel.
- 01 1B bit mask of the 1st pixel line of the first char.
- 02 1B bit mask of the 2nd pixel line of the first char.
- ⋮
- 15 1B bit mask of the 15th pixel line of the first char.
- 16 1B Width of the second char in pixel.
- 17 1B bit mask of the 1st pixel line of the second char.
- ⋮

## 5.5 – SYSTEM MANAGER

The system manager is responsible for starting and stopping applications and for general system jobs. It provides several dialogue services and it owns the file manager, which can only be accessed via the system manager process (for more information see the "FILE MANAGER" chapter). System manager commands are triggered via a message, which has to be sent with RST 10H (MSGSEND) to the system manager process. The system manager process always has the ID 3.

### 5.5.1 – Application Management

**ID: 016 (MSC\_SYS\_PRGRUN – Program\_Run\_Command**

Description: Loads and starts an application or opens a document with a known type by loading the associated application first. If bit 7 of P3 is not set, the system will open a message box, if an error occurs during the loading process.

Library: SySystem\_PRGRUN.

Message: 00 1B 016.

01 1W File path and name address.

03 1B Bit0–3: File path and name RAM bank (0~15).

Bit7: Flag, if system error message should be suppressed.

Response: See MSR\_SYS\_PRGRUN

**ID: 144 (MSR\_SYS\_PRGRUN) – Program\_Run\_Response**

Description: The system manager sends this message after trying to load an application or after opening an associated document. If the operation was successful, you will find the application ID and the process ID in P8 and P9. If it failed because of loading problems P8 contains the file manager error code.

Message: 00 1B 144.

01 1B Success status.

0 – OK.

1 – File does not exist.

2 – File is not an executable and its type is not associated with an application.

3 – Error while loading (see P8 for error code).

4 – Memory full.

→ If success status is 0:

08 1B Application ID.

09 1B Process ID (the applications main process).

→ If success status is 3:

08 1B File manager error code.

### **ID: 017 (MSC\_SYS\_PRGEND) – Program\_End\_Command**

Description: Stops an application and releases all its used system resources. Please note, that this command can't release memory, stop processes and timers or close windows, which are not registered for the application.

Library: SySystem\_PRGEND

Message: 00 1B 017

01 1B Application ID.

Response: The system manager does not send a response message.

### **ID: 020 (MSC\_SYS\_PRGSTA) – Program\_Run\_Dialogue\_Command**

Description: Opens the "run" dialogue. The user then can select an application or a document.

Message: 00 1B 020.

Response: The system manager does not send a response message.

### **ID: 024 (MSC\_SYS\_PRGSET) – Program\_Run\_ControlPanel\_Command**

Description: Starts the control panel application or one of its two sub modules.

Message: 00 1B 024.

01 1B Control panel sub module.

0 → Main window.

1 → Display settings.

2 → Time and date settings.

Response: The system manager does not send a response message.

### **ID: 025 (MSC\_SYS\_PRGTSK) – Program\_Run\_TaskManager\_Command**

Description: Starts the task manager application.

Message: 00 1B 025.

Response: The system manager does not send a response message.

### **ID: 030 (MSC\_SYS\_PRGSRV) – Program\_SharedService\_Command**

Description: Search, start and release shared services.



Message: 00 1B 030.  
 04 1B Command type:  
           0 → Search application or shared service.  
           1 → Search, start and use shared service.  
           2 → Release shared service.  
       → If P4 is 0 or 1:  
 01 1W Address of the 12-byte application ID string.  
       → If P4 is 0 or 1:  
 03 1B RAM bank (0~15) of the 12-byte application  
           ID string.  
       → If P4 is 2:  
 03 1B Application ID of shared service.  
 Response: See MSR\_SYS\_PRGSRV.

#### **ID: 158 (MSR\_SYS\_PRGSRV) – Program\_SharedService\_Response**

Description: Command type 0 ("search") will return 5 (not found) or 0 (OK). In the latter case you will find the application and process ID in P8 and P9. Command type 1 ("search, start and use") will return 0 (OK) if the shared services has been found or loaded successfully. In the other case it will return a loading error code of 1, 2, 3 or 4, which is identical with these of MSR\_SYS\_PRGRUN. Command type 2 ("release") will always return 0 (OK).

Message: 00 1B 158.  
 01 1B Result status:  
           0 → OK.  
           5 → Application or shared service not found  
               (can only occur on command type 0).  
           1~4 → Error while starting shared service; same  
               codes like in MSR\_SYS\_PRGRUN, please  
               read there for a detailed description.  
       → If command type was 0 or 1, and result status is 0:  
 08 1B Application ID of shared service.  
 09 1B Process ID (the applications main process).  
       → If result status is 3:  
 08 1B File manager error code.

### 5.5.2 – System Management

The system manager will not send response messages after processing the following commands.

#### **ID: 018 (MSC\_SYS\_SYSWNX) –**

– System\_Dialogue\_NextWindow\_Command

Description: Opens the dialogue for changing the current window.  
The next window is preselected. THIS COMMAND IS NOT IMPLEMENTED YET.

Message: 00 1B 018.

#### **ID: 019 (MSC\_SYS\_SYSWPR) –**

– System\_Dialogue\_PreviousWindow\_Command

Description: Opens the dialogue for changing the current window.  
The previous window is preselected. THIS COMMAND IS NOT IMPLEMENTED YET.

Message: 00 1B 019.

#### **ID: 021 (MSC\_SYS\_SYSSEC) –**

– System\_Dialogue\_SystemSecurity\_Command

Description: Opens the "SymbOS security" dialogue.

Message: 00 1B 021.

#### **ID: 022 (MSC\_SYS\_SYSQUIT) – System\_Dialogue\_ShutDown\_Command**

Description: Opens the "shut down" dialogue.

Message: 00 1B 022.

#### **ID: 023 (MSC\_SYS\_SYSOFF) – System\_ShutDown\_Command**

Description: Resets the computer.

Message: 00 1B 023.

#### **ID: 028 (MSC\_SYS\_SYSCFG) – System\_Configuration\_Command**

Description: Loads or saves the configuration or reinitializes the desktop background or the screen saver.

Message: 00 1B 028.

01 1B Action type:

0 → Reload configuration.

1 → Save current configuration.

2 → Reload/reinitialize desktop backg. picture.

3 → Reload or reinitialize screen saver.

### 5.5.3 – Dialogue Services

#### ID: 029 (MSC\_SYS\_SYSWRN) – Dialogue\_Infobox\_Command

Description: Opens an info, warning or confirm box and displays three line f text and up to three click buttons.

Library: SySystem\_SYSWRN

Message: 00 1B 029

01 1W Content data address

03 1B Content data RAM bank (0~15)

04 1B bit0–2: Number of buttons (1–3)

1 → “OK” button

2 → “Yes”, “No” buttons

3 → “Yes”, “No”, “Cancel” buttons

bit3–5: Titletext

0 → Default (bit7=[0]“Error!”/[1]“Info“)

1 → “Error!”

2 → “Info”

3 → “Warning”

4 → “Confirmation”

bit6: Flag, if window should be modal window.

bit7: Box type:

0 → Default (warning [!] symbol).

1 → Info (own symbol will be used).

Content 00 1W Address of text line 1.

data: 02 1W 4 \* (text line 1 pen) + 2.

04 1W Address of text line 2.

06 1W 4 \* (text line 2 pen) + 2.

08 1W Address of text line 3.

10 1W 4 \* (text line 3 pen) + 2.

→ If bit7 of P4 is 1:

12 1W Address of symbol (24x24px 4col SymbOS graphic format).

Response: See MSR\_SYS\_SYSWRN.

#### ID: 157 (MSR\_SYS\_SYSWRN) – Dialogue\_Infobox\_Response

Description: The system manager sends back this message to the application, when a infobox should be opened, or if the user clicked one of the buttons.

Message: 00 1B 157.  
 01 1B Message type:  
 0 → The infobox is currently used by another application. It can only be opened once at the same time, if it's not a pure info msg (one button, not a modal window). The user should close the other infobox first before it can be opened again by the app.  
 1 → The infobox has been opened successful as a modal window. This message won't be sent for non-modal window infoboxes.  
 2 → The user clicked "OK".  
 3 → The user clicked "Yes".  
 4 → The user clicked "No".  
 5 → The user clicked "Cancel" or close button.  
 → If P1 is 1:  
 02 1B Number of the infobox window + 1. The application should store this number as the modal window ID of its own window, so that the infobox will be handled as the modal window of the application window. As long as it is open the application window can't get the focus position. For more information about the window data structure and modal windows see the chapter "desktop manager".

### **ID: 031 (MSC\_SYS\_SELOPN) – Dialogue\_FileSelector\_Command**

Description: Opens the file selection dialogue. You can filter the entries of the directory by attributes and filename extension. We recommend always to set Bit3 of the attribute filter byte. The File mask/path/name string (260 bytes) must always be placed in the transfer RAM area (C000H~FFFFH).

Library: SysSystem\_SELOPN.

Message: 00 1B 031.  
 06 1B bit0–3: File mask, path and name RAM bank (0~15).  
 bit6: Flag, if "open" (0) or "save" (1) dialogue.  
 bit7: Flag, if file (0) or directory (1) selection.

- 07 1B Attribute filter:
  - bit0 = 1 → Don't show read only files.
  - bit1 = 1 → Don't show hidden files.
  - bit2 = 1 → Don't show system files.
  - bit3 = 1 → Don't show volume ID entries.
  - bit4 = 1 → Don't show directories.
  - bit5 = 1 → Don't show archive files.
- 08 1W File mask, path and name address (C000H–FFFFH).
- 00 3B File extension filter (e.g. “\*”).
- 03 1B 0.
- 04 256B Path and filename.
- 10 1W Maximum number of directory entries.
- 12 1W Maximum size of directory data buffer.

Response: See MSR\_SYS\_SELOPN.

#### **ID: 159 (MSR\_SYS\_SELOPN) – Dialogue\_FileSelector\_Response**

Description: The system manager sends back this message to the application, when a file selection dialogue should be opened. If opening was successful the application will first receive a type “-1” message and then, after the user choosed his file or aborted, a type 0 or 1 message. If opening failed the application will directly receive a type 2, 3 or 4 message.

- Message:
- 00 1B 159
  - 01 1B Message type
    - 0 → The user choosed a file or directory and closed the dialogue with “OK”. The complete file path and name can be found in the filepath buffer of the application.
    - 1 → The user aborted the file selection. The content of the applications filepath buffer is unchanged.
    - 2 → The file selection dialogue is currently used by another application. It can only be opened once at the same time. The user should close the dialogue first before it can be opened again by the application.

- 3 → Memory full. There was not enough memory available for the directory buffer and/or the list data structure.
  - 4 → No window available. The desktop manager couldn't open a new window for the dialogue, as the maximum number of windows (32) has already been reached.
  - 1 → The dialogue has been opened successful and the user is doing his file selection right now.
- If P1 is -1:
- 02 1B Number of the dialogue window + 1. The application should store this number as the modal window ID of its own window, so that the file selection dialogue will be handled as the modal window of the application window. As long as it is open the application window can't get the focus position. For more information about the window data structure and modal windows see the chapter "desktop manager".

### 5.5.4 – System Manager Functions

The system manager functions have to be called with RST 28H (BNKFCL).

#### **SYSINF** (8103H) – System\_Information

Description: This function is mainly used by the task manager and the control panel application. Request types 0~2 are not documented yet.

How to call: `ld hl,8103H : rst 28H`

Input: E – Request type (see below):

- 0 → Get general information.
- 1 → Get application information.
- 2 → Get task information.
- 3 → Load mass storage device configuration.
- 4 → Save mass storage device configuration.
- 5 → Load a part of the configuration.

- 6 → Save a part of the configuration.  
 7 → Get config memory address.  
 8 → Get font and version string memory address.  
 D, IX, IY – Sub specification (see below).  
 Output: DE, IX, IY – Result data (see below).  
 Registers: AF, BC, HL.
- Request type: 3 (Load mass storage device configuration).  
 Description: Loads the complete device configuration into the applications memory (8\*16 bytes). For a description of the data structure please see "Configuration Data/Core Area Part/Mass Storage Devices".  
 Input: E = 3.  
 IX – Destination address (must be placed inside the transfer RAM area).  
 Output: –
- Request type: 4 (Save mass storage device configuration).  
 Description: Saves the complete device configuration from the applications memory (8\*16 bytes).  
 Input: E = 4.  
 IX – Source address (must be placed inside the transfer RAM area).  
 Output: –
- Request type: 5 (Load a part of the configuration core area).  
 Description: Loads a part of the core area into the applications memory. For a description of the data structure please see "Configuration Data/Core Area Part".  
 Input: E = 5.  
 D – Number of bytes.  
 IX – Destination address (transfer RAM area).  
 IY – Source offset (starting from byte 163 [=system path] in the core part).  
 Output: –
- Request type: 6 (Save a part of the configuration core area).  
 Description: Saves a part of the core area from the applications memory.

- Input: E = 6.  
D – Number of bytes  
IX – Source address (transfer RAM area)  
IY – Destination offset (starting from byte 163  
[=system path] in the core part).
- Output: –
- Request type: 7 (Get config memory address).  
Description: Sends back the address of the core area part  
(including the 6byte–header, so you have to add 6 to  
have the starting ddress), which is always placed in  
RAM bank 0, and the data area part together with  
the data area parts RAM bank number.
- Input: E = 7.  
Output: DE – Core area address (including 6byte–header;  
RAM bank 0).  
IX – Data area address.  
IYI – Data area RAM bank (0~8).
- Request type: 8 (get font and version string memory address)  
Description: Sends back the address and total size of the font,  
which is always placed in RAM bank 0, and the  
address of the version tring, which is placed in the  
same RAM bank like the data area part (see request  
type 7).
- Input: E = 8.  
Output: DE – Font address (RAM bank 0)  
IX – Font length (=2 byte header + 98\*16 byte char  
bitmaps)  
IY – Address of the version information (this is  
placed in the data area RAM bank).  
The version information has a length of  
32 bytes:  
00 1B Version Major.  
01 1B Version Minor.  
02 30B Version String (terminated by 0).



## 5.6 – FILE MANAGER

The file manager is owned by the system manager process, which is the only one, who is allowed to call file manager functions. If an application wants to use the file manager, it needs to send a special message to the system manager process, which includes all registers. The system manager then will call the specified file manager function and sends a message with the result back to the caller application. The system manager process always has the ID 3. Please note, that in SymbOS all texts must be terminated with a 0 byte. This is true for the pathes and filenames used in the file manager, too.

### 5.6.1 – System Manager Messages

#### **ID: 026 (MSC\_SYS\_SYSFIL) – System\_Filemanager\_Command**

Description: An application has to send this message to the system manager (process ID 3) to call a file manager function.

Message:   00 1B 026.  
               01 1B File manager function ID.  
               02 1W Input for AF.  
               04 1W Input for BC.  
               06 1W Input for DE.  
               08 1W Input for HL.  
               10 1W Input for IX.  
               12 1W Input for IY

#### **ID: 154 (MSR\_SYS\_SYSFIL) – System\_Filemanager\_Response**

Description: The system manager sends this message back to the application, after the file manager function has been called.

Message:   00 1B 154  
               01 1B File manager function ID  
               02 1W Output for AF  
               04 1W Output for BC  
               06 1W Output for DE  
               08 1W Output for HL  
               10 1W Output for IX  
               12 1W Output for IY

### 5.6.2 – Error Codes

Nearly all file-manager functions return the success status in the carry flag. If the carry flag is not set, the operation was successful. If it is set, an error occurred. In this case, the A-register contains the error code number. The following is a list of all possible error codes.

- 000 – Device does not exist.
- 001 – OK.
- 002 – Device not initialised.
- 003 – Media is damaged.
- 004 – Partition does not exist.
- 005 – Unsupported media or partition.
- 006 – Error while sector read/write.
- 007 – Error while positioning.
- 008 – Abort while volume access.
- 009 – Unknown volume error.
- 010 – No free filehandler.
- 011 – Device does not exist.
- 012 – Path does not exist.
- 013 – File does not exist.
- 014 – Access is forbidden.
- 015 – Invalid path or filename.
- 016 – Filehandler does not exist.
- 017 – Device slot already occupied.
- 018 – Error in file organisation.
- 019 – Invalid destination name.
- 020 – File/path already exist.
- 021 – Wrong sub command code.
- 022 – Wrong attribute.
- 023 – Directory full.
- 024 – Media full.
- 025 – Media is write protected.
- 026 – Device is not ready.
- 027 – Directory is not empty.
- 028 – Invalid destination device.
- 029 – Not supported by file system.
- 030 – Unsupported device.

- 031 – File is read only.
- 032 – Device channel not available.
- 033 – Destination is not a directory.
- 034 – Destination is not a file.
- 255 – Undefined Error.

### 5.6.3 – Mass Storage Device Functions

#### ID: 000 (STOINI) – Storage\_Init

Description: Removes all mass storage devices.

Input: –

Output: –

Registers: BC, DE, HL.

#### ID: 001 (STONEW) – Storage\_New

Description: Adds a new mass storage device.

Input: A – Device (0~7).

C – Sub drive.

DE – Driver address.

L – Removeable media flag (1 → Removeable).

B – Drive letter ("A"-"Z").

IX – Device name (11 characters).

Output: CY – Error state (0 – Ok, 1 – Error; A → error code).

Registers: AF, BC, DE, HL, IX, IY.

#### ID: 002 (STORLD) – Storage\_Reload

Description: Reloads a mass storage device, if its “removeable media” status is activated. The format and the filesystem type will be loaded again.

Input: A – Device (0~7).

Output: CY – Error state (0 – Ok, 1 – Error; A → error code).

Registers: AF, BC, DE, HL, IX, IY.

#### ID: 003 (STODEL) – Storage\_Delete

Description: Removes an existing mass storage device.

Input: A – Device (0~7).

Output: CY – Error state (0 – Ok, 1 – Error; A → error code).

Registers: AF, BC, DE, HL.

**ID: 004 (STOINP)** – Storage\_ReadSector

Description: Reads a sector from a mass storage device (no memory banking).

Input:       A – Device (0~7).  
               IY, IX – First sector number.  
               B – Number of sectors.  
               DE – Destination address.

Output:       CY – Error state (0 – Ok, 1 – Error; A → error code).

Registers:   AF, BC, DE, HL, IX, IY.

**ID: 005 (STOOUT)** – Storage\_WriteSector

Description: Writes a sector to a mass storage device (no memory banking).

Input:       A – Device (0~7).  
               IY, IX – First sector number.  
               B – Number of sectors.  
               DE – Source address.

Output:       CY – Error state (0 – Ok, 1 – Error; A → error code).

Registers:   AF, BC, DE, HL, IX, IY.

**ID: 006 (STOACT)** – Storage\_Activate

Description: Loads the format and the file system type of a mass storage device.

Input:       A – Device (0~7).

Output:       CY – Error state (0 – Ok, 1 – Error; A → error code).

Registers:   AF, BC, DE, HL, IX, IY.

**ID: 007 (STOINF)** – Storage\_Information

Description: Returns information about a mass storage device.

Input:       A – Device (0~7).

Output:       A – Type:  
               00 → Device does not exist.  
               01 → Device is ready.  
               02 → Device is not initialized.  
               03 → Device is corrupt.  
               B – Medium:  
               01 → Floppy disc single side (Amsdos, PCW).  
               02 → Floppy disc double side (FAT 12).  
               08 → RAM disc (\*not supported yet\*).  
               16 → IDE HD or CF card (FAT 12, FAT 16, FAT 32).

C – File system:

01 → Amsdos Data.

02 → Amsdos System.

03 → PCW 180K.

16 → FAT 12.

17 → FAT 16.

18 → FAT 32.

D – Sectors per cluster:

IY, IX – Total number of clusters.

Registers: E, HL.

#### **ID: 08 (STOTRN)** – Storage\_DataTransfer

Description: Reads or writes a number of sectors (512 bytes) from/to the mass storage device. Sector 0 is the first sector of the partition of the device.

Input: A – Device (0~7).

IY, IX – First sector number.

B – Number of sectors.

C – Direction (0=read, 1=write).

HL – Source/destination address.

E – Source/destination RAM bank (0~15).

Output: CY – Error state (0 – Ok, 1 – Error; A → error code).

Registers: AF, BC, DE, HL, IX, IY.

### **5.6.4 – File Management Functions**

#### **ID: 016 (FILINI)** – File\_Init

Description: Initialises the whole file manager. You should never call this function by yourself, as it resets everything!

Input: –

Output: –

Registers: AF, BC, DE, HL.

#### **ID: 017 (FILNEW)** – File\_New

Description: Creates a new file and opens it for read/write access. If the file was already existing, it will be emptied first. The operation will be aborted, if the existing file is read only or an sub directory. For additional information see 018 (FILOPN).

Library: SyFile\_FILNEW  
 Input: IXh – File path and name RAM bank (0~15)  
 HL – File path and name address.  
 A – Attributes:  
     bit0 = 1 → Read only.  
     bit1 = 1 → Hidden.  
     bit2 = 1 → System.  
     bit5 = 1 → Archive.  
 Output: A – Filehandler ID.  
         CY – Error state (0 – Ok, 1 – Error; A → error code).  
 Registers: F, BC, DE, HL, IX, IY.

### **ID: 018 (FILOPN) – File\_Open**

Description: Opens an existing file for read/write access. You can open up to 7 different files at the same time. The media will be reloaded first, if the device is set to “removeable media” and there is no other open file on the same device.

Library: SyFile\_FILOPN  
 Input: IXh – File path and name RAM bank (0~15).  
 HL – File path and name address.  
 Output: A – Filehandler ID.  
         CY – Error state (0 – Ok, 1 – Error; A → error code).  
 Registers: F, BC, DE, HL, IX, IY.

### **ID: 019 (FILCLO) – File\_Close**

Description: Closes an opened file. If there is unwritten data in the sector cache, it will be written to disc at once. This command closes a file in any case, even if an error ocured. If an error ocured during file reading/writing you must close the file, too, to make the filehandler free again!

Library: SyFile\_FILCLO.  
 Input: A – Filehandler ID.  
 Output: CY – Error state (0 – Ok, 1 – Error; A → error code).  
 Registers: AF, BC, DE, HL, IX, IY.

### **ID: 020 (FILINP) – File\_Input**

Description: Reads a specified amount of bytes out of an opened file. After read byte. If you try to read more bytes than available, the zero flag will be reset. In any case BC contains the amount of read bytes (which could also be 0).

Library: SyFile\_FILINP.  
 Input: A – Filehandler ID.  
       HL – Destination address.  
       E – Destination RAM bank (0~15).  
       BC – Number of bytes.  
 Output: BC – Number of read bytes.  
       Z = 1 → All requested bytes have been read.  
       0 → The end of the file has been reached, and less  
           bytes than requested have been read (check BC).  
       CY – Error state (0 – Ok, 1 – Error; A → error code).  
 Registers: AF, DE, HL, IX, IY.

### **ID: 021 (FILOUT) – File\_Output**

Description: Writes a specified amount of bytes into an opened file.  
 After this operation the file pointer will be moved behind  
 the last written byte.  
 Library: SyFile\_FILOUT.  
 Input: A – Filehandler ID.  
       HL – Source address.  
       E – Source RAM bank (0~15).  
       BC – Number of bytes.  
 Output: BC – Number of written bytes.  
       A = 0 → All bytes have been written.  
       1 → The device is full, and less bytes have been  
           written (check BC).  
       CY – Error state (0 – Ok, 1 – Error; A → error code).  
 Registers: AF, DE, HL, IX, IY.

### **ID: 022 (FILPOI) – File\_Pointer**

Description: Moves the file pointer to another position. The difference  
 is specified with IY and IX, IY is the high word, IX the  
 low word (difference –  $65536 * IY + IX$ ).  
 Ex.: IY=0, IX=1, C=1 → Increases the position by 1.  
       IY=65535, IX=-10, C=2 → Sets the pointer before the  
           last 10 bytes of the file.  
 Library: SyFile\_FILPOI  
 Input: A – Filehandler ID.  
       IY, IX – Difference.

C – Reference point:

0 → File begin (difference is unsigned).

1 → Current pointer position (difference is signed).

2 → File end (difference is signed).

Output: IY, IX – New absolute pointer position.

CY – Error state (0 – Ok, 1 – Error; A → error code).

Registers: AF, BC, DE, HL.

### **ID: 023 (FILF2T) – File\_Decode\_Timestamp**

Description: Decodes the file timestamp, which is used for the file system. You can use this function after reading the timestamp of a file with 035 (DIRPRR) or 038 (DIRINP).

Library: SyFile\_FILF2T.

Input: BC – Time code:

bit 0–4 – Second/2.

bit 5–10 – minute.

bit 11–15 – hour.

DE – Date code:

bit 0–4 – Day (starting from 1).

bit 5–8 – month (starting from 1).

bit 9–15 – year–1980.

Output: A – Second.

B – Minute.

C – Hour.

D – Day (starting from 1).

E – Month (starting from 1).

HL – Year.

Registers: F.

### **ID: 024 (FILT2F) – File\_Encode\_Timestamp**

Description: Encodes the file timestamp, which is used for the file system. You can use this function before changing the timestamp of a file with 034 (DIRPRS).

Library: SyFile\_FILT2F

Input: A – Second.

B – Minute.

C – Hour.

D – Day (starting from 1).

E – Month (starting from 1).

HL – Year.



Output: BC – Time code (see FILF2T)  
 DE – Date code (see FILF2T)  
 Registers: AF, HL, IX, IY.

#### **ID: 025 (FILLIN) – File\_LineInput**

Description: Reads one text line out of an opened file. A text line is terminated by a single 13, a single 10, a combination of 13+10, a combination of 10+13 or by a single 26 (“end of file” code).

Library: SyFile\_FILLIN

Input: A – Filehandler ID.

HL – Destination buffer address (size must be 255 bytes).

E – Destination buffer RAM bank (0~15).

Output: C – Number of read bytes (0~254; without terminator).

B – Flag, if line/file end reached (0=no, 1=yes).

Z = 0 → 1 or more bytes have been loaded.

1 → EOF reached, nothing has been loaded.

CY – Error state (0 – Ok, 1 – Error; A → error code).

Registers: AF, DE, HL, IX, IY.

### **5.6.5 – Directory Management Functions**

#### **ID: 032 (DIRDEV) – Directory\_Device**

Description: Selects the current drive.

Library: SyFile\_DIRDEV.

Input: A – Driveletter ("A"-"Z").

Output: CY – Error state (0 – Ok, 1 – Error; A → error code).

Registers: AF, BC, DE, HL, IX, IY.

#### **ID: 033 (DIRPTH) – Directory\_Path**

Description: Selects the current path for the current or a different drive.

Library: SyFile\_DIRPTH.

Input: IXh – File path RAM bank (0~15).

HL – File path address.

Output: CY – Error state (0 – Ok, 1 – Error; A → error code).

Registers: AF, BC, DE, HL, IX, IY.

**ID: 034 (DIRPRS) – Directory\_Property\_Set**

Description: Changes a property of a file or a directory. You can set the attribute, the "created" time and the "modified" time. For more information about the time and date code see 023 (FILF2T).

Library: SyFile\_DIRPRS

Input: IXh – File path and name RAM bank (0~15).

HL – File path and name address.

A – Property type.

0 – Attribute.

→ C – Attribute:

bit0 = 1 → Read only.

bit1 = 1 → Hidden.

bit2 = 1 → System.

bit5 = 1 → Archive.

1 – Timestamp modified.

→ BC – Time code, DE – Date code.

2 – Timestamp created.

→ BC – Time code, DE – Date code.

BC,DE – See above.

Output: CY – Error state (0 – Ok, 1 – Error; A → error code).

Registers: AF, BC, DE, HL, IX, IY.

**ID: 035 (DIRPRR) – Directory\_Property\_Get**

Description: Reads a property of a file or a directory. For more information about the time and date code see 023 (FILF2T).

Library: SyFile\_DIRPRR.

Input: IXh – File path and name RAM bank (0~15).

HL – File path and name address.

A – Property type:

0 – Attribute.

1 – Timestamp modified.

2 – Timestamp created.

Output: C – Attributes (if requested):

bit0 = 1 → Read only.

bit1 = 1 → Hidden.

bit2 = 1 → System.

bit3 = 1 → Volume ID.

bit4 = 1 → Directory.

bit5 = 1 → Archive.

BC, DE – Time and date code (if requested).  
 CY – Error state (0 – Ok, 1 – Error; A → error code).

Registers: AF, HL, IX, IY.

#### **ID: 036 (DIRREN)** – Directory\_Rename

Description: Renames a file or a directory. The new filename must not include a path.

Library: SyFile\_DIRREN.

Input: IXh – RAM bank (0~15) of old and new filename.

HL – Address of file path and old filename.

DE – Address of new filename.

Output: CY – Error state (0 – Ok, 1 – Error; A → error code).

Registers: AF, BC, DE, HL, IX, IY.

#### **ID: 037 (DIRNEW)** – Directory\_New

Description: Creates a new directory.

Library: SyFile\_DIRNEW

Input: IXh – Directory path and name RAM bank (0~15)

HL – Directory path and name address

Output: CY – Error state (0 – Ok, 1 – Error; A → error code).

Registers: AF, BC, DE, HL, IX, IY.

#### **ID: 038 (DIRINP)** – Directory\_Input

Description: Reads the content of a directory. You can specify a name filter by adding a file mask to the path (\* and ? are allowed) and an attribute filter. We recommend always to set Bit3 (volume ID) of the attribute filter byte. Filenames don't contain spaces. For a more powerful function see 013 (DEVDIR).

Library: SyFile\_DIRINP.

Input: IXh – Directory path RAM bank (0~15).

HL – Directory path address (may include a search mask).

IXI – Attribute filter:

bit0 = 1 → Don't show read only files.

bit1 = 1 → Don't show hidden files.

bit2 = 1 → Don't show system files.

bit3 = 1 → Don't show volume ID entries.

bit4 = 1 → Don't show directories.

bit5 = 1 → Don't show archive files.

A – Destination buffer RAM bank (0~15)  
 DE – Destination buffer address  
 BC – Destination buffer length  
 IY – Number of entries, which should be skipped  
 Output: HL – Number of read entries  
 BC – Remaining unused space in the destination buffer  
 CY – Error state (0 – Ok, 1 – Error; A → error code).  
 Registers: AF, DE, IX, IY.  
 Data structure: 00 4B File length (32bit double word).  
 04 1W Date code, see 023 (FILF2T).  
 06 1W Time code, see 023 (FILF2T).  
 08 1B Attributes, see 035 (DIRPRR).  
 09 ?B File or sub directory name.  
 ?? 1B 0 terminator.

#### **ID: 039 (DIRDEL) – Directory\_DeleteFile**

Description: Deletes one or more files. You can delete multiple files by using a file mask (\* and ? are allowed). Files, which are read only, can't be deleted. This function also can't be used for deleting directories. Use 040 (DIRRMD), if you want to delete directories.

Library: SyFile\_DIRDEL.  
 Input: IXh – File path and name/mask RAM bank (0~15).  
 HL – File path and name/mask address.  
 Output: CY – Error state (0 – Ok, 1 – Error; A → error code).  
 Registers: AF, BC, DE, HL, IX, IY.

#### **ID: 040 (DIRRMD) – Directory\_DeleteDirectory**

Description: Deletes a sub directory. The sub directory has to be empty and not read only, otherwise the operation will be aborted.

Library: SyFile\_DIRRMD.  
 Input: IXh – Directory path and name RAM bank (0~15).  
 HL – Directory path and name address.  
 Output: CY – Error state (0 – Ok, 1 – Error; A → error code).  
 Registers: AF, BC, DE, HL, IX, IY.

**ID: 041 (DIRMOV)** – Directory\_Move

Description: Moves a file or sub directory into another directory of the same drive. You can either move files or sub directories with this function, in both cases the source path+name must not end with a "/".

Library: SyFile\_DIRMOV.

Input: IXh – File/directory old and new path RAM bank (0~15).

HL – File/directory source path and name address.

DE – File/directory destination path address.

Output: CY – Error state (0 – Ok, 1 – Error; A → error code).

Registers: AF, BC, DE, HL, IX, IY.

**ID: 042 (DIRINF)** – Directory\_DriveInformation

Description: Returns information about one drive.

Library: SyFile\_DIRINF

Input: A – Driveletter ("A"–"Z").

C – Information type.

0 – General drive information.

1 – free and total amount of memory.

Output: → Information type 0:

A – Type:

00 – Device does not exist.

01 – Device is ready.

02 – Device is not initialized.

03 – Device is corrupt.

B – Medium:

01 – Floppy disc single side (Amsdos, PCW).

02 – Floppy disc double side (FAT 12).

08 – RAM disc.

16 – IDE hard disc or CF card (FAT 16, FAT 32).

C – File system:

01 – Amsdos Data.

02 – Amsdos System.

03 – PCW 180K.

16 – FAT 12.

17 – FAT 16.

18 – FAT 32.

D – Sectors per cluster

IY, IX – Total number of clusters.

→ Information type 1:

HL, DE – Number of free 512Byte sectors.

IY, IX – Total number of clusters.

C – Sectors per cluster

→ Information type 0 and 1:

CY – Error state (0 – Ok, 1 – Error; A → error code).

Registers: F.

### **ID: 013 (DEVDIR) – Directory\_Input\_Extended**

**Description:** It reads the content of a directory and converts it into ready to use list control data. First you have to reserve two memory areas in the same RAM bank. One area needs to be reserved inside the data RAM area. It will contain the texts (filenames, dates etc.) and numbers (file sizes) for the list control. You can choose any size, but we recommend at least 4000 Bytes. BC must contain its size, when you call the function. DE contains the address, and the low nibble of A the RAM bank number. The second area needs to be reserved inside the transfer RAM area of the same bank. It contains the data structure of the list control. Its size is calculated like this:

$$\text{Size} = \text{Maximum\_number\_of\_entries} * (4 + \text{Additional\_columns} * 2)$$

So when you have two additional columns (like size and attributes) and want to load up to 100 entries, you need to reserve 800 bytes. As there are no more Z80-registers available, the address of this memory area and the maximum number of entries must be written to the beginning of the other memory area. For additional information about reading directories see 038 (DIRINP).

**Library:** SyFile\_DEVDIR.

**Input:**

- A – bit0–3 → Destination buffer RAM bank (0~15).
- bit4–7 → Directory path RAM bank (0~15).
- HL – Directory path address (may include a search mask).
- DE – Destination buffer address. This must first contain 2 words with additional information at the beginning:
  - 00 1W Address of list control table
  - 02 1W Maximum number of entries
 The function will overwrite this information and fill the buffer with the directory data.

BC – Maximum size of destination buffer.

IXI – Attribute filter:

bit0 = 1 → Don't show read only files.

bit1 = 1 → Don't show hidden files.

bit2 = 1 → Don't show system files.

bit3 = 1 → Don't show volume ID entries.

bit4 = 1 → Don't show directories.

bit5 = 1 → Don't show archive files.

IY – Number of entries, which should be skipped.

IXh – Additional columns:

bit0 = 1 → File size.

bit1 = 1 → Date and time (last modified).

bit2 = 1 → Attributes.

Output: HL – Number of read entries.

CY – Error state (0 – Ok, 1 – Error; A → error code).

Registers: AF, BC, DE, IX, IY.

## 5.6.6 – Device Manager Functions

The device manager functions have to be called with RST 20H (BNKSCL).

### **TIMGET** (810CH) – Device\_TimeGet

Description: Returns the current time.

How to call: rst 20H : dw 810CH

Input: –

Output: A – Second (0 ~ 59).

B – Minute (0 ~ 59).

C – Hour (0 ~ 23).

D – Day (1 ~ 31).

E – Month (1 ~ 12).

HL – Year (1900 ~ 2100).

IXI – Timezone (–12 ~ +13).

Registers: F, IY.

### **TIMSET** (810FH) – Device\_TimeSet

Description: Sets the current time.

How to call: rst 20H : dw 810FH

Input:       A – Second (0 ~ 59).  
               B – Minute (0 ~ 59).  
               C – Hour (0 ~ 23).  
               D – Day (1 ~ 31).  
               E – Month (1 ~ 12).  
               HL – Year (1900 ~ 2100).  
               IXI – Timezone (–12 ~ +13).

Output:       –

Registers:   AF, BC, DE, HL, IY.

### **SCRSET** (8136H) – Device\_ScreenModeCPCSet

Description: Sets the current CPC screen mode. This function is CPC specific only.

How to call: ld hl,8136H : rst 28H

Input:        E – CPC screen mode (0, 1, 2).

Output:       –

Registers:   –

### **SCRGET** (8139H) – Device\_ScreenMode

Description: Returns the current screen mode, colour depth and resolution.

How to call: ld hl,8139H : rst 28H

Input:        –

Output:       E – Screen mode:  
                   CPC/EP: 1, 2       MSX: 5, 6, 7  
                   PCW:    0        G9K: 8, 9, 10, 11  
               D – Number of colours (2–16).  
               IX – X resolution.  
               IY – Y resolution.

Registers:   –

### **MOSGET** (813CH) – Device\_MousePosition

Description: Returns the current position of the mouse pointer.

How to call: rst 20H : dw 813CH

Input:        –

Output:       DE – X position.

              HL – Y position.

Registers:   –



**MOSKEY (813FH) – Device\_MouseKeyStatus**

Description: Returns the current status of the mouse keys.

How to call: rst 20H : dw 813FH

Input: –

Output: A – Key Status  
           bit 0 = 1 → Left mouse button is pressed.  
           bit 1 = 1 → Right mouse button is pressed.  
           bit 2 = 1 → Middle mouse button is pressed.

Registers: F.

**KEYTST (8145H) – Device\_KeyTest**

Description: Returns the current status of a key. For the scan codes see KEYBOARD SCAN CODES.

How to call: ld hl,8145H : rst 28H

Input: E – Keyboard scan code.

Output: E – Key status.  
           0 → Key is currently not pressed;  
           1 → Key is currently pressed.

Registers: AF, BC, D ,HL ,IX ,IY.

**KEYSTA (8148H) – Device\_KeyStatus**

Description: Returns the status of the shift/control/alt/capslock keys.

How to call: ld hl,8148H : rst 28H

Input: –

Output: E – bit0 = 1 → Shift pressed.  
           bit1 = 1 → Control pressed.  
           bit2 = 1 → Alt pressed.  
           D – Caps lock status (1 → Locked).

Registers: AF, BC, HL, IX, IY.

**KEYPUT (814BH) – Device\_KeyPut**

Description: Puts a char back into the keyboard buffer.

How to call: rst 20H : dw 814BH

Input: A – Char (ASCII code).

Output: CY – Status (1 → Keyboard buffer full).

Registers: AF, BC, HL.

**IOMINP (8157H) – Device\_IO\_MultiIn [CPC only]**

Description: Reads multiple bytes from a hardware port in a very fast way and writes them to a destination address in memory.

This function is only available in SymbOS CPC due to its limited banking abilities.

How to call: `rst 20H : dw 8157H`

Input: DE – Destination address.

IX – bit12–15 → Destination bank (0~15).  
           bit0–11 → Length.

IX – Port address.

Registers: AF, BC, DE, HL.

### **IOMOUT (815AH) – Device\_IO\_MultiOut [CPC only]**

Description: Writes multiple bytes to a hardware port in a very fast way from a source address in memory. See also IOMINP. This function is only available in SymbOS CPC due to its limited banking abilities.

How to call: `rst 20H : dw 8157H`

Input: DE – Source address.

IX – bit12–15 → Source bank (0~15).  
           bit0–11 → Length.

IX – Port address.

Registers: AF, BC, DE, HL.

## **5.7 – SYMSHELL TEXT TERMINAL**

SymShell commands are triggered via a message, which has to be sent with RST 10H (MSGSEND) to the SymShell process. SymShell will pass its process ID and the text screen resolution to the application via the command line.

### **5.8.1 – SymShell Commands and responses**

#### **ID: 064 (MSC\_SHL\_CHRINP) – SymShell\_CharInput\_Command**

Description: Requests a char from an input source. The input source can be the standard channel or the console keyboard. If the keyboard is used, SymShell waits for the user and won't send a response as long as no key is pressed.

Library: SyShell\_CHRINP.

Message: 00 1B 064.

01 1B Channel (0 → Standard, 1 → Keyboard).

Response: See MSR\_SHL\_CHRINP.

**ID: 192 (MSR\_SHL\_CHRINP) – SymShell\_CharInput\_Response**

Description: If a char could be received from the keyboard, a file or another source, it will be sent to the application via this response message. If the user pressed Control+C or if the end of the file (EOF) has been reached, the EOF flag will be set.

Message:

|    |    |   |
|----|----|---|
| 00 | 1B | 192.  |
| 01 | 1B | EOF flag (If $\neq 0 \rightarrow$ EOF reached, no char available!). |
| 02 | 1B | Char.   |
| 03 | 1B | Error state.  |

254 – Unknown process (SymShell doesn't know the process, which sent the command, so it won't provide any service).  
 253 – Destination device full.  
 252 – Internal ring buffer full.  
 251 – Too many processes (SymShell can't handle the amount of processes running at the same time in its text terminal environment).  
 Any other: See “Error Codes” in chapter “File Manager”.

**ID: 065 (MSC\_SHL\_STRINP) – SymShell\_StringInput\_Command**

Description: Requests a string from an input source. The input source can be the standard channel or the console keyboard. The maximum length of a string is 255 chars, so the buffer must have a size of 256 bytes (255 + terminator). A string is always terminated by 0.

Library: SyShell\_STRINP.

Message:

|    |    |   |
|----|----|---|
| 00 | 1B | 065.  |
| 01 | 1B | Channel (0 $\rightarrow$ Standard, 1 $\rightarrow$ Keyboard). |
| 02 | 1B | Destination buffer RAM bank (0~15).                           |
| 03 | 1W | Destination buffer address.                                   |

Response: See MSR\_SHL\_STRINP.

**ID: 193 (MSR\_SHL\_STRINP) – SymShell\_StringInput\_Response**

Description: If a text line could be received from the keyboard, a file or another source (terminated by 13/10), it will be sent the application via this response message. If the user

pressed Control+C or if the end of the file (EOF) has been reached, the EOF flag will be set.

Message: 00 1B 193.  
 01 1B EOF flag (If  $\neq 0 \rightarrow$  EOF reached, no string available!).  
 03 1B Error state (see above “SymShell\_CharInput\_Response”).

#### **ID: 066 (MSC\_SHL\_CHROUT) – SymShell\_CharOutput\_Command**

Description: Sends a char to the output destination. The output destination can be the standard channel or the console text screen.

Library: SyShell\_CHROUT.

Message: 00 1B 066.  
 01 1B Channel (0  $\rightarrow$  Standard, 1  $\rightarrow$  Screen).  
 02 1B Char.

Response: See MSR\_SHL\_CHROUT.

#### **ID: 194 (MSR\_SHL\_CHROUT) – SymShell\_CharOutput\_Response**

Description: Informs the application, if the char has been sent correctly. An application shouldn't send more than one char at the same time, before such a response has been received.

Message: 00 1B 194.  
 03 1B Error state (see above “SymShell\_CharInput\_Response”).

#### **ID: 067 (MSC\_SHL\_STROUT) – SymShell\_StringOutput\_Command**

Description: Sends a string to the output destination. The output destination can be the standard channel or the console text screen. A string has always to be terminated by 0. The length, which has to be specified, must not include the 0-terminator.

Library: SyShell\_STROUT.

Message: 00 1B 067.  
 01 1B Channel (0  $\rightarrow$  Standard, 1  $\rightarrow$  Screen).  
 02 1B String RAM bank (0~15).  
 03 1W String address.  
 05 1B String length (without 0-terminator).

Response: See MSR\_SHL\_STROUT.

**ID: 195 (MSR\_SHL\_STROUT) – SymShell\_StringOutput\_Response**

Description: Informs the application, if the string has been sent correctly. An application shouldn't send more than one string at the same time, before such a response has been received.

Message: 00 1B 195  
 03 1B Error state (see above "SymShell\_CharInput\_Response")

**ID: 068 (MSC\_SHL\_EXIT) – SymShell\_Exit\_Command**

Description: The application informs SymShell about an exit event. If an application quits itself, SymShell has to be informed about that, so that it can remove the application from its internal management table. In this case the exit type has to be 0 ("quit").

Library: SyShell\_EXIT

Message: 00 1B 068.  
 01 1B Exit type:  
     0 → Application quits itself  
     1 → Application releases focus and goes into blur mode

Response: SymShell does not send a response message.

**ID: 069 (MSC\_SHL\_PTHADD) – SymShell\_PathAdd\_Command**

Description: ...

Library: SyShell\_PTHADD.

Message: 00 1B 069.  
 01 1W Address of base path (0 → default).  
 03 1W Address of additional path component.  
 05 1W Address of new full path.  
 07 1B Pathes RAM bank (0~15).

Response: See MSR\_SHL\_PTHADD.

**ID: 197 (MSR\_SHL\_PTHADD) – SymShell\_PathAdd\_Response**

Description: ...

Message: 00 1B 197.  
 01 1W Position behind last char in new path.  
 03 1W Position behind last / in new path.  
 05 1B bit0=1 → New path ends with /.  
     bit1=1 → New path contains wildcards.

### 5.7.2 – Symshell Text Terminal Control

- 00 Stop textoutput and ignore remaining part of the line.
- 01 –
- 02 Switch cursor off. This will make the cursor invisible.
- 03 Switch cursor on.
- 04 Save current cursor position.
- 05 Restore last saved cursor position.
- 06 Activate textoutput (see also 21).
- 07 –
- 08 Move cursor one char to the left.
- 09 Move cursor one char to the right.
- 10 Move cursor one char downwards.
- 11 Move cursor one char upwards.
- 12 Clear screen and place cursor at position 1/1.
- 13 Move cursor to the beginning of the current line.
- 14 Move cursor by multiple chars (P1 – Direction and steps)
  - 1~80 → cursor will move 1~80 chars to the right.
  - 81~160 → cursor will move 1~ 80 chars to the left (parameter=80).
  - 161~185 → cursor will move 1~25 chars downwards (parameter=160).
  - 186~210 → cursor will move 1~25 chars upwards (parameter=185).
 The cursor will not cross any borders.
- 15 –
- 16 Clear char at cursor position (using space [32]).
- 17 Clear line from cursor left.
- 18 Clear line from cursor right.
- 19 Clear screen from cursor up.
- 20 Clear screen from cursor down.
- 21 Deactivate textoutput. No more chars will be printed until a code 06 appears.
- 22 Set a tab at the current column.
- 23 Clear a tab at the current column.
- 24 Clear all tabs.
- 25 Jump to next tab.
- 26 Fill screen area with a specified char. **This control code is not implemented yet.**
  - P1 – Char.
  - P2 – X start.            P4 – X end.
  - P3 – Y start.            P5 – Y end.

- 27    –
- 28    Set terminal window size. The minimum size is 10x4, the maximum is 80x25 (MSX: 80x24). After the window has been resized, the screen will be cleared and the cursor placed in the upper left corner (1/1).  
       P1 – Width.  
       P2 – Height.
- 29    Scroll window up or down one line. This will not influence the current cursor position.  
       P1 – Direction (1 → up, 2 → down).
- 30    Move cursor to the upper left corner (1/1).
- 31    Move cursor to a specified screen location.  
       P1 – X pos (1~80).  
       P2 – Y pos (1~25).

### 5.7.3 – Extended ASCII Codes

|                    |               |               |
|--------------------|---------------|---------------|
| 136 – cursor up    | 154 – Alt + C | 172 – Alt + U |
| 137 – cursor down  | 155 – Alt + D | 173 – Alt + V |
| 138 – cursor left  | 156 – Alt + E | 174 – Alt + W |
| 139 – cursor right | 157 – Alt + F | 175 – Alt + X |
| 140 – F0           | 158 – Alt + G | 176 – Alt + Y |
| 141 – F1           | 159 – Alt + H | 177 – Alt + Z |
| 142 – F2           | 160 – Alt + I | 178 – Alt + 0 |
| 143 – F3           | 161 – Alt + J | 179 – Alt + 1 |
| 144 – F4           | 162 – Alt + K | 180 – Alt + 2 |
| 145 – F5           | 163 – Alt + L | 181 – Alt + 3 |
| 146 – F6           | 164 – Alt + M | 182 – Alt + 4 |
| 147 – F7           | 165 – Alt + N | 183 – Alt + 5 |
| 148 – F8           | 166 – Alt + O | 184 – Alt + 6 |
| 149 – F9           | 167 – Alt + P | 185 – Alt + 7 |
| 150 – F.           | 168 – Alt + Q | 186 – Alt + 8 |
| 151 – Alt + @      | 169 – Alt + R | 187 – Alt + 9 |
| 152 – Alt + A      | 170 – Alt + S |               |
| 153 – Alt + B      | 171 – Alt + T |               |

### 5.7.4 – Keyboard Scan Codes

The scan code are used in the "Device\_KeyTest" function. Please note, that they are equal on all supported platforms.

|                   |              |            |                     |
|-------------------|--------------|------------|---------------------|
| 00 – Cursor Up    | 20 – F4      | 40 – 8     | 60 – S              |
| 01 – Cursor Right | 21 – Shift   | 41 – 7     | 61 – D              |
| 02 – Cursor Down  | 22 – \       | 42 – U     | 62 – C              |
| 03 – F9           | 23 – Control | 43 – Y     | 63 – X              |
| 04 – F6           | 24 – ^       | 44 – H     | 64 – 1              |
| 05 – F3           | 25 – –       | 45 – J     | 65 – 2              |
| 06 – Enter        | 26 – @       | 46 – N     | 66 – Esc            |
| 07 – F.           | 27 – P       | 47 – Space | 67 – Q              |
| 08 – Cursor Left  | 28 – ;       | 48 – 6     | 68 – Tab            |
| 09 – Alt          | 29 – :       | 49 – 5     | 69 – A              |
| 10 – F7           | 30 – /       | 50 – R     | 70 – Capslock       |
| 11 – F8           | 31 – .       | 51 – T     | 71 – Z              |
| 12 – F5           | 32 – 0       | 52 – G     | 72 – Joystick Up    |
| 13 – F1           | 33 – 9       | 53 – F     | 73 – Joystick Down  |
| 14 – F2           | 34 – O       | 54 – B     | 74 – Joystick Left  |
| 15 – F0           | 35 – I       | 55 – V     | 75 – Joystick Right |
| 16 – Clr          | 36 – L       | 56 – 4     | 76 – Fire 2         |
| 17 – [            | 37 – K       | 57 – 3     | 77 – Fire 1         |
| 18 – Return       | 38 – M       | 58 – E     | 78 – [not used]     |
| 19 – ]            | 39 – ,       | 59 – W     | 79 – Del            |

## 5.8 – SYSTEM CONFIGURATION

The SYMBOS.INI file is divided into 5 parts:

- Header, which contains the identifier and the length of three following parts
- Core area part, which contains data loaded in the first RAM bank
- Data area part, which contains additional data usually loaded in a different RAM bank
- Transfer area part (currently empty)
- Font

### 5.8.1 – Header

- 0000 2B Identifier, which also contains the version of the config file [byte0]="S", [byte1]=1 (current version).
- 0002 1W Length of the header (=8 bytes) plus the core area part of the SymbOS system configuration (will be always loaded to RAM-bank 0).



- 0004 1W Length of the data area part (the RAM-bank depends on the computer platform)
- 0006 1W Length of the transfer area part (not used, always 0)

## 5.8.2 – Core Area Part

### 5.8.2.1 – Mass storage devices

- 0000 128B Device configuration; this consists of 8 data records at 16 bytes for each device
- 00 1B Drive letter (upper case) or 0, if device slot is empty.
- 01 1B bit0–3: Type (0=Floppy, 1=IDE/SCSI) → Driver slot.  
bit4–6: Reserved (set to 0).  
bit7: Flag, if removeable media (1=yes).
- 02 1B Sub drive:  
→ If the device is a floppy disc:  
bit0–1: Drive.  
bit2: Head.  
bit3: Flag, if double step.  
bit4–7: Reserved (set to 0).  
→ If the device is an IDE/SCSI/SD device:  
bit0–3: partition (0=not partitioned).  
bit4–7: IDE → channel (master=0, slave=1).  
SCSI → sub device (0~15).
- 03 1B Reserved (set to 0).
- 04 12B Device name (terminated by 0).

### 5.8.2.2 – Display and miscellaneous (1)

- 0128 17W Colour palette (the border is defined by the 17th word)  
For each entry:  
bit0–3: Blue component.  
bit4–7: Green component.  
bit8–11: Red component.
- 0162 1B Screen mode:
- |                      |                      |
|----------------------|----------------------|
| 0 PCW (768x255x2)    | 7 MSX (512x212x16)   |
| 1 CPC,EP (320x200x4) | 8 G9K (384x240x16)   |
| 2 CPC,EP (640x200x2) | 9 G9K (512x212x16)   |
| 5 MSX (256x212x16)   | 10 G9K (768x240x16)  |
| 6 MSX (512x212x4)    | 11 G9K (1024x212x16) |

- 0163 32B System path.
- 0195 1B Time zone (−12 to +12).
- 0196 1B Background type (0~15=plain colour, −1=background graphic).
- 0197 32B Background graphic path and filename, terminated by 0 (only, if “background type” = −1).

### 5.8.2.3 – Keyboard (1) and mouse

- 0229 1B Keyboard delay (in 1/50s; between first and second char).
- 0230 1B Keyboard repeat speed (delay between every following chars).
- 0231 1B Joystick mouse delay (until mouse reaches full speed).
- 0232 1B Joystick mouse speed (in pixel)
- 0233 1B Mouse speed (CPC–SYMBiFACE [PS/2] and MSX) factor ( $\text{final\_movement} - \text{original\_movement} * \text{mouse\_speed} / 16$ ).
- 0234 1B Mouse double click delay (maximum time in 1/50s, when a double click is recognized)
- 0235 1B Flag, if swap left/right mouse keys.
- 0236 1B Mouse wheel speed (currently only CPC–SYMBiFACE [PS/2] and MSX)

### 5.8.2.4 – Miscellaneous (2) and Desktop Links

- 0237 1B SYMBOS.INI drive ("A", ...)
- 0238 1B Miscellaneous flags.  
bit0: Autosave config.
- 0239 1B Flag (1), if SymbOS extension module should be loaded.
- 0240 1B Flag for extended hardware (+1=Mouse, +2=Real time clock, +4=IDE/SCSI interface, +16=M4Board).
- 0241 1B Virtual desktop (0=no virtual desktop,  
bit0–3 → X–resolution, 1=512, 2=1000,  
bit4–7 → Y–resolution, not yet defined).
- 0242 1B Number of desktop icons.
- 0243 1B Number of start menu/programs entries.
- 0244 1B Number of taskbar short-cut entries (currently not supported)

- 0245 1B Machine type:  
           0=CPC 464                               7=MSX1  
           1=CPC 664                               8=MSX2  
           2=CPC 6128                              9=MSX2+  
           3=CPC 464+                             10=MSX turboR  
           4=CPC 6128+                           12=PCW8xxx  
           6=Enterprise                          13=PCW9xxx
- 0246 16W Desktop icon positions; for each of the 8 icons there are two words, the first contains the X-, the second the Y-position.
- 0278 32B Path and filename of the autoexec command line file.
- 0310 1B Flag, if autoexec command line file should be executed.

### 5.8.3 – Data Area Part

#### 5.8.3.1 – Desktop Links (2)

- 0000 400B Start menu program entry names (20 entries at 20 bytes, each terminated by 0).
- 0400 640B Start menu program entry pathes and filenames (20 entries at 32 bytes, each terminated by 0).
- 1040 256B Desktop icon pathes and filenames (8 entries at 32 bytes, each terminated by 0).
- 1296 192B Desktop icon names (8 entries, each consists of 2 lines at 12 bytes, each line is terminated by 0).
- 1488 1176B Desktop icon graphics (8 entries, each consists of the 3 bytes graphic header and the 144 byte (6\*24) bitmap).
- 2664 768B File extension association (16 entries at 48 bytes)
- 00 3B Extension 1 (uppercase; if byte0=1, then the whole entry is not defined).
  - 03 3B Extension 2 (if byte0=1, then this one entry is not defined).
  - 06 3B Extension 3 (s.a.).
  - 09 3B Extension 4 (s.a.).
  - 12 3B Extension 5 (s.a.).
  - 15 33B Application path and filename, which will be started, if a file with one of the above listed extensions has been opened.

### 5.8.3.2 – Screen Saver

- 3432 1B Flag, if screen saver is present.
- 3433 1B Duration of user inactivity, after which the screen saver will be started.
- 3434 33B Screen saver application path and filename (terminated by 0).
- 3467 64B Screen saver specific configuration data (can be stored and read here).

### 5.8.3.3 – Keyboard (2)

- 3531 80B Keyboard definition (normal).
- 3611 80B Keyboard definition (shift).
- 3691 80B Keyboard definition (control).
- 3771 80B Keyboard definition (alt).

### 5.8.3.4 – Security

- 3851 16B Security username.
- 3867 16B Security password.
- 3883 1B Security flags [not used yet, set to 0].

## 5.9 – SCREENSAVER APPLICATIONS

This is a list of commands, which will be sent to the screen saver application. Usually they will be sent by the desktop manager or by the control panel. The screensaver must be able to handle these commands and one additional response message for a proper interaction.

### ID: 001 (MSC\_SAV\_INIT) – ScreenSaver\_Init\_Command

Description: The caller process, which has started the screensaver (usually the desktop manager or the control panel) has sent an initialisation command. The screensaver now should store the sender process ID to be able to send a configuration response message later (see MSR\_SAV\_CONFIG). Then it has to copy the configuration data into its own memory area. This data can have a size of up to 64 bytes and is stored in the SYMBOS.INI file together with the other system settings. If the screensaver requires more than 64 bytes for its configuration it has to manage its own config file.

Library: ScrSav\_MAIN.  
 Message: 00 1B 001.  
           01 1B Config data (64 byte) RAM bank (0~7).  
           02 1W Config data (64 byte) address.  
 Response: No response from the screensaver expected.

#### **ID: 002 (MSC\_SAV\_START) – ScreenSaver\_Start\_Command**

Description: The caller process asks the screensaver to start its animation. The animation should be shown as long as no key has been pressed and the mouse hasn't been moved.  
 Library: ScrSav\_MAIN  
 Message: 00 1B 002.  
 Response: No response from the screensaver expected.

#### **ID: 003 (MSC\_SAV\_CONFIG) – ScreenSaver\_Config\_Command**

Description: The caller process asks the screensaver to open a configuration dialogue. In such a window the user has the possibility to modify the screensaver settings. If there is nothing to configure at all, the screensaver can ignore this command or just open an info window.  
 Library: ScrSav\_MAIN.  
 Message: 00 1B 003.  
 Response: See MSR\_SAV\_CONFIG.

#### **ID: 004 (MSR\_SAV\_CONFIG) – ScreenSaver\_Config\_Response**

Description: The user has finished modifying the settings and clicked on the “OK” button of the configuration dialogue.  
 Library: ScrSav\_CFGSAV  
 Message: 00 1B 001.  
           01 1B Config data (64 byte) RAM bank (0~7).  
           02 1W Config data (64 byte) address.

## **5.10 – SYMBOS MEMORY MAP**

### **5.10.1 – General Memory Usage**

The following diagram shows, in which way the different memory banks and blocks are used in SymbOS.

|       | Bank 0  | Bank 1                             | Bank n                   |
|-------|---|------------------------------------|--------------------------|
| FFFFH | System data<br>System manager                                       | Free                               | Free                     |
| C000H |   |                                    |                          |
| BFFFH | Buffers<br>SubRoutines<br>DeviceManager<br>ScreenManager            | Free                               | Free                     |
| 8000H |   |                                    |                          |
| 7FFFH | DesktopManager  | Free                               | Free                     |
| 4000H |   |                                    |                          |
| 3FFFH | DesktopManager<br>SystemManager<br>FileManager-LL<br>Kernel / jumps | FileManager-<br>HL<br>Kernel jumps | Free<br><br>Kernel jumps |
| 0000H |   |                                    |                          |

### 5.10.2 – Application Memory Usage

The memory inside an application RAM bank (1–n) is used in the following way:

1. 0000–03FF Kernel jumps, Kernel multitasking and banking routines.
2. 0400–FFFF Application code and internal application data.
3. 0400–3FFF Application data used by the screen manager.  
4000–7FFF (One object has to be inside one 16K block).  
8000–BFFF  
C000–FFFF
4. C000–FFFF Application "transfer" data, used by the desktop manager, message buffer, stack.

### 5.10.3 – Memory Configurations

The following diagram shows, how the memory is configured during the activity of one of the modules of SymBOS.

|       | DesktopManager<br>(C1) | ScreenManager<br>(C4-7) | FileManager-HL<br>(C4) |
|-------|------------------------|-------------------------|------------------------|
| FFFFH | Bank n<br>Block 3      | Bank 0<br>Block 3       | Bank 0<br>Block 3      |
| C000H | Transfer RAM           |                         |                        |
| BFFFH | Bank 0<br>Block 2      | Bank 0<br>Block 2       | Bank 0<br>Block 2      |
| 8000H |                        | ScreenManager           |                        |
| 7FFFH | Bank 0<br>Block 1      | Bank n<br>Block m       | Bank 1<br>Block 0      |
| 4000H | DesktopManager         | Data RAM                | FileManager-HL         |
| 3FFFH |                        |                         |                        |
| 0000H | Bank 0<br>Block 0      | Bank 0<br>Block 0       | Bank 0<br>Block 0      |

|       | FileManager-LL<br>(**) | Application<br>(C2) |
|-------|------------------------|---------------------|
| FFFFH | Bank 0<br>Block 3      | Bank n<br>Block 3   |
| C000H |                        | Trnf, Code, Data    |
| BFFFH | Slot x,y<br>Disk-ROM   | Bank n<br>Block 2   |
| 8000H |                        | Code, Data          |
| 7FFFH | Bank n<br>Block m      | Bank n<br>Block 1   |
| 4000H | DataRAM                | Code, Data          |
| 3FFFH | Bank 0<br>Block 0      | Bank n<br>Block 0   |
| 0000H | FileManager-LL         | Code, Data          |

## 5.11 – SCREEN MANAGER

The screen manager contains all routines for the direct access of the video hardware. There is currently only one function, that can be used by applications as well.

**TXLEN (815DH) – Screen\_TextLength**

**Description:** Returns the width and height of a textline in pixels, if it would be printed to the screen. You can define the text length (number of chars) in lY. If the text is terminated by 0 or 13 you should use -1 for the maximal text length. Please note, that this function always uses the system font for calculating the width and height.

How to call: `rst 20H : dw 815DH.`

Input: HL – Text address.

A – Text RAM bank (1~15).

IY – Maximal number of chars (text length).

Output: DE – Text width in pixels.

A – Text height in pixels.

Registers: F, BC, HL, IX.

## 5.12 – NETWORK DAEMON

The SymbOS network daemon provides all services for full network access. It's running as a shared service process [...]

### 5.12.1 – Configuration

Config\_Get    CFGGET   001 130   A – type, E, HL – data buffer  
→ (buffer has been filled)

```
Config_Set      CFGSET      002 131  A - type, E,HL - config data
→ (config has been set)
```

### 5.12.2 – Transportation Layer Services

```
TCP_Open  TCPOPn  016 144  A – mode, HL – local port
                               (IX,IY – remote IP, DE – remote port)
                               CY=0 → ok, A – handle
```



|                |        |     |     |  |
|----------------|--------|-----|-----|--|
| TCP_Close      | TCPCLO | 017 | 145 | A – handle<br>CY=0 → ok, A – handle  |
| TCP_Status     | TCPSTA | 018 | 146 | A – handle<br>CY=0 → ok, A – handle, L – status<br>( BC – received bytes,<br>IX,IY – remote IP, DE – remote port)  |
| TCP_Receive    | TCPRCV | 019 | 147 | A – handle, BC – length,<br>E,HL – memory<br>CY=0 → ok, A – handle,<br>BC – number of remaining bytes,<br>Z=1 → all bytes have been received                           |
| TCP_Send       | TCPSND | 020 | 148 | A – handle, BC – length,<br>E,HL – memory<br>CY=0 → ok, A – handle,<br>BC – number of sent bytes,<br>HL – number of remaining bytes,<br>Z=1 → all bytes have been sent |
| TCP_Skip       | TCPSKP | 021 | 149 | A – handle, BC – length<br>CY=0 → ok, A – handle   |
| TCP_Flush      | TCPFLS | 022 | 150 | A – handle<br>CY=0 → ok, A – handle  |
| TCP_Disconnect | TCPDIS | 023 | 151 | A – handle<br>CY=0 → ok, A – handle  |
| TCP_Event      | TCPEVT |     | 159 | A – handle, L – status<br>(BC – received bytes, IX,IY – remote<br>IP, DE – remote port)  |
| UDP_Open       | UDPOPN | 032 | 160 | HL – local port, E – memory bank<br>CY=0 → ok, A – handle  |
| UDP_Close      | UDPCLO | 033 | 161 | A=handle<br>CY=0 → ok, A – handle  |
| UDP_Status     | UDPSTA | 034 | 162 | A=handle<br>CY=0 → ok, A – handle, L – status<br>(BC – received bytes, IX,IY – remote<br>IP, DE – remote port)   |
| UDP_Receive    | UDPRCV | 035 | 163 | A – handle, HL – memory<br>CY=0 → ok, A – handle   |
| UDP_Send       | UDPSND | 036 | 164 | A – handle, BC – length,<br>HL – memory, IX,IY – remote IP,<br>DE=remote port<br>CY=0 → ok, A=handle   |

|           |        |     |     |   |
|-----------|--------|-----|-----|---|
| UDP_Skip  | UDPSKP | 037 | 165 | A – handle<br>CY=0 → ok, A – handle   |
| UDP_Event | UDPEVT |     | 175 | A – handle, L – status<br>(BC – received bytes, IX,IY – remote<br>IP, DE – remote port) |

### 5.12.3 – Application Layer Services

|             |        |     |     |  |
|-------------|--------|-----|-----|--|
| DNS_Resolve | DNSRSV | 112 | 240 | E,HL – address<br>CY=0 → Ok, IX,IY – IP  |
| DNS_Verify  | DNSVFY | 113 | 241 | E,HL – address<br>A – type of address (0 → no valid<br>address, 1 → IP address,<br>2 → domain address) |

## 5.13 – SYMBOS CONSTANTS

### 5.13.1 – Process-IDs

|                |       |                          |
|----------------|-------|--------------------------|
| PRC_ID_KERNEL  | equ 1 | Kernel process.          |
| PRC_ID_DESKTOP | equ 2 | Desktop manager process. |
| PRC_ID_SYSTEM  | equ 3 | System manager process.  |

### 5.13.2 – Messages

|               |         |  |
|---------------|---------|--|
| MSC_GEN_QUIT  | equ 0   | Application is beeing asked, to quit<br>itself.      |
| MSC_GEN_FOCUS | equ 255 | Application is beeing asked, to focus its<br>window. |

### 5.13.3 – Kernel Commands

|                |       |  |
|----------------|-------|--|
| MSC_KRL_MTADDP | equ 1 | Add process (P1/2=stack, P3=priority<br>(7 high – 1 low), P4=RAM bank (0~8)) |
| MSC_KRL_MTDELP | equ 2 | delete process (P1=ID)   |
| MSC_KRL_MTADDT | equ 3 | add timer (P1/2=stack,<br>P4=RAM bank (0~8))                                 |
| MSC_KRL_MTDELT | equ 4 | delete timer (P1=ID)   |
| MSC_KRL_MTSLEP | equ 5 | set process to sleep mode  |
| MSC_KRL_MTWAKP | equ 6 | wake up process  |

|                |       |   |
|----------------|-------|---|
| MSC_KRL_TMADDT | equ 7 | add counter service (P1/2=address, P3=RAM bank, P4=process, P5=frequency) |
| MSC_KRL_TMDELT | equ 8 | delete counter service (P1/2=address, P3=RAM bank)                        |
| MSC_KRL_TMDELP | equ 9 | delete all counter services of one process (P1=process ID)                |

### 5.13.4 – Kernel Responses

|                 |         |  |
|-----------------|---------|--|
| MSR_KRL_MTADDP  | equ 129 | process has been added (P1=0/1→ok/failed, P2=ID)         |
| MSR_KRL_MTDELP  | equ 130 | process has been deleted                                 |
| MSR_KRL_MTADDT  | equ 131 | timer process has been deleted (P1=0/1→ok/failed, P2=ID) |
| MSR_KRL_MTDELT  | equ 132 | timer has been removed                                   |
| MSR_KRL_MTSLEPP | equ 133 | process is sleeping now                                  |
| MSR_KRL_MTWAKP  | equ 134 | process has been waked up                                |
| MSR_KRL_TMADDT  | equ 135 | counter service has been added (P1=0/1→ok/failed)        |
| MSR_KRL_TMDELT  | equ 136 | counter service has been deleted                         |
| MSR_KRL_TMDELP  | equ 137 | all counter services of a process have been deleted      |

### 5.13.5 – System Commands

|                |        |  |
|----------------|--------|--|
| MSC_SYS_PRGRUN | equ 16 | load application or document (P1/2=address filename, P3=RAM bank filename) |
| MSC_SYS_PRGEND | equ 17 | quit application (P1=ID)   |
| MSC_SYS_SYSWNX | equ 18 | open dialogue to change current window (next) (-)                          |
| MSC_SYS_SYSWPR | equ 19 | open dialogue to change current window (previously) (-)                    |
| MSC_SYS_PRGSTA | equ 20 | open dialogue to load application or document (-)                          |
| MSC_SYS_SYSSEC | equ 21 | open system security dialogue (-)  |
| MSC_SYS_SYSQIT | equ 22 | open shut shown dialogue (-)   |
| MSC_SYS_SYSOFF | equ 23 | shut down (-)  |

|                |        |   |
|----------------|--------|---|
| MSC_SYS_PRGSET | equ 24 | start control panel (P1=submodule → 0=main window, 1=display settings, 2=date/time)   |
| MSC_SYS_PRGTSK | equ 25 | start taskmanager (-)   |
| MSC_SYS_SYSFIL | equ 26 | call filemanager function (P1=number, P2-13=AF, BC, DE, HL, IX, IY.)  |
| MSC_SYS_SYSHLP | equ 27 | start help (-)  |
| MSC_SYS_SYSCFG | equ 28 | call config function (P1=number, 0=load, 1=save, 2=reload background)   |
| MSC_SYS_SYSWRN | equ 29 | open message/confirm window (P1/2=adresse, P3=RAM bank, P4=number of buttons)   |
| MSC_SYS_PRGSRV | equ 30 | shared service function (P4=type [0=search, 1=start, 2=release], P1/2=address 12char ID, P3=RAM bank 12char ID or P3=program ID, if type=2) |
| MSC_SYS_SELOPN | equ 31 | open fileselect dialogue (P6=filename RAM bank, P8/9=filename address, P7=forbidden attributes, P10=max entries, P12=max buffer size)       |

### 5.13.6 – System Responses

|                |         |  |
|----------------|---------|--|
| MSR_SYS_PRGRUN | equ 144 | application has been started (P1=result → 0=ok, 1=file doesnt exist, 2=file is not executable, 3=error while loading [P8=filemanager error code], 4=mem. full, P8=app ID, P9=process ID) |
| MSR_SYS_SYSFIL | equ 154 | filemanager function returned (P1=num, P2-13=AF, BC, DE, HL, IX, IY)   |
| MSR_SYS_SYSWRN | equ 157 | message/confirm window response (P1 → 0=already in use, 1=opened [P2=number], 2=ok, 3=yes, 4=no, 5=cancel/close)   |
| MSR_SYS_PRGSRV | equ 158 | shared service function response (P1=state [5=not found, other codes see MSR_SYS_PRGRUN], P8=app ID, P9=process ID)  |

MSR\_SYS\_SELOPN    equ 159    message from fileselect dialogue  
(P1 → 0=Ok, 1=cancel, 2=already in  
use, 3=no memory free, 4=no window  
free, -1=open ok, modal window has  
been opened [P2=number])

### 5.13.7 – Desktop Commands

MSC\_DSK\_WINOPN    equ 32    open window (P1=RAM bank,  
P2/3=address data record)

MSC\_DSK\_WINMEN    equ 33    redraw menu bar (P1=window ID)  
[only if focus]

MSC\_DSK\_WININH    equ 34    redraw window content (P1=window ID,  
P2=-1/-Num/Object, P3=Object)  
[only if focus]

MSC\_DSK\_WINTOL    equ 35    redraw window toolbar (P1=window ID)  
[only if focus]

MSC\_DSK\_WINTIT    equ 36    redraw window title (P1=window ID)  
[only if focus]

MSC\_DSK\_WINSTA    equ 37    redraw window status lien  
(P1=window ID) [only if focus]

MSC\_DSK\_WINMVX    equ 38    set content X offset (P1=window ID,  
P2/3=XPos) [only if focus]

MSC\_DSK\_WINMVY    equ 39    set content Y offset (P1=window ID,  
P2/3=XPos) [only if focus]

MSC\_DSK\_WINTOP    equ 40    takes window to the front  
(P1=window ID) [always]

MSC\_DSK\_WINMAX    equ 41    maximize window (P1=window ID)  
[always]

MSC\_DSK\_WINMIN    equ 42    minimize window (P1=window ID)  
[always]

MSC\_DSK\_WINMID    equ 43    restore window size (P1=window ID)  
[always]

MSC\_DSK\_WINMOV    equ 44    moves window to a new position  
(P1=window ID, P2/3=XPos, P4/5=YPos)  
[always]

MSC\_DSK\_WINSIZ    equ 45    resize the window (P1=window ID,  
P2/3=XPos, P4/5=YPos) [always]

|                |        |  |
|----------------|--------|--|
| MSC_DSK_WINCLS | equ 46 | closes and removes window (P1=window ID) [always]  |
| MSC_DSK_WINDIN | equ 47 | redraw window content, even if it has not focus (P1=window ID, P2=-1/-Num/Object, P3=Object) [always]                          |
| MSC_DSK_DSKSRV | equ 48 | desktop service request (P1=type, P2-P5=parameters)  |
| MSC_DSK_WINSLD | equ 49 | redraw window scrollbars (P1=window ID) [only if focus]  |
| MSC_DSK_WINPIN | equ 50 | redraw window content part (P1=window ID, P2=-1/-Num/Object, P3=Object, P4/5=Xbeg, P6/7=Ybeg, P8/9=Xlen, P10/11=Ylen) [always] |
| MSC_DSK_WINSIN | equ 51 | redraw content of a super control (P1=window ID, P2=super control ID, P3=SubObject) [always]                                   |

### 5.13.8 – Desktop Responses

|                |         |   |
|----------------|---------|---|
| MSR_DSK_WOPNER | equ 160 | open window failed; the maximum of 32 windows has been reached  |
| MSR_DSK_WOPNOK | equ 161 | open window successfull (P4=number)   |
| MSR_DSK_WCLICK | equ 162 | window has been clicked (P1=window number, P2=action, P3=subspecification, P4/5,P6/7,P8/9=parameters)   |
| MSR_DSK_DSKSRV | equ 163 | desktop service answer (P1=type P2-P5=parameters)   |
| MSR_DSK_WFOCUS | equ 164 | window got/lost focus (P1=window number, P2=type [0=blur, 1=focus])                                     |
| MSR_DSK_CFOCUS | equ 165 | control focus changed (P1=window number, P2=control number, P3=reason [0=mouse click/wheel, 1=tab key]) |
| MSR_DSK_WRESIZ | equ 166 | window has been resized (P1=window number)  |
| MSR_DSK_WSCROL | equ 167 | window content has been scrolled (P1=window number)   |
| MSR_DSK_EXTDSK | equ 168 | command for extended desktop (used internally; P1=command, P2-x=parameters)                             |

|                |         |   |
|----------------|---------|---|
| FNC_DXT_DSKBGR | equ 001 | background has been updated   |
| FNC_DXT_FILRUN | equ 002 | file has been opened via prgrun<br>(P2/3=address, P4=bank)  |
| FNC_DXT_FILBRW | equ 003 | file has been selected via file browser<br>(P2/3=address, P4=bank)                                |
| FNC_DXT_MENCLK | equ 004 | startmenu has been clicked<br>(P2/3=value)  |
| FNC_DXT_DSKCLK | equ 005 | desktop window has been clicked<br>(P2=action, P3=subspecification,<br>P4/5,P6/7,P8/9=parameters) |

### 5.13.9 – Shell Commands

|                |        |   |
|----------------|--------|---|
| MSC_SHL_CHRINP | equ 64 | char is requested (P1=channel<br>[0 → Standard, 1 → Keyboard])  |
| MSC_SHL_STRINP | equ 65 | line is requested (P1=channel<br>[0 → Standard, 1 → Keyboard],<br>P2=RAM bank, P3/4=address)                |
| MSC_SHL_CHROUT | equ 66 | char should be writtten (P1=channel<br>[0 → Standard, 1 → Screen], P2=char)                                 |
| MSC_SHL_STROUT | equ 67 | line should be writtten (P1=channel<br>[0 → Standard, 1 → Screen], P2=RAM<br>bank, P3/4=address, P5=length) |
| MSC_SHL_EXIT   | equ 68 | application released focus or quit itself<br>(P1 → 0=quit, 1=blur)  |

### 5.13.10 – Shell Responses

|                |         |  |
|----------------|---------|--|
| MSR_SHL_CHRINP | equ 192 | char has been received (P1=EOF-flag<br>[0=no EOF], P2=char, P3=error status) |
| MSR_SHL_STRINP | equ 193 | line has been received (P1=EOF-flag<br>[0=no EOF], P3=error status)          |
| MSR_SHL_CHROUT | equ 194 | char has been written (P3=error status)                                      |
| MSR_SHL_STROUT | equ 195 | line has been written (P3=error status)                                      |

### 5.13.11 – Screensaver Messages

|              |       |  |
|--------------|-------|--|
| MSC_SAV_INIT | equ 1 | initialises the screen saver<br>(P1=bank of config data, P2/3=address<br>of config data [64bytes]) |
|--------------|-------|--|

|                |       |  |
|----------------|-------|--|
| MSC_SAV_START  | equ 2 | start screen saver   |
| MSC_SAV_CONFIG | equ 3 | open screen savers own config window<br>(at the end the screen saver has to send<br>the result back to the sender)   |
| MSR_SAV_CONFIG | equ 4 | returns user adjusted screen saver<br>config data (P1=bank of config data,<br>P2/3=address of config data [64bytes]) |

### 5.13.12 – Desktop Actions

|                 |        |   |
|-----------------|--------|---|
| DSK_ACT_CLOSE   | equ 5  | close button has been clicked or<br>ALT+F4 has been pressed   |
| DSK_ACT_MENU    | equ 6  | menu entry has been clicked<br>(P8/9=menu entry value)  |
| DSK_ACT_CONTENT | equ 14 | a control of the content has been<br>clicked (P3=sub spec [see dsk_sub...],<br>P4=key or P4/5=Xpos within the<br>window, P6/7=Ypos, P8/9=control value) |
| DSK_ACT_TOOLBAR | equ 15 | A control of the toolbar has been clicked<br>(see DSK_ACT_CONTENT)  |
| DSK_ACT_KEY     | equ 16 | Key has been pressed without touching/<br>modifying a control (P4=ASCII Code)   |
| DSK_SUB_MLCLICK | equ 0  | left mouse button has been clicked  |
| DSK_SUB_MRCLICK | equ 1  | right mouse button has been clicked   |
| DSK_SUB_MDCLICK | equ 2  | double click with the left mouse button   |
| DSK_SUB_MMCLICK | equ 3  | middle mouse button has been clicked  |
| DSK_SUB_KEY     | equ 7  | keyboard has been clicked and did<br>modify/click a control (P4=ASCII Code)   |
| DSK_SUB_MWHEEL  | equ 8  | mouse wheel has been moved<br>(P4=Offset)   |

### 5.13.13 – Desktop Services

|                |       |   |
|----------------|-------|---|
| DSK_SRV_MODGET | equ 1 | get screen mode (output P2=mode,<br>P3=virtual desktop)             |
| DSK_SRV_MODSET | equ 2 | set screen mode (input P2=mode,<br>P3=virtual desktop)              |
| DSK_SRV_COLGET | equ 3 | get colour (input: P2=number,<br>output: P2=number, P3/4=RGB value) |



|                |       |   |
|----------------|-------|---|
| DSK_SRV_COLSET | equ 4 | set colour (input: P2=number, P3/4=RGB value)   |
| DSK_SRV_DSKSTP | equ 5 | Freeze desktop (input P2=type [0=Pen0, 1=Raster, 2=background, 255=no screen modification, switch off mouse]) |
| DSK_SRV_DSKCNT | equ 6 | continue desktop  |
| DSK_SRV_DSKPNT | equ 7 | clear desktop (Eingabe P2=Typ [0=Pen0, 1=Raster, 2=background])   |
| DSK_SRV_DSKBGR | equ 8 | initialize and redraw desktop background  |
| DSK_SRV_DSKPLT | equ 9 | redraw the complete desktop   |

### 5.13.14 – Jumps

|            |           |         |
|------------|-----------|---------|
| jmp_memsum | equ 8100H | MEMSUM  |
| jmp_sysinf | equ 8103H | SYSINF  |
| jmp_clcnum | equ 8106H | CLCNUM  |
| jmp_mtgcnt | equ 8109H | MTGCNT  |
| jmp_timget | equ 810CH | TIMGET  |
| jmp_timset | equ 810FH | TIMSET  |
| jmp_memget | equ 8118H | MEMGET  |
| jmp_memfre | equ 811BH | MEMFRE  |
| jmp_memsiz | equ 811EH | MEMSIZ  |
| jmp_meminf | equ 8121H | MEMINF  |
| jmp_bnrwd  | equ 8124H | BNKRWD  |
| jmp_bnrwd  | equ 8127H | BNKWWD  |
| jmp_bnrwt  | equ 812AH | BNKRBT  |
| jmp_bnrwt  | equ 812DH | BNKWBT  |
| jmp_bnrwt  | equ 812FH | BNKWBW  |
| jmp_bnrwt  | equ 8130H | BNKCOP  |
| jmp_bnrwt  | equ 8133H | BNKGET  |
| *empty*    | equ 8136H | *empty* |
| jmp_scrget | equ 8139H | SCRGET  |
| jmp_mosget | equ 813CH | MOSGET  |
| jmp_moskey | equ 813FH | MOSKEY  |
| jmp_bnk16c | equ 8142H | BNK16C  |
| jmp_keytst | equ 8145H | KEYTST  |
| jmp_keysta | equ 8148H | KEYSTA  |
| jmp_keyput | equ 814BH | KEYPUT  |

|            |           |                   |
|------------|-----------|-------------------|
| jmp_bufput | equ 814EH | BUFPUT            |
| jmp_bufget | equ 8151H | BUFGET            |
| jmp_bufsta | equ 8154H | BUFSTA            |
| jmp_iominp | equ 8157H | IOMINP (cpc only) |
| jmp_iomout | equ 815AH | IOMOUT (cpc only) |
|            |           |                   |
| jmp_bnkcll | equ FF03H | BNKCLL            |
| jmp_bnkret | equ FF00H | BNKRET            |

### 5.13.15 – Filemanager Functions (call via MSC\_SYS\_SYSFIL)

|                |         |
|----------------|---------|
| FNC_FIL_STOINI | equ 000 |
| FNC_FIL_STONew | equ 001 |
| FNC_FIL_STORLD | equ 002 |
| FNC_FIL_STODEL | equ 003 |
| FNC_FIL_STOINP | equ 004 |
| FNC_FIL_STOOUT | equ 005 |
| FNC_FIL_STOACT | equ 006 |
| FNC_FIL_STOINF | equ 007 |
| FNC_FIL_STOTRN | equ 008 |
|                |         |
| FNC_FIL_DEVDIR | equ 013 |
| FNC_FIL_DEVINI | equ 014 |
| FNC_FIL_DEVSET | equ 015 |
|                |         |
| FNC_FIL_FILINI | equ 016 |
| FNC_FIL_FILNEW | equ 017 |
| FNC_FIL_FILOPN | equ 018 |
| FNC_FIL_FILCLO | equ 019 |
| FNC_FIL_FILINP | equ 020 |
| FNC_FIL_FILOUT | equ 021 |
| FNC_FIL_FILPOI | equ 022 |
| FNC_FIL_FILF2T | equ 023 |
| FNC_FIL_FILT2F | equ 024 |
| FNC_FIL_FILLIN | equ 025 |
|                |         |
| FNC_FIL_DIRDEV | equ 032 |
| FNC_FIL_DIRPTH | equ 033 |

|                |         |
|----------------|---------|
| FNC_FIL_DIRPRS | equ 034 |
| FNC_FIL_DIRPRR | equ 035 |
| FNC_FIL_DIRREN | equ 036 |
| FNC_FIL_DIRNEW | equ 037 |
| FNC_FIL_DIRINP | equ 038 |
| FNC_FIL_DIRDEL | equ 039 |
| FNC_FIL_DIRRMD | equ 040 |
| FNC_FIL_DIRMOV | equ 041 |
| FNC_FIL_DIRINF | equ 042 |

## 6 – UZIX

### 6.1 – COMMANDS

#### 6.1.1 – Conventions

COMMAND NAME (type of command)

Format: Valid formats for the command

Function: Way of operating the command

Details: Describes some details about the format

Uzix commands are all loaded from disk. This guide describes all commands and utilities that are installed by default on UZIX 2.0.

##### 6.1.1.1 – Format Notations

<filename>

Filename in the form: dir1/dir2/file

<filenames>

Several filename in the form: dir1/dir2/file

<dirname>

Directory name in the form: /dir1/dir2/

[ ] Delimits optional parameter.

| It means that only one of the items can be used.

A <device> can be:

|               |  |
|---------------|--|
| fd0~fd7       | Disk drives.                           |
| null          | Null device.                           |
| lpr           | Printer.                               |
| tty/tty0~tty2 | Monitor.                               |
| console       | Keyboard.                              |
| mem/kmem      | Memory.                                |
| sga0~sga(n)   | Hard disk partitions.                  |
| sge(n)        | Hard disk partition where the UZIX is. |

## 6.1.2 – Commands Description

### **ADDUSER** (Administration Utility)

Format: adduser

Function: Add a user to the system.

### **ALIAS** (Shell Utility)

Format: alias [<name> [<command> [<command> ...]]]

Function: Presents or sets an alias command.

### **BANNER** (Uzix Utility)

Format: banner <message>

Function: Print a message in big chars.

### **BASENAME** (Shell Utility)

Format: basename <name> [suffix]

Function: Removes component orientation from a directory.

### **BOGOMIPS** (System Utility)

Format: bogomips

Function: Prints processing speed on BogoMips.

### **CAL** (Uzix Utility)

Format: cal [month] year

Function: Shows a calendar.

### **CAT** (Files Utility)

Format: cat <filenames>

Function: Concatenate files and print to standard output.

### **CD** (Files Utility)

Format: cd [<dirname>]

Function: Change directories.

### **CDIFF** (Text Utility)

Format: cdiff [-c n] <file1> <file2>

Function: Prints the difference between two files with context.

Details: [-c] Produces output containing n lines of context.

**CGREP** (Text Utility)

Format: `cgrep [-a n] [-b n] [-f] [-l n] [-n] [-w n] <pattern> [  
 [<arqs>...]`

Function: Search a string and print the lines where it was found.

Details: `[-a]` Number of lines to print after the found line.  
`[-b]` Number of lines to print before the found line.  
`[-f]` Suppress filename on output.  
`[-l]` Truncates lines at length n before comparison.  
`[-n]` Suppress linenumbers on output.  
`[-w]` Set the window size (same as `-a` e `-b`)

**CHGRP** (Files Utility)

Format: `chgrp <gid> <filename>`

Function: Changes the group owning user for each file.

**CHMOD** (Files Utility)

Format: `chmod <modo_ascii> | <modo_octal> <filenames>`

Function: Change file access permissions.

Details: The symbolic format (ASCII) for the mode is as follows:

`[ugoa] [+ | -] [rwx]`, where:

|                         |                        |                                    |
|-------------------------|------------------------|------------------------------------|
| <code>u</code> → user   | <code>a</code> → all   | <code>x</code> → record            |
| <code>g</code> → group  | <code>r</code> → read  | <code>+</code> → add permission    |
| <code>o</code> → others | <code>w</code> → write | <code>-</code> → remove permission |

The numeric format (octal) is the following:

1° octal digit: 1 – save image text of attributes  
 2 – group ID  
 4 – user ID  
 2° octal digit: 1 – execution  
 2 – write  
 4 – read

**CHOWN** (Files Utility)

Format: `chown <uid> <filename>`

Function: Changes the regular user and the group owning user to the specified file.

**CHROOT** (Files Utility)

Format: `chroot <dirname>`

Function: Change the root directory.

**CKSUM** (Files Utility)

Format: cksum [<filename> [filename ...]]

Function: Shows the checksum and file size.

**CLEAR** (Shell Utility)

Format: clear

Function: Clears the screen.

**CMP** (Files Utility)

Format: cmp <filename1> <filename2>

Function: Compare files.

**CRC** (Files Utility)

Format: crc [<filename> [filename ...]]

Function: Shows the checksum of the data files.

**CP** (Files Utility)

Format: cp [-pifsmrRvx] <filename1> <filename2>

cp [-pifsrRvx] <filename1> [<filename2>...] <dir>

Function: Copy files.

Details: [-p] Preserves all attributes of the original file.  
 [-i] Checks the destination for file with the same name.  
 [-f] Remove files in destination.  
 [-s] Copies only some attributes.  
 [-m] Copies multiple subdirectories into one.  
 [-r] Copy directories recursively.  
 [-R] Copies directories and treats special files as ordinary.  
 [-v] Displays file names before copying.  
 [-x] Skip directories that are on file systems other than where copying started.

**CPDIR** (Files Utility)

Format: cpdir [-ifvx] <dirname1> <dirname2>

Function: Copy directories.

Details: [-i] Checks the destination for file with the same name.  
 [-f] Remove files in destination.  
 [-v] Displays file names before copying.  
 [-x] Skips subdirectories that are on file systems other than where copying started.

**DATE** (Uzix Utility)

Format: date

Function: Displays the current system date and time.

**DD** (Files Utility)

Format: dd [if=<filename>] [of=<filename>] [ibs=<bytes>]  
 [obs=<bytes>] [bs=<bytes>] [cbs=<bytes>]  
 [files=<number>] [skip=<blocks>]  
 [seek=<blocks>] [count=<blocks>]  
 [conv={ascii | ebcdic | ibm | lcase  
 | ucase | swab | noerror | sync}]

Function: Copy file converting it.

Details: [if=<filename>] Read from file  
 [of=<filename>] Write to file  
 [ibs=<bytes>] Read <bytes> bytes at a time  
 [obs=<bytes>] Write <bytes> bytes at a time  
 [bs=<bytes>] Reads and writes <bytes> bytes at a time  
 [cbs=<bytes>] Converts <bytes> bytes at a time  
 [files=<num.>] Copies <num.> files  
 [skip=<blocks>] Skip <blocks> blocks of “bs” size at the beginning of the entry  
 [seek=<blocks>] Skip <blocks> blocks of “bs” size at start of output  
 [count=<blocks>] Copies only <blocks> of size “bs” into the input  
 conv=conversion[,conversion...] → converts the file according to the following arguments:

|         |   |
|---------|---|
| ascii   | Convert from EBCDIC to ASCII.             |
| ebcdic  | Convert from ASCII to EBCDIC.             |
| ibm     | Convert from ASCII to alternative EBCDIC. |
| lcase   | Converts all characters to lowercase.     |
| ucase   | Converts all characters to uppercase.     |
| swab    | Swaps a pair of input bytes.              |
| noerror | Continue after detecting an error.        |
| sync    | Completes a “bs” block with 00H bytes.    |



**DF** (Files Utility)

Format: `df [-ikn]`

Function: Print the free disk space in units of 512 bytes.

Details: `[-i]` List information used by inodes.

`[-k]` Print in units of 1 Kbyte.

`[-n]` Not access `/etc/mtab` to obtain information.

**DHRY** (System Utility)

Format: `dhry`

Function: Displays processing speed in dhrystones.

**DIFF** (Text Utility)

Format: `diff [-c | -e | -C n] [-br] <filename1> <filename2>`

Function: Print the difference between two files.

Details: `[-C n]` Produces output containing `n` lines of context.

`[-b]` Ignores white space in the comparison.

`[-c]` Produces an output containing 3 lines of context.

`[-e]` Produces an “ed-script” to convert.

`[-r]` Applies diff recursively.

**DIRNAME** (Shell Utility)

Format: `dirname <filename>`

Function: Print the filename suffix.

**DOSDEL** (Uzix Utility)

Format: `dosdel <drivedos><filenamedos>`

Function: Erase a file in MSXDOS disks.

**DOSDIR** (Uzix Utility)

Format: `dosdir [-lr] <drivedos>`

Function: List files of the an MSXDOS disk.

Details: `[-l]` Long listing.

`[-r]` Prints subdirectories recursively and descending.

**DOSREAD** (Uzix Utility)

Format: `dosread [-a] <drivedos><filenamedos> [<filenameuzix>]`

Function: Read file from MSXDOS disk.

Details: `[-a]` ASCII file.

**DOSWRITE** (Uzix Utility)

Format: doswrite [-a] <drivedos><filenamedos> [<filenameuzix>]

Function: Write a file to MSXDOS disk.

Details: [-a] ASCII file.

**DU** (Uzix Utility)

Format: du [-as] [-l n] <dirname> ...

Function: Print space occupied by directories and subdirectories.

Details: [-a] Print space used by all files.

[-s] Summary only.

[-l] List n subdirectories levels.

**ECHO** (Shell Utility)

Format: echo [-ne] [<string> [<string>...]]

Function: Print a text line.

Details: [-n] Does not feed a line at the end of the text

[-e] Enables interpretation of the following characters:

\a Alert (bell).

\b Backspace.

\c Suppress line feed.

\f Form feed.

\n New line.

\r Carriage return.

\t Horizontal tab.

\v Vertical tab.

\\ Ignores space in the text between \\ (backslash).

\nnn Print char of ASCII code nnn (octal).

\xnn Print char of ASCII code nn (hex).

**ED** (Text Utility)

Format: ed [-Ghs] [-p string] [arquivo]

Function: Execute a standard text editor.

-G Forces retrocompatibility.

-h Shows the program help.

-s Supress diagnostics.

-p Sets a command prompt.

**EXIT** (Administration Utility)

Format: exit [<status>]

Function: Exit the current session.

**FALSE** (Shell Utility)

Format: false

Function: Null; only returns with error status “1”.

**FGREP** (Text Utility)

Format: fgrep [-cfhlsv] [<string\_file>] [<string>] <filename>...

Function: Searches for a string and prints the lines where it was found.

Details: [-c] Prints only the number of lines.  
 [-f] Searches for string in file <filename>.  
 [-h] Omit file headers from output.  
 [-l] Lists file names only once.  
 [-n] Prints line numbers for each line.  
 [-s] Status only.  
 [-v] Print only lines without the <string>.

**FILE** (Uzix Utility)

Format: file <filename> [<filename>...]

Function: Makes an assumption about what type the file is.

**FLD** (Text Utility)

Format: fld -u -z\* -[b t s? i? fm1.n1,m2.n2] {<input\_file>  
 [ <output\_file> ] }

Function: Reads and concatenates fields from a file.

Details: [-?] Show help. Same as [-h].  
 -u Unzips tabs.  
 [-p] Compress tabs.  
 -z\* Skip the first \* spaces.  
 [-b] Skip the starting spaces of the field.  
 [-t] Removes excessive spaces from the field.  
 [-s?] Field separator on output will be “?”.  
 [-i?] Field separator on input will be “?”.  
 [-fm1.n1,m2.n2] Field definition:  
     m1.n1 = beginning of field and m2.n2 = end of field,  
     where m = number of fields and n = number of chars.  
 [-f#] Get the user input field.

**FORTUNE** (Uzix Utility)

Format: fortune

Function: Randomly prints a proverb.

**GREP** (Text Utility)

Format: `grep -cnfv [-p<padrão>] <filenames>`

Function: Searches for a string and prints the lines where found.

Details: `[-c]` Prints only the number of lines.

`[-f]` Print file names.

`[-n]` Prints line numbers for each line

`[-v]` Print only lines without the <string>

`[-p]` Sets the string (default). The following control characters can be used:

`x` Ordinary character.

`\` Quote any character.

`^` Start of line.

`$` End of line.

`.` Any character.

`:l` Lowercase.

`:u` Capital letters.

`:a` Alphabetical.

`:d` Digits (numeric).

`:n` Alphanumeric.

`:r` Russian characters.

`:s` Space.

`:t` Tab.

`:c` Control characters (except LF and TAB).

`:e` Starts sub-expression.

`*` Repeats zero or more.

`+` Repeats one or more.

`-` Optionally search for expression.

`[..]` Any of these (in the FROM-TO range).

`[^..]` Any except these.

`\nnn` Numeric value (C style).

**HEAD** (Text Utility)

Format: `head [-n] [<filenames> ...]`

Function: Prints the beginning of the text.

Details: `[-n]` Number of lines to print (the standard is 10).

**HELP** (Uzix Utility)

Format: `help`

Function: Prints some commands in their format.

**INIT** (Administration Utility)

Format: /bin/init

Function: Process startup control.

**KILL** (Uzix Utility)

Format: kill [-signal] pid [pid...]

Function: Ends system processes.

Details: [-signal] is a signal to be sent to a process that is running (eg HUP, INT, QUIT, KILL or 9).

**LOGIN** (Administration Utility)

Format: login <username>

Function: Start a session.

**LN** (Text Utility)

Format: ln [-ifsSmrRvx] <filename1> <filename2>

ln [-ifsSrRvx] <filename> [<filename>...] <dirname>

Function: Add links between files.

Details: [-i] Warn before removing existing destination files.

[-f] Removes existing destination files.

[-s] Add symbolic link.

[-S] Add symbolic link while trying normal link.

[-m] Interleaves trees.

[-r] Adds recursive link to directories.

[-R] Same as [-r].

[-v] Print file name before adding link.

[-x] Skips subdirectories that are on file systems other than where adding links started.

**LOGOUT** (Uzix Utility)

Format: logout

Function: Ends a session.

**LS** (Files Utility)

Format: ls [-1ACFLRacdfgiklqrstu] [<filename> [<filename>...]]

Function: List the directory contents.

Details: [-1] Use only one column in the output.

[-A] Lists all files except "." and "..".

- [-C] Sorts files in the listing (in columns).
- [-F] Does not identify the file type.
- [-L] Lists files by symbolic links.
- [-R] Lists the contents of directories recursively.
- [-a] Lists all files including "." and "..".
- [-c] Sorts files by change date.
- [-d] List directories like other files.
- [-f] Does not sort files and directories.
- [-g] Prints the name of the user who owns the group.
- [-i] Prints the inode number of files.
- [-k] Print file size in Kbytes.
- [-l] Print file attributes.
- [-q] Prints question marks in place of special characters.
- [-r] Sort files and directories in reverse order.
- [-s] Print file size in bytes.
- [-t] Sorts files by creation date.
- [-u] Sorts files by last access date.

### **MAN** (System Utility)

Format: `man -wqv [seção] <commandname>`

Function: Apresenta o on-line manual.

Details: `-w` Displays only manual with exact section/name.  
`-q` Silent mode, for faulty formatter commands.  
`-v` Formatted presentation mode (verbose).

### **MKDIR** (Files Utility)

Format: `mkdir [-p] [-m <mode>] <dirname>`

Function: Create directories.

Details: `[-p]` Create parent directories according to the mask.  
`[-m]` Sets the mode (0666 minus umask bits).

### **MKNOD** (Files Utility)

Format: `mknod [-m <mode>] <filename> {b | c | u} <major> <minor>`

Function: Create special files.

Details: `[-m]` Define the mode.  
`b` Bufferized file (block).  
`c/u` Not bufferized file (character).

**MORE** (Uzix Utility)

Format: `more <filenames>`

Function: Paging utility.

Details: When the prompt is present, use the following keys:

space Displays the next page.  
 return Displays the next line.  
 n Go to the next file if it exists.  
 p Go to the previous file if it exists.  
 q Quits the 'more' command.

**MOUNT** (Uzix Utility)

Format: `mount [-r] <device> <path>`

Function: Mounts the <device> in the specified <path>.

Details: [-r] Mounts in the read only mode.

**MV** (Files Utility)

Format: `mv [-isfmvx] <filename1> <filename2>`

`mv [-ifsvx] <filename> [<filename> ...] <dirname>`

Function: Rename or move files.

Details: [-i] Warn before overwriting files with same name.  
 [-f] Removes existing target files.  
 [-s] Creates symbolic link and does not move the file.  
 [-m] Merge directories without searching target directory.  
 [-v] Print file name before moving.  
 [-x] Skips subdirectories that are on file systems other than where file movement started.

**PASSWD** (Administration Utility)

Format: `passwd [<login>]`

Function: Change user password.

**PROMPT** (Shell Utility)

Format: `prompt <string>`

Function: Change the Uzix prompt.

**PS** (Uzix Utility)

Format: `ps [-] [lusrmhn]`

Function: Prints a process status report.

Details: [-l] Long format.  
 [-u] User format (username and start time).  
 [-s] Signal format.  
 [-m] Memory information.  
 [-a] Displays processes from other users as well.  
 [-h] No header.  
 [-r] Only running processes.  
 [-n] Numeric output for user.

### **PWD** (Shell Utility)

Format: pwd

Function: Prints the path of the current working directory.

### **QUIT** (Administration Utility)

Format: quit

Function: Ends current session.

### **REBOOT** (Administration Utility)

Format: reboot

Function: Restart the computer.

### **RM** (Files Utility)

Format: rm <filename>

Function: Remove files.

### **RMDIR** (Files Utility)

Format: rmdir [-p] <dirname>

Function: Remove directories.

Details: [-p] Remove parent directory if empty after removal from the specified directory.

### **SASH** (Application Utility)

Format: sash

Function: It's a kind of shell with built-in commands.

### **SET** (Administration Utility)

Format: [<name> [<value>]]

Function: Displays or sets environment variables.



**SLEEP** (Administration Utility)

Format: `sleep [<seconds>]`

Function: Makes the system “sleep” for <seconds> seconds.

**SU** (Administration Utility)

Format: `su [<username>]`

Function: Temporarily connect as superuser or other user.

**SOURCE** (Uzix Utility)

Format: `source <filename>`

Function: Displays the source of the file.

**SUM** (Files Utility)

Format: `sum [<filename> [<filename>...]]`

Function: Analyze the checksum and block counter of the file.

**SYNC** (Programming Utility )

Format: `sync`

Function: Unloads file system buffers.

**TAIL** (Text Utility)

Format: `tail [-c n | -n n] [-f] [<filename> [<filename>]]`

Function: Prints the last lines of a file.

Details: `[-c]` Print n characters.  
`[-f]` In FIFO or special file, read after EOF.  
`[-n]` Print n lines.

**TAR** (Files Utility)

Format: `tar [cxt] [voFfpD] <filenametape> [<filename> [<filename>...]]`

Function: Concatenate/extract files for storage.

Details: `[c]` Create new tar file.  
`[x]` Extract files from the tar archive.  
`[t]` Lists the contents of the tar file.  
`[v]` Verbose mode.  
`[o]` Defines original user and owner on extraction.  
`[F]` Ignores errors.  
`[f]` Next argument is the name of the tar file.  
`[p]` Restore file modes, ignore mask.  
`[D]` Don't recursively add directories.

**TEE** (Shell Utility)

Format: tee <filename>

Function: Reads from standard input and writes to a file.

**TIME** (Uzix Utility)

Format: time <command> [<command arguments>]

Function: Executes the command and prints the real time, user time, and system time (hours–minutes–seconds).

**TOP** (Uzix Utility)

Format: top [-d <delay>] [-q] [-s] [-i]

Function: Lists the most active processes.

Details: [-d] Specifies the time for screen refresh.  
 [-q] Specifies update without any delay.  
 [-s] Safe Mode (disables interactive commands).  
 [-i] Ignore idle processes.

**TOUCH** (Files Utility)

Format: touch [-c] [-d <time/date>] [-m] <filename>

Function: Change the time and date of files.

Details: [-c] Does not create files that do not exist.  
 [-d] Change to <time/date> instead of using current time/date. Format: HH:MM:SS DD:MM:YY.  
 [-m] Changes only the file modification time/date.

**TR** (Text Utility)

Format: tr from to [+<start>] [-<end>] [<inputfile> [<outputfile>]]

Function: Swaps characters in a file (transliterates).

Details: Escape Sequences:

|      |                          |    |                   |
|------|--------------------------|----|-------------------|
| :z   | Empty range              | :a | Same as a–zA–Z    |
| :l   | Same as a–z              | :u | Same as A–Z       |
| :m   | Same as á–n              | :b | Same as Ç–f       |
| :r   | Same as á–nÇ–f           | :d | Same as 0–9       |
| :n   | Same as a–zA–Z0–9        | :s | Same as \001–\040 |
| :. . | All ASCII range minus \0 |    |                   |

**TRACE** (Uzix Utility)

Format: trace {on}

Function: Trace mode?

**TRUE** (Shell Utility)

Format: true

Function: Null; only returns with error status "0".

**UMOUNT** (Uzix Utility)

Format: umount <device>

Function: Unmounts file system from the specified device.

**UMASK** (Uzix Utility)

Format: umask [<mask>]

Function: Remove masks.

**UNALIAS** (Shell Utility)

Format: unalias <name>

Function: Removes an alias command.

**UNAME** (Shell Utility)

Format: uname [-snrvma]

Function: Prints system information.

Details: [-m] Print machine type.  
 [-n] Prints client machine name on the network.  
 [-r] Print operating system distribution.  
 [-s] Print operating system name.  
 [-v] Print operating system version.  
 [-a] Prints all of the above items.

**UNIQ** (Text Utility)

Format: uniq [-cdnzN.M+L] [-<fields>] [+<letters>] [<filename>]

Function: Remove duplicate lines in sorted files.

Details: [-u] Only print unrepeated lines.  
 [-d] Only print duplicate lines.  
 [-c] Prints the number of times the line is repeated.  
 [-z] Same as -c, but prints in octal numbers.  
 [-N.M] Skip N words and M letters.  
 [+L] Compares only L letters.

**WC** (Text Utility)

Format: wc [-bhpw] [<filename>]

Function: Prints the number of bytes, words and lines in a file.

Details: [-b] Open file in binary mode.  
 [-h] Displays program help.  
 [-p] Page count.  
 [-w] Finds the maximum line width.

### **WHOAMI** (Shell Utility)

Format: whoami

Function: Prints the username associated with the current userid.

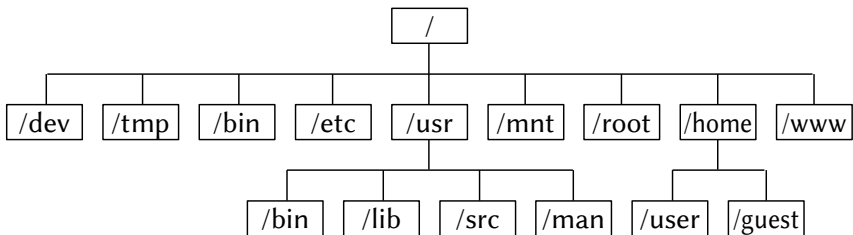
### **YES** (Shell Utility)

Format: yes [<string>]

Function: Prints “y” or <string> repeatedly to standard output.

## **6.2 – HIERARCHICAL STRUCTURE**

In Uzix there is a pre-defined structure of subdirectories. This structure can be modified by the user, but it is not advisable to do so because it is standard in the Unix world. This structure is as follows:



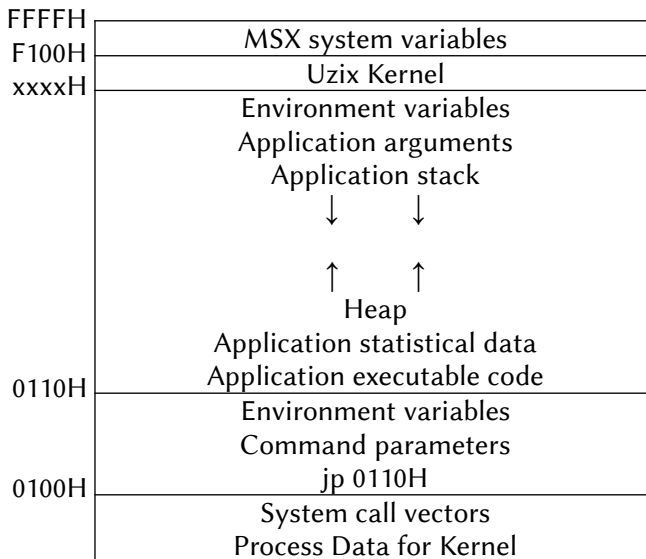
Each of these subdirectories has a specific, but not mandatory, use. A description of each is below.

|      |  |
|------|--|
| /    | Root directory   |
| /dev | Contains the special filenames associated with hardware or software devices. |
| /tmp | Used by all system for creating temporary files.                             |
| /bin | Contains the most generic applications on the system.                        |
| /etc | Files used to administer the system.   |
| /usr | General system files. This subdirectory contains more 4 subdirectories:      |

|        |  |
|--------|--|
| /bin   | Generic Applications.  |
| /lib   | Libraries.   |
| /src   | Source codes.  |
| /man   | System manuals (text files).   |
| /mnt   | Used as a connection point for a system of file from another device. Also used for mounting. |
| /root  | System administrator working directory.  |
| /home  | Used by regular users as their desktop.  |
| /user  | User “user”.   |
| /guest | User “guest”.  |
| /www   | Internet files.  |

### 6.3 – MEMORY MAPPING

Memory mapping is the biggest difference between Uzix 1.0 and 2.0. It is illustrated below, where xxxxH is 8000H for Uzix 1.0 and C000H for 2.0.



Uzix 1.0 is entirely resident in the high memory area, starting at address 8000H. Every process always occupies 32 Kbytes of memory. The

Uzix 2.0 has a resident part on page 3 (from C000H) and makes the additional calls from there. Each process can be 16K, 32K or 48K.

## 6.4 – SYSTEM CALLS

Uzix is an operating system for MSX that implements AT&T Unix Version 7 functionality. It is a multi-user system and implements preemptive multitasking, while also offering network infrastructure (TCP/IP). However, the following precautions must be taken:

- NEVER use DI and EI instructions;
- NEVER access the hardware directly;
- NEVER access data below 0100H or above the application.

To make a system call it is necessary to stack the parameters in the reverse order of the declaration, then the call number and then making a CALL 08H. It is the application's responsibility to unstack the parameters after the CALL. The 16-bit return value is placed in the DE register. The only exception is the lseek call, whose return value is 32 bits and is placed in HL:DE (HL is the most significant word). The table below lists the direct calls, their parameters and call number.

### 6.4.1 – Direct System Calls

**ACCESS (#00)** – Determines the access level of a file.

Syntax:     err = access (path, mode)

int     err

char   \*path

int     mode

Input:     path: String pointing to the file to be analyzed.

mode: 0 – Tests that the file exists and is searchable.

1 – Execute.

2 – Write.

4 – Read.

Output:    err: 0 → Successful test (if mode = 0).

          -1 → Error (error code in errno).

Assembler: (access = 33.)

sys access; name; mode

**ALARM (#01)** – Schedules a signal after a specified time.

Syntax:     time = alarm (secs)

          int     time

          int     secs

Input:       secs:   Time in seconds (maximum 32 767).

Output:      time:   Previous time remaining in alarm.

Assembler:   (alarm = 27.)

              (seconds in r0)

              sys alarm

              (previous amount in r0)

Note:        Causes the SIGALRM signal to be sent to the calling process after the number of seconds given by the argument. Unless captured or ignored, the signal ends the process.

              The return value is the amount of time remaining previously.

**BRK (#02)** – Change core allocation.

Syntax:      err = brk (addr)

          int     err

          char    \*addr

Input:       addr:   Address.

Output:      err:    0 → Command executed successfully.

              -1 → The program needs more memory than the system limit or overflows the maximum number of segmentation records.

Assembler:   (brk = 17.)

              sys break; addr

Note:        Defines, for the system, the lowest location not used by the program (called the range) for addr. Usually only growing programs whose data areas increase need to break.

**CHDIR (#03)** – Change default directory.

Syntax:      err = chdir (path)

          int     err

          char    \*path

Input:       path:   String of the directory to be defined.

Output:      err:    0 → Command executed successfully.

              -1 → “path” is not a directory or is not searchable.

Assembler:   (chdir = 12.)

              sys chdir; dirname

**CHMOD (#04)** – Change mode of file.

Syntax:     err = chmod (path, mode)

int     err  
char    \*path  
int     mode

Input:     path: String pointing to the file to be changed.  
            mode: Resulting from an OR combining the following values:

04 000 Set user ID on execution.  
02 000 Set group ID on execution.  
01 000 Save text image after execution.  
00 400 Read by owner.  
00 200 Write by owner.  
00 100 Execute (search on directory) by owner.  
00 070 Read, write, execute (search) by group.  
00 007 Read, write, execute (search) by others.

Output:    err:    0 → Command executed successfully.  
             -1 → File not found or user not allowed access.

Assembler: (chmod = 15.)  
            sys chmod; name; mode

**CHOWN (#05)** – Change owner and group of a file.

Syntax:     err = chown (path, owner, group)

int     err  
char    \*path  
int     owner  
int     group

Input:     path: String pointing to the file.  
            owner: New file user.  
            group: New file group.

Output:    err:    0 → Owner is changed.  
             -1 → Illegal owner changes.

Assembler: (chown = 16.)  
            sys chown; name; owner; group

**CLOSE (#06)** – Close a file.

Syntax:     err = close(path)

int     err  
char    \*path



Input: path: String pointo to the file to be closed.

Output: err: 0 → File succesfully closed.

-1 → Unknown file descriptor.

Assembler: (close = 6.)

(file descriptor in r0)

**GETSET (#07)** – Implements calls that read or change system variable values.

Syntax: var = getset(operation, ...)

int var

operation → Depends on the called function.

Input: getset(0) → getpid(void)

getset(1) → getppid(void)

getset(2) → getuid(void)

getset(3,uid) → setuid (uid)

int uid

getset(4) → geteuid(void)

getset(5) → getgid(void)

getset(6,gid) → setgid(int gid)

getset(7) → getegid(void)

getset(8) → getprio(void)

getset(9,pid,prio) → setprio(pid, prio)

int pid

char prio

getset(10) → umask(mask)

int mask

getset(11,onoff) → systrace(onoff)

int onoff

Output: It depends on the called function.

Assembler: It depends on the called function.

**DUP (#08)** – Duplicate an open file descriptor.

Syntax: newd = dup(oldd)

int newd

int oldd

Input: oldd: Old file descriptor.

Output: newd: New file descriptor.

-1 if the descriptor is invalid or if there are already too many files open.

Assembler: (dup = 41.)  
 (file descriptor in r0)  
 (new file descriptor in r1)  
 sys dup  
 (file descriptor in r0)

**DUP2 (#09)** – Duplicate an open file descriptor.

Syntax: err = dup2(oldd, newd)  
 int newd  
 int oldd

Nota: The dup2 entry is implemented by adding 0100 to oldd.

**EXECVE (#10)** – Execute a file.

Syntax: err = execve(name, argv, envp)  
 int err  
 char \*name  
 char \*\*argv  
 char \*\*envp

Input: name: Name of the file to be executed.  
 argv: Array of pointers to arguments.  
 envp: Pointer to an array of strings that constitute the process environment.

**EXIT (#11)** – Ends a process.

Syntax: param = exit(status)  
 int param  
 int status

Input: status: Lowest byte (LSB) is passed to “param”.

Output: param: Receives the lowest byte of “status”.

Assembler: (exit = 1.)  
 (status in r0)  
 sys exit

**FORK (#12)** – Generate a new process.

Syntax: newp = fork(void)  
 int newp

Input: None.

Output: newp: ID of the created process. If it returns -1, it failed in creating the process.

Assembler: (fork = 2.)  
 sys fork  
 (new process return)  
 (old process return, new process ID in r0)

**FSTAT (#13)** – Gets file information.

Syntax: stat = fstat(fd, \*buf)  
           int    stat  
           int    fd  
           void   \*buf  
 Library: #include <sys/types.h>  
           #include <sys/stat.h>  
 Input:   fd:    File descriptor.  
           \*buf:  Pointer to an empty buffer.  
 Output:  stat:  Information obtained. These are the same as the  
                   open, creat, dup or pipe commands.

**GETFSYS (#14)** – Get system information.

Syntax: stat = getfsys(dev, \*buf)  
           int    stat  
           int    dev  
           void   \*buf  
 Library: –  
 Input:   dev:    Device.  
           \*buf:  Pointer to an empty buffer.  
 Output:  stat:  Obtainet status.

**IOCTL (#15)** – Device control.

Syntax: err = ioctl(fd, req, ...)  
           int    err  
           int    fd  
           int    req  
 Library: #include <sgtty.h>  
 Input:   fd:    File descriptor.  
           req:    Request.  
 Output:  err:    0 → Command successful.  
                   -1 → The file descriptor does not refer to the type  
                       of file to which it was directed.  
 Assembler: (ioctl = 54.)  
               sys ioctl; fildes; request; argp

**KILL (#16)** – Sends the “sig” signal to the process specified in “r0”.

Syntax: `err = kill(pid, sig)`

`int err`

`int pid`

`int sig`

Input: `pid`: Process ID.

`sig`: Signal to be sent.

Output: `err`: 0 → The process was ended.

–1 → The process does not exist or does not have the same current userid or is not a superuser.

Assembler: `(kill = 37.)`

`(process number in r0)`

`sys kill; sig`

Note: If the process number is 0, the signal is sent to all other processes in the sender's process group.

**LINK (#17)** – Link to a file.

Syntax: `err = link(oldname, newname)`

`int err`

`char *oldname`

`char *newname`

Input: `oldname`: Old filename.

`newname`: New filename.

Output: `err`: 0 → Link successfully created.

–1 → Link creation failed.

Assembler: `(link = 9.)`

`sys link; oldname; newname`

**MKNOD (#18)** – Make a directory or a special file.

Syntax: `err = mknod(name, mode, dev)`

`int err`

`char *name`

`int mode`

`int dev`

Input: `name`: String pointing to the new file/directory.

`mode`: New file/directory mode.

`dev`: Device.

Output: `err`: 0 → File/directory created successfully.

–1 → File/directory already exists or user is not superuser.

Note: Only the superuser can use this command.

**MOUNT (#19)** – Mount the filesystem.

Syntax: `err = mount(spec, dir, rwflag)`

`int err`  
`char *spec`  
`char *dir`  
`int rwflag`

Library: `#include <sys/mount.h>`

Input: `spec: -`  
`dir: -`  
`rwflag: -`

Output: `err: 0` → Command executed successfully.  
`-1` → Error executing command.

**OPEN (#20)** – Open a file for read or write.

Syntax: `err = open(name, flags, mode)`

`int err`  
`char *name`  
`int flags`  
`int mode`

Input: `name:` Name of file to be open.  
`flags: -`  
`mode: 0` → Read only.  
`1` → Read/Write.

Output: `err: 0` → File successfully open.  
`-1` → Failed to open file.

Assembler: `(open = 5.)`  
`sys open; name; mode`  
`(file descriptor in r0)`

**PAUSE (#21)** – Pauses the system.

Syntax: `err = pause(void)`  
`int err`

Input: None.

Output: None.

Assembler: `(pause = 29.)`  
`sys pause`

Note: This command never returns normally. It is used to pause the system until it receives a “kill” or “alarm” signal.

**PIPE (#22)** – Create an interprocess channel.

Syntax:    err = pipe(fd)  
               int    err  
               int    \*fd

Input:      fd:    File descriptor.

Output:     err:    0 → Channel successfully created.  
               -1 → Channel creation failed.

Assembler: (pipe = 42.)  
               sys pipe  
               (read file descriptor in r0)  
               (write file descriptor in r1)

**READ (#23)** – Read from file.

Syntax:    err = read(fd, buf, bytes)  
               int    err  
               int    fd  
               void   \*buf  
               int    bytes

Input:      fd:    File descriptor.  
               buf:   Empty buffer.  
               bytes: Number of bytes to read.

Output:     err:    0 → End of file reached.  
               -1 → Read error.

Assembler: (read = 3.)  
               (file descriptor in r0)  
               sys read; buffer; nbytes  
               (byte count in r0)

**SBRK (#24)** – Change core allocation.

Syntax:    err = sbrk(int incr)

Nota:      Ver BRK (#02).

**LSEEK (#25)** – Move read/write pointer.

Syntax:    err = lseek(fd, offset, flag)  
               int    err  
               int    fd  
               long   offset  
               int    flag

Input:     fd:     File descriptor.  
           offset: Offset.  
           Flag:   0 → The pointer is set to offset bytes.  
                   1 → The pointer is set to its current location  
                       plus offset.  
                   2 → The pointer is set to the size of the file  
                       plus offset.

Output:    err:    0 → Command executed successfully.  
              -1 → Error executing command.

Assembler: (lseek = 19.)  
           (file descriptor in r0)  
           sys lseek; offset1; offset2; whence  
           [Offset1 and offset2 are the high and low offset words;  
           r0 and r1 contain the pointer on return].

### **SIGNAL (#26) – Catch or ignore signals.**

Syntax:    err = signal(sig\_num, (\*func)(int) )  
           int    err  
           char   sig\_num  
           void   (\*func)(int)

Input:     sig\_num:   Signal number.  
           (\*func)(int): –

Output:    The value (int)–1 is returned if the given signal is  
           out of range.

Note:      List of signals with names as in the “include” file.

|     |         |   |
|-----|---------|---|
| 1   | SIGHUP  | Hangup                                      |
| 2   | SIGINT  | Interrupt                                   |
| 3*  | SIGQUIT | Quit  |
| 4*  | SIGILL  | Illegal instruction (not reset when caught) |
| 5*  | SIGTRAP | Trace trap (not reset when caught)          |
| 6*  | SIGIOT  | IOT instruction                             |
| 7*  | SIGEMT  | EMT instruction                             |
| 8*  | SIGFPE  | Floating point exception                    |
| 9   | SIGKILL | Kill (cannot be caught or ignored)          |
| 10* | SIGBUS  | Bus error                                   |
| 11* | SIGSEGV | Segmentation violation                      |
| 12* | SIGSYS  | Bad argument to system call                 |

- |    |         |  |
|----|---------|--|
| 13 | SIGPIPE | Write on a pipe or link with no one to read it |
| 14 | SIGALRM | Alarm clock                                    |
| 15 | SIGTERM | Software termination signal                    |
| 16 |         | Unassigned                                     |

The starred signals in the list above cause a core image if not caught or ignored.

Assembler: (signal = 48.)  
 sys signal; sig; label  
 (old label in r0)

### **STAT (#27)** – Get file status.

Syntax:     err = stat(path, buf)  
           int     err  
           char    \*path  
           void    \*buf

Library:    #include <sys/types.h>  
              #include <sys/stat.h>

Input:       path:   Filename path.  
              buf:    Empty buffer.

Output:      err:    –

### **STIME (#28)** – Set system time.

Syntax:     err = stime(tvec)  
           int     err  
           int     \*tvec

Input:       tvec:   Time in seconds from 01/01/1970.

Output:      err:    0 → Date and time set successfully.  
               -1 → Error in setting date and time .

Assembler: (stime = 25.)  
 (time in r0–r1)  
 sys stime

### **SYNC (#29)** – Update super-block.

Syntax:     err = sync(void)  
           int     err

Input:       None.

Output:      None.

Assembler: (sync = 36.)  
 sys sync



**TIME (#30)** – Get time and date.

Syntax:    void time(tloc)  
               int    \*tloc

Library:    #include <sys/types.h>  
               #include <sys/timeb.h>

Input:       None.

Output:     tloc:   Time in seconds from 01/01/1970.

**TIMES (#31)** – Get process times.

Syntax:     t = times(struct tms \*tvec)  
               int    t

Input:       None.

Output:     See note.

Assembler: (times = 43.)  
               sys times; buffer

Note:       Returns time information for the current process and for terminated child processes of the current process. All times are in 1/Hz seconds, where Hz = 60 or Hz = 50. After the call, the buffer will appear as follows:

```
struct tbuffer {
    long proc_user_time;
    long proc_system_time;
    long child_user_time;
    long child_system_time;
};
```

**UMOUNT (#32)** – Unmount filesystem.

Syntax:     err = umount(spec)  
               int    err  
               char   \*spec

Input:       –

Output:     –

Assembler:  –

**UNLINK (#33)** – Remove directory entry.

Syntax:     err = unlink(path)  
               int    err  
               char   \*path

Input:       path:  String with directory to be removed.

Output:   err:   0  $\rightarrow$  Directory entry removed successfully.  
               -1  $\rightarrow$  Error removing directory.

```
Assembler: (unlink = 10.)
      sys unlink; name
```

## UTIME (#34) – Set file times.

```
Syntax:  err = utime(path, utimbuf *buf)
         int    err
         char   *path
         struct utimbuf *buf
```

Library: `#include <sys/types.h>`

Input: path: Filename path.  
Buf: -

Output: —

```
Assembler: (utime = 30.)
            sys utime; file; timep
```

**WAITPID (#35)** – Wait for process to change state.

```
Syntax:  err = waitpid(pid, statloc, options)
         int    err
         int    pid
         int    *statloc
         int    options
```

```
Library:    #include <sys/types.h>
            #include <sys/wait.h>
```

Input:      pid:      Process ID.  
             statloc: Wait status.

options: <-1 Meaning wait for any child process whose process group ID is equal to the absolute value of pid.

-1 Meaning wait for any child process.

0 Meaning wait for any child process whose  
process group ID is equal to that of the  
calling process.

- > 0 Meaning wait for the child whose process ID is equal to the value of pid.

**WRITE (#36)** – Write on a file.

Syntax:     err = write(fd, buf, nbytes)

int       fd  
void     \*buf  
int       nbytes

Input:     fd:     File descriptor.  
          buf:     Buffer with nbytes contiguous bytes which are  
                    written on the output file  
          nbytes:  Number of bytes to be written.

Output:    err:     0 → Writing successful.  
             -1 → Error during writing.

Assembler: (write = 4.)  
            (file descriptor in r0)  
            sys write; buffer; nbytes  
            (byte count in r0)

**REBOOT (#37)** – Restart system.

Syntax:     err = reboot(p1, p2)

int       err  
char     p1  
char     p2

Library:   #include <sys/reboot.h>  
            #include <unistd.h>

Input:     p1:     –  
          p2:     –

Output:    err:     0 → Not applicable.  
             -1 → Restart error.

**SYMLINK (#38)** – Create a new name for a file.

Syntax:     err = symlink(oldname, newname)

int       err  
char     \*oldname  
char     \*newname

Library:   #include <fcntl.h>  
            #include <unistd.h>

Input:     oldname: Old filename.  
          newname:  New filename.

Output:    err:     0 → Command executed successfully.  
             -1 → Error creating name.

**CHROOT (#39)** – Change root directory.

Syntax:     err = chroot(path)  
               int     err  
               char    \*path

Library:    #include <unistd.h>

Input:      path:    New path to root directory.

Output:     err:     0 → Command executed successfully.  
               -1 → Change error.

**MOD\_REG (#40)**

Syntax:     err = mod\_reg (sig, (func)())  
               int     err  
               int     sig  
               int     (\*func)()

**MOD\_DEREG (#41)**

Syntax:     err = mod\_dereg (sig)  
               int     err  
               int     sig

**MOD\_CALL (#42)**

Syntax:     err = mod\_call (sig, fnc, args, argsz)  
               int     err  
               int     sig  
               int     fnc  
               char    \*args  
               int     argsz

**MOD\_SENDSREPLY (#43)**

Syntax:     err = mod\_sendreply (pid, fnc, r, rsz)  
               int     err  
               int     pid  
               int     fnc  
               char    \*r  
               int     rsz

**MOD\_REPLY (#42)**

Syntax:     err = mod\_reply (sig, fcn, r)  
               int     err  
               int     sig  
               int     fcn  
               char    \*r

### 6.4.2 – Indirect System Call

**CREAT** – Create a new file.

Call:       creat(path, mode)  
               char   \*path  
               int     mode

Syntax:     err = open(path, 0x301, mode)

Input:       path: Filename path.  
               0x301: O\_CREAT | O\_TRUNC | O\_WRONLY  
               mode: Mode.

Output:      err:    0 → File created successfully.  
                      -1 → Error creating file.

Assembler: (creat = 8.)  
               sys creat; name; mode  
               (file descriptor in r0)

### 6.4.3 – Calls via GETSET

**GETPID** – Get process ID.

Call:       getpid(void)  
               id = getset(0)  
               Library: #include <unistd.h>  
               Input:   None.  
               Output:  id:    Process ID.  
               Assembler: (getpid = 20.)  
                           sys getpid  
                           (pid in r0)

**GETPPID** – Get parent process ID.

Call:       getppid(void)  
               id = getset(1)  
               Library: #include <unistd.h>  
               Input:   None.  
               Output:  id:    Process ID.

**GETUID** – Returns the real user ID of the current process.

Call:       getuid(void)  
               id = getset(2)  
               Library: #include <unistd.h>

Input: None.  
 Output: id: Process ID.  
 Assembler: (getuid = 24.)  
               sys getuid  
               (real user ID in r0, effective user ID in r1)

### **SETUID** – Set user and group ID.

Call: setuid(uid)  
 Syntax: err = getset(3, uid)  
           int err  
           int uid  
 Library: #include <unistd.h>  
 Input: uid: User process ID.  
 Output: err: 0 → ID successfully set.  
               -1 → Error setting ID.  
 Assembler: (setuid = 23.)  
               (user ID in r0)  
               sys setuid  
               (setgid = 46.)  
               (group ID in r0)  
               sys setgid

### **GETEUID** – Returns the effective user ID of the calling process.

Call: getuid(void)  
 Syntax: id = getset(4)  
 Library: #include <unistd.h>  
 Input: None.  
 Output: id: Process ID.

### **GETGID** – Get the real group ID.

Call: getgid(void)  
 Syntax: id = getset(5)  
 Library: #include <unistd.h>  
 Input: None.  
 Output: id: Group ID.  
 Assembler: (getgid = 47.)  
               sys getgid  
               (real group ID in r0, effective group ID in r1)

**SETGID** – Set group identity.

Call: `setgid(gid)`  
 Syntax: `err = getset(6, gid)`  
           int    err  
           int    gid  
 Library: `#include <unistd.h>`  
 Input:    gid:   Group ID.  
 Output:   err:   0 → ID set successfully.  
            -1 → Error in setting the ID.  
 Assembler: (setgid = 46.)  
              (group ID in r0)  
              sys setgid

**GETEGID** – Return the effective group ID of the calling process.

Call: `getegid(void)`  
 Syntax: `id = getset(7)`  
 Library: `#include <unistd.h>`  
 Input:    None.  
 Output:   id:    Group ID.

**GETPRIO** – Get the priority of a given process.

Call: `getprio(void)`  
 Syntax: `prd = getset(8)`  
 Library: `#include <sched.h>`  
 Input:    None.  
 Output:   prd:   Priority.

**SETPRIO** – Set the priority of a process.

Call: `setprio(pid, prio)`  
 Syntax: `err = getset(9, pid, prio)`  
           int    err  
           int    pid  
           char   prio  
 Library: `#include <unistd.h>`  
 Input:    pid:   Process ID.  
           prio:   Process priority.  
 Output:   err:   0 → Priority set successfully.  
            -1 → Error setting priority.

**UMASK** – Define a file creation mask.

Call: `umask(mask)`

Syntax: `oldm = getset(10, mask)`

`int oldm`

`int mask`

Input: `mask`: Mode. Only the lowest 9 bits are valid.

Output: `oldm`: Old mask value.

Assembler: (`umask = 60.`)

`sys umask; complmode`

**SYSTRACE** – Generate and apply protocols for system calls.

Call: `systrace(onoff)`

Syntax: `err = getset(11, onoff)`

`void err`

`int onoff`

Input: `onoff`: New protocol.

Output: –

#### 6.4.4 – TCP/IP module

The TCP/IP module implements a subset of IPv4 and allows Uzix to communicate with other systems that support the protocol. The TCP/IP module signature is 04950H, and it provides the functions listed below.

| Call                           | C prototype  | FNC# |
|--------------------------------|--|------|
| <code>ipconnect</code>         | <code>int ipconnect(char mode, ip struct t *ipstruct)</code>             | 1    |
| <code>ipgetc</code>            | <code>int ipgetc(uchar socknum)</code>                                   | 2    |
| <code>ipputc</code>            | <code>int ipputc(uchar socknum, uchar byte)</code>                       | 3    |
| <code>ipwrite</code>           | <code>int ipwrite(uchar socknum, uchar *bytes, int len)</code>           | 4    |
| <code>ipread</code>            | <code>int ipread(uchar socknum, uchar *bytes, int len)</code>            | 5    |
| <code>ipclose</code>           | <code>int ipclose(uchar socknum)</code>                                  | 6    |
| <code>iplisten</code>          | <code>int iplisten(int aport, uchar protocol)</code>                     | 7    |
| <code>ipaccept</code>          | <code>int ipaccept(ip struct t *ipstruct, int aport, uchar block)</code> | 8    |
| <code>ping</code>              | <code>int ping(uchar *IP, unsigned long *unused, uint len)</code>        | 9    |
| <code>setsockopttimeout</code> | <code>int setsockopttimeout(uchar socknum, uint timeout)</code>          | 10   |
| <code>ipunlisten</code>        | <code>int ipunlisten(int aport)</code>                                   | 11   |



|                |                                       |    |
|----------------|---------------------------------------|----|
| ipgetpingreply | icmpdata t*ipgetpingreply(void)       | 12 |
| gettcpinfo     | tcpinfo t*gettcpinfo(void)            | 13 |
| getsockinfo    | sockinfo t*getsockinfo(uchar socknum) | 14 |

The data type used are:

```
// protocol numbers (protocol for iplisten)
ICMP_PROTOCOL = 1
TCP_PROTOCOL  = 6
UDP_PROTOCOL  = 17

// open modes
TCP_ACTIVE_OPEN  = 255
TCP_PASSIVE_OPEN = 0

// protocols (ipconnect mode)
IPV4_TCP  = 1
IPV4_UDP  = 2
IPV4_ICMP = 3

// UDP modes
UDPMODE_ASC  = 1
UDPMODE_CKSUM = 2

// error codes
ECONTIMEOUT = 080H
ECONREFUSED = 081H
ENOPERM     = 082H
ENOPORT     = 083H
ENOROUTE    = 084H
ENOSOCK     = 085H
ENOTIMP     = 086H
EPROT       = 087H
EPORTINUSE  = 088H

// allowed states for sockstatus in sockinfo_t
TCP_CLOSED      = 000H
TCP_LISTEN      = 001H
TCP_SYN_SENT    = 042H
TCP_SYN_RECEIVED = 043H
TCP_ESTABLISHED = 0C4H
TCP_FIN_WAIT1   = 045H
```

```

TCP_FIN_WAIT2      = 046H
TCP_CLOSE_WAIT     = 087H
TCP_CLOSING        = 008H
TCP_LAST_ACK       = 009H
TCP_TIMEWAIT       = 00AH
UDP_LISTEN         = 091H
UDP_ESTABLISHED    = 094H

```

```

ip_struct_t = { uchar remote_ip[4],
                uint remote_port,
                uint local_port }

```

```

icmpdata_t = {  uchar type,
                uchar icmpcode,
                unsigned long unused,
                uchar data[28], /* pad para 64 bytes */
                uint len;
                uchar sourceIP[4],
                uchar ttl }

```

```

tcpinfo_t = {  uchar IP[4],
                uchar dns1ip[4],
                uchar dns2ip[4],
                char datalink[5],
                char domainname[DOMSIZE=128],
                int used_sockets,
                int avail_sockets,
                int used_buffers,
                int avail_buffers,
                int IP_chksum_errors }

```

```

sockinfo_t = {  int localport,
                int remoteport,
                uchar remote_ip[4],
                char socketstatus, /* bit 7: permissao
                                   de escrita
                                   bit 6: estado de
                                   listen
                                   bits 3-0: estado
                                   */
                char sockettype, /* TCP=1, UDP=2 */
                char sockerr, /* codigo de erro */
                int pid }

```

### 6.4.5 – Error codes

Uzix system calls return a value greater than 0 on success and less than 0 on error. The error code is placed in the global variable (defined in the stub of the Uzix programs) **errno**. Listed below are possible error codes.

|         |    |                           |
|---------|----|---------------------------|
| EPERM   | 1  | Operation not permitted   |
| ENOENT  | 2  | No such file or directory |
| ESRCH   | 3  | No such process           |
| EINTR   | 4  | Interrupted system call   |
| EIO     | 5  | I/O error                 |
| ENXIO   | 6  | No such device or address |
| E2BIG   | 7  | Arg list too long         |
| ENOEXEC | 8  | Exec format error         |
| EBADF   | 9  | Bad file number           |
| ECHILD  | 10 | No child processes        |
| EAGAIN  | 11 | Try again                 |
| ENOMEM  | 12 | Out of memory             |
| EACCES  | 13 | Permission denied         |
| EFAULT  | 14 | Bad address               |
| ENOTBLK | 15 | Block device required     |
| EBUSY   | 16 | Device or resource busy   |
| EEXIST  | 17 | File exists               |
| EXDEV   | 18 | Cross-device link         |
| ENODEV  | 19 | No such device            |
| ENOTDIR | 20 | Not a directory           |
| EISDIR  | 21 | Is a directory            |
| EINVAL  | 22 | Invalid argument          |
| ENFILE  | 23 | File table overflow       |
| EMFILE  | 24 | Too many open files       |
| ENOTTY  | 25 | Not a typewriter          |
| ETXTBSY | 26 | Text file busy            |
| EFBIG   | 27 | File too large            |
| ENOSPC  | 28 | No space left on device   |
| ESPIPE  | 29 | Illegal seek              |
| EROFS   | 30 | Read-only file system     |
| EMLINK  | 31 | Too many links            |

|              |         |                                     |
|--------------|---------|-------------------------------------|
| EPIPE        | 32      | Broken pipe                         |
| EDOM         | 33      | Math argument out of domain of func |
| ERANGE       | 34      | Math result not representable       |
| EDEADLK      | 35      | Resource deadlock would occur       |
| ENAMETOOLONG | 36      | File name too long                  |
| ENOLCK       | 37      | No record locks available           |
| EINVFNC      | 38      | Function not implemented            |
| ENOTEMPTY    | 39      | Directory not empty                 |
| ELOOP        | 40      | Too many symbolic links encountered |
| ESHELL       | 41      | It's a shell script                 |
| ENOSYS       | EINVFNC |                                     |

## 6.5 – VT-5 TERMINAL CODES

|        |         |  |
|--------|---------|--|
| Ctrl G | 07H     | Beep.  |
| Ctrl H | 08H     | Backspace.   |
| Ctrl I | 09H     | TAB.   |
| Ctrl J | 0AH     | Advances one line.                                     |
| Ctrl K | 0BH     | Move cursor to origin.                                 |
| Ctrl L | 0CH     | Clears screen and moves cursor to origin.              |
| Ctrl M | 0DH     | Carriage return.                                       |
| Ctrl \ | 1CH     | Advances cursor one position.                          |
| Ctrl ] | 1DH     | Moves cursor back one position.                        |
| Ctrl ^ | 1EH     | Move cursor up.  |
| Ctrl _ | 1FH     | Move cursor down.                                      |
|        | 7FH     | Deletes character and moves cursor to the left.        |
| Esc A  | 1BH,41H | Moves cursor up.                                       |
| Esc B  | 1BH,42H | Moves cursor down.                                     |
| Esc C  | 1BH,43H | Move cursor to the right.                              |
| Esc D  | 1BH,44H | Move cursor left.                                      |
| Esc E  | 1BH,45H | Clears screen and places cursor at origin.             |
| Esc H  | 1BH,48H | Places cursor at the origin.                           |
| Esc J  | 1BH,4AH | Erases to the end of the screen, does not move cursor. |
| Esc j  | 1BH,6AH | Clears screen and places cursor at origin.             |
| Esc K  | 1BH,4BH | Erases to end of line, do not move cursor.             |
| Esc L  | 1BH,4CH | Insert line above cursor, move rest of screen          |

|           |             |   |
|-----------|-------------|---|
|           |             | down, leave cursor at start of new line.  |
| Esc I     | 1BH,6CH     | Erases to end of line, do not move cursor.  |
| Esc M     | 1BH,4DH     | Erases cursor line, moves rest of screen to line and places cursor at beginning of next line. |
| Esc x 4   | 1BH,78H,34H | Selects block cursor.   |
| Esc x 5   | 1BH,78H,35H | Turns cursor off.   |
| Esc Y n m | 1BH,59H,m,n | Move cursor to column m-32 and row y-32.  |
| Esc y 4   | 1BH,79H,34H | Selects underlined cursor.  |
| Esc y 5   | 1BH,79H,35H | Turns on cursor.  |

## 7 – SYSTEM VARIABLES

### 7.1 – SYSTEM AREA FOR MSXDOS1

**F1C1H, 1** – Countdown timer for the drives. By setting this counter to 0, the drives' motors are stopped.

**F1C2H, 1** – Sub-counter of the countdown timer for the drive.

**F1C3H, 1** – Countdown counter sub-counter for the drive.

**F1C4H, 1** – Number of the currently active drive.

**F1C5H, 1** – Track number where the drive head A: is.

**F1C6H, 1** – Track number where the head of drive B: is.

**F1C7H, 1** – Logic drive active.

**F1C8H, 1** – Number of physical drives present.

**F1C9H, 24** – Routine for printing on the screen a string ending with “\$”. DE – Starting address of the string.

**F1CAH~F1E1H** – ?

**F1E2H, 6** – Routine to abort the program in case of error.

**F1E8H, 12** – Calls the address pointed by (HL) in RAM and returns with the DOS Kernel (BDOS) page active.

**F1F4H, 3** – Jump to the filename check routine.  
HL – Address of the first character of the filename.

**F1F7H, 4** – Device name “PRN”.

**F1FBH, 4** – Device name “LST”.

**F1FFH, 4** – Device name “NUL”.

**F203H, 4** – Device name “AUX”.

**F207H, 4** – Device name “CON”.

**F20BH, 11** – Reserved for new device or filenames.

**F216H, 1** – Current device number:

–5 → PRN; –4 → LST; –3 → NUL; –2 → AUX; –1 → CON.

**F217H~F220H** – ?

**F221H, 2** – FCB date of the current file.

**F223H, 2** – FCB time of the current file.

**F22BH, 12** – Table containing the number of days in the months.

|                     |                      |
|---------------------|----------------------|
| F22BH [31] January  | F231H [31] July      |
| F22CH [28] February | F232H [31] August    |
| F22DH [31] March    | F233H [30] September |
| F22EH [30] April    | F234H [31] October   |
| F22FH [31] May      | F235H [30] November  |
| F230H [30] June     | F236H [31] December  |

**F237H, 4** – Used internally by function 10 of BDOS.

**F23BH, 1** – Flag to indicate whether the characters should go to the printer. (0 = no; other value, yes)

**F23CH, 2** – Current DTA address.

**F23EH, 1** – ?

**F23FH, 4** – Current sector number of the disk.

**F243H, 2** – Pointer to the DPB address of the current drive.

**F245H, 1** – Current relative sector of the directory starting from the first (0).

**F246H, 1** – Drive that contains the current sector of the directory  
(0 = A ; 1 = B ; etc.)

**F247H, 1** – Default drive (0 = A ; 1 = B ; etc)

**F248H, 1** – Day

**F249H, 1** – Month

**F24AH, 1** – Year-1980 (add 1980 to obtain the correct year)

**F24BH, 1** – ?

**F24CH, 2** – Hour and minutes

**F24EH, 1** – Day of the week (0 = Sunday, 1 = Monday, etc.)

### **7.1.1 – Hooks called by disk routines**

**F24FH, 3** – Routine that displays the message “Insert disk for drive”.

A – Drive number (41H = A :, 42H = B :, etc)

**F252H, 3** – Get the FAT content.

**F255H, 3** – Filename repair routine.

**F258H, 3** – Directory search routine.

**F25BH, 3** – Increment the directory entry (last entry in A).

**F25EH, 3** – Routine that calculates the next sector of the directory.

**F261H, 3** – Filename repair routine.

**F264H, 3** – 'OPEN' function routine.

**F267H, 3** – Returns the last FAT.

**F26AH, 3** – 'GETDPB' routine of the disk interface (SFIRST).

**F26DH, 3** – Routine function 'CLOSE' (writes FAT).

**F270H, 3** – Routine function 'RDABS – 2FH' (HL = DMA, DE = Sector, B = number of sectors). H.DISKREAD.

**F273H, 3** – Error handling routine when accessing the disk.

**F276H, 3** – 'WRABS' function routine (writes sector).

**F279H, 3** – Routine function 'WRABS' (HL = DMA, DE = Sector, B = number of sectors).

**F27CH, 3** – Multiplication routine ( $HL = DE * BC$ ).

**F27FH, 3** – Division routine ( $BC = BC/DE$ ; HL = Rest).

**F282H, 3** – Returns the absolute cluster.

**F285H, 3** – Returns the next absolute cluster.



**F288H, 3** – Disk sector reading.

**F28BH, 3** – Sector writing on the disk.

**F28EH, 3** – Starts the operation of reading blocks (records) of the disk.

**F291H, 3** – Finalizes the option of reading blocks (records) of the disk.

**F294H, 3** – End of the operation of reading blocks (records) of the disk.

**F297H, 3** – Error in the operation with blocks (records).

**F29AH, 3** – Starts the operation of writing blocks (records) in the disk.

**F29DH, 3** – Finalizes the option of writing blocks (records) in the disk.

**F2A0H, 3** – Calculates sequential sectors.

**F2A3H, 3** – Gets the number of sectors in a cluster.

**F2A6H, 3** – Allocate a sequence of FATs.

**F2A9H, 3** – Releases a sequence of FATs.

**F2ACH, 3** – 'BUFIN' function (adds data to the buffer)

**F2AFH, 3** – 'CONOUT' function (BDOS 02H).

**F2B2H, 3** – Get the time and date of the file.

**F2B5H, 3** – February identification routine (28/29 days).

### **7.1.2 – Other DOS data**

**F2B8H, 1** – Number of the current directory entry.

**F2B9H, 11** – Filename / extension of the current file.

**F2C4H, 1** – Byte of file attributes of the last directory entry read.

If bit 7 is set, files with a NOT attribute of 0 can be opened.

This can be done by setting bit 7 of the FCB-drive byte, by calling the BDOS OPEN routine. (FCB+0).

**F2C5H~F2CEH** – ?

**F2CFH, 2** – Time of the current file.

**F2D1H, 2** – Date of the current file.

**F2D3H, 2** – Initial cluster of the current file.

**F2D5H, 4** – Size of the current file.

**F2D9H~F2DBH** – ?

**F2DCH, 1** – Files with attributes other than F2DCH are also accepted.  
(F2C4H bit-7 overrides that!)

**F2DDH~F2E0H** – ?

**F2E1H, 1** – Current drive for writing and reading absolute sectors.

**F2E2H~F2FDH** – ?

**F2FEH, 2** – Sub-counter from the countdown timer to the drive.

**F301H, 1** – ?

**F302H, 2** – Pointer to the MSXDOS abort handler routine.

**F304H, 2** – Stores the value of the SP (Stack Pointer) register.

**F306H, 1** – Default drive for MSXDOS (0 = A :, 1 = B :, etc).

**F307H, 2** – Stores the value of the DE register (FCB address).

**F309H, 2** – Used by the DPB for searching (First / Next).

**F30BH, 2** – Current sector of the directory.

**F30DH, 1** – Check flag (0 = Off; other value, on).

**F30EH, 1** – Date format (0- yymmdd; 1- mmdday; 2- ddmyy).

**F30FH, 4** – Area used by Kanji mode.

**F313H, 1** – Contains the version of the ROM of the MSXDOS.  
00H = version 1.x; 20H = version 2.0; 21H = 21H = version 2.1; etc.  
Obs: The Nextor returns 99H.

**F314H~F322H** – ?

**F323H, 2** – Address of the disk error handler.

**F325H, 2** – Address of the handler of the CTRL+C keys.

### **7.1.3 – Hooks for the 'COM:' port**

**F327H, 5** – Routine 'AUXINP' (A = byte read from the AUX device).

**F32CH, 5** – 'AUXOUT' routine (A = byte to be sent to the AUX device).

**F331H, 5** – Routine for manipulating BDOS functions.

### **7.1.4 – Keyboard**

**F336H, 1** – Key pressed flag. Contains FFH if any key is pressed and 03H for CTRL+STOP.

**F337H, 5** – Contains the ASCII code of the key pressed and 03H for CTRL+STOP pressed together.

### **7.1.5 – MSXDOS Variables**

**F338H, 1** – Flag to indicate the presence of an internal clock (0 = no; another value, yes).

**F339H, 7** – Routine used by the internal clock.

#### **F340H, 1 – REBOOT**

If it is 0, DOS will reset all variables again.

#### **F341H, 1 – RAMAD0**

Slot of page 0 of RAM (format equal to RDSLT – 000CH /BIOS).

#### **F342H, 1 – RAMAD1**

Slot of page 1 of RAM (format equal to RDSLT – 000CH /BIOS).

#### **F343H, 1 – RAMAD2**

Slot of page 2 of the RAM (format equal to RDSLT – 000CH /BIOS).

#### **F344H, 1 – RAMAD3**

Slot on page 3 of the RAM (format equal to RDSLT – 000CH /BIOS).

**F345H, 1** – Number of free buffers (025H).

**F346H, 1** – Flag to indicate whether the system was booted from MSXDOS on a floppy disk. (0 = no; other value, yes)

**F347H, 1 – NMBDRV**

Total number of logical drives in the system.

**F348H, 1 – MASTER**

DOS Kernel slot ID (format equal to RDSLT – 000CH /BIOS).

**F349H, 2 – HIMSAV**

Pointer to a copy of the FAT of the last connected logical drive (1.5 Kbytes) followed by a copy of the FAT of the next to last connected logical drive (1.5 Kbytes) and so on, up to drive A :. It also indicates the highest memory area available for DOS.

**F34BH, 2** – Final address of the MSXDOS Kernel (start for COMMAND.COM). The MSXDOS Kernel start address is stored at 0006H/0007H.

**F34DH, 2 – SECBUF**

Pointer to a copy of the FAT of the default drive (1.5K).

**F34FH, 2 – BUFFER**

Pointer to a 512-byte buffer used as Disk BASIC's DTA.

**F351H, 2 – DIRBUF**

Pointer to a 512-byte buffer used for transferring sectors of the disk (used by DSKI\$ and DSKO\$ of BASIC).

**7.1.6 – DPB addresses****F353H, 2 – DPBBASE**

Pointer to the DPB of the current file.

**F355H, 16 – DPBLIST**

F355H, 2 – DPB address of drive A :.

F357H, 2 – DPB address of drive B :.

F359H, 2 – DPB address of drive C :.

F35BH, 2 – DPB address of drive D :.

F35DH, 2 – DPB address of drive E :.

F35FH, 2 – DPB address of drive F :.

F361H, 2 – DPB address of drive G :.

F363H, 2 – DPB address of drive H :.

### 7.1.7 – Routines used by MSXDOS

**F365H, 3 –** Jump from the primary slot reading routine.  
(A – Primary slot state)

**F368H, 3 – SETROM**

Jump to the DOS Kernel (BDOS) switch routine on page 1 (not available from Disk BASIC)

**F36BH, 3 – SETRAM**

Jump to the RAM change routine on page 1 (not available from Disk BASIC).

### 7.1.8 – Inter-slot movement routines

**F36EH, 3 – SLTMOV**

Jump to LDIR from RAM on page 1 (not available from Disk BASIC).

**F371H, 3 – AUXINP**

Jump to the auxiliary device entry routine.  
Output: A – Value read (1AH when CTRL+Z).

**F374H, 3 – AUXOUT**

Jump to the auxiliary device exit routine.  
Input: A – Amount to send.

**F377H, 3 – BLDCHK**

Jump to the 'BLOAD' command routine. The address pointed to by F378H/F379H is the highest RAM address available for Disk BASIC. Contains JP 0000H under MSXDOS.

**F37AH, 3 – BSVCHK**

Jump to the 'BSAVE' command routine (Contains JP 0000H under MSXDOS). Input: c – Number of the routine to be called.

**F37DH, 3 – ROMBDOS**

Jump to BDOS command handler.

\*\*\* See also addresses F85FH to F87EH and FB20H to FB34H.

## **7.2 – SYSTEM AREA FOR MSXDOS2**

### **7.2.1 – Physical information about disks**

**F1C1H, 1** – Countdown timer for the drives. By setting this counter to 0, the drive motors are stopped.

**F1C2H, 1** – 1st sub-counter of the countdown timer for the drive.

**F1C3H, 1** – 2nd sub-counter of the countdown timer for the drive.

**F1C4H, 1** – Number of the currently active drive.

**F1C5H, 1** – Track number where the drive head A: is.

**F1C6H, 1** – Track number where the head of drive B: is.

**F1C7H, 1** – Logic drive active.

**F1C8H, 1** – Number of physical drives connected.

### **7.2.2 – Hooks called by disk routines (1)**

**F1C9H, 24** – Routine for printing on the screen a string ending with “\$”. DE – Starting address of the string.

**F1E2H~F1E4H** – ?

**F1E5H, 3** – Jump to the interrupt handler (only when processing BDOS functions).

**F1E8H, 3** – Jump to the BIOS routine 'RDSLT-000CH' (only when processing BDOS functions).

**F1EBH, 3** – Jump to the BIOS routine 'WRSLT-0014H' (only when processing BDOS functions).

**F1EEH, 3** – Jump to the BIOS routine 'CALSLT-001CH' (only when processing BDOS functions).

**F1F1H, 3** – Jump to the BIOS routine 'ENASLT-0024H' (only when processing BDOS functions).

**F1F4H, 3** – Jump to the BIOS routine 'CALLF-0030H' (only when processing BDOS functions).

- F1F7H, 3** – Jump to the routine for switching to “DOS Mode” (pages 0 and 2 for system segments).
- F1FAH, 3** – Jump to the switching routine to “User Mode”.
- F1FDH, 3** – Jump to the routine that selects the DOS Kernel segments on page 1.
- F200H, 3** – Jump to the routine that allocates a segment of 16 Kbytes of RAM.
- F203H, 3** – Jump to the routine that releases a segment of 16 Kbytes of RAM.
- F206H, 3** – Jump to the BIOS routine 'RDSLT-000CH'.
- F209H, 3** – Jump to the BIOS routine 'WRSLT-0014H'.
- F20CH, 3** – Jump to the BIOS routine 'CALSLT-001CH'.
- F20FH, 3** – Jump to the BIOS routine 'CALLF-0030H'.
- F212H, 3** – Jump to the routine that places a 16 Kbyte segment on the page indicated by HL.
- F215H, 3** – Jump to the routine that reads the page of the current 16 Kbytes segment. HL – Page read.
- F218H, 3** – Jump to the routine that enables the 16 Kbyte segment of the mapped memory on page 0.
- F21BH, 3** – Jump to the routine that reads the current 16 Kbytes segment of the mapped memory on page 0.
- F21EH, 3** – Jump to the routine that enables the 16 Kbyte segment of the mapped memory on page 1.
- F221H, 3** – Jump to the routine that reads the current 16 Kbytes segment of the mapped memory on page 1.
- F224H, 3** – Jump to the routine that enables the 16 Kbyte segment of the mapped memory on page 2.
- F227H, 3** – Jump to the routine that reads the current 16 Kbytes segment of the mapped memory on page 2.

**F22AH, 3** – Page 3 does not support segment change.

**F22DH, 3** – Jump to the routine that reads the current 16 Kbytes segment of the mapped memory on page 3.

**F230H~F23BH** – ?

### **7.2.3 – Logical information about disks**

**F23CH, 1** – Current logical drive (0 = A:, 1 = B:, etc.).

**F23DH, 2** – Current DTA address.

**F23FH, 4** – Current sector number for access.

**F243H, 2** – DPB address of the current drive.

**F245H, 1** – Relative number of the current sector of the directory area.

**F246H, 1** – Drive number of the current directory (0 = A :, 1 = B :, etc.).

**F247H, 1** – Default drive number (0 = A :, 1 = B :, etc.).

**F248H, 3** – +0 = Day / +1 = Month / +2 = Year-1980 (Add 1980 to obtain the correct year)

**F24CH, 1** – ?

**F24CH, 2** – Hour

**F24EH, 1** – Day of the week

### **7.2.4 – Hooks called by disk routines**

**F24FH, 3** – H.PROM

Jump to the routine that displays the message “Insert disk for drive”.  
A – Drive pain number (41H = A :, 42H = B :, etc)

**F252H, 3** – Hook called before the execution of a BDOS function.  
Page 0 – Block map (F2D0H). Page 2 – Block map (F2CFH).

**F255H, 3** – Hook of the filename repair routine.

**F258H, 3** – Hook of the disk BASIC subdirectory manipulation routine.  
Used by several other routines.



**F25BH, 3** – Hook of the routine that increments the directory entry.  
The new entry is stored in AF

**F25EH, 3** – Hook of the routine that loads the next sector of the directory.

**F261H, 3** – Hook of function 02H of BDOS.

**F264H, 3** – OPEN Routine

**F267H, 3** – Returns the last FAT

**F26AH, 3** – Looking for the first FCB (SFIRST)

**F26DH, 3** – Writes the FAT.

**F270H, 3** – Hook of the sector direct reading routine (function 2FH of BDOS). HL – DMA, DE – Initial sector, B – Number of sectors.

**F273H, 3** – Disk error.

**F276H, 3** – Write hook in the subdirectory sector (folder)

**F279H, 3** – Hook of the direct sector writing routine (BDOS function 30H). HL – DMA, DE – Initial sector, B – Number of sectors.

**F27CH, 3** – Hook of the multiplication routine ( $HL = DE * BC$ ).

**F27FH, 3** – Hook of the division routine ( $BC = BC/DE$ ;  $HL = \text{Rest}$ ).

**F282H~F282H** – ?

### 7.2.5 – MSXDOS2 variables

**F2B3H, 2** – User defined TPA address. The initial 32 bytes of the TPA are used for special functions:

| Off set | Description                              |
|---------|--|
| 00H~02H | Reserved                                 |
| 03H     | Used by VDP speed (bit 3 of F2B6H)       |
| 04H~1FH | Reserved                                 |
| 20H     | BDOS expansion and interruption routines |

**F2B5H, 1** – ?

**F2B6H, 1** – Byte of flags:

- b0~b2 – Reserved
- b3 – Fast VDP (0 = yes; 1 = no)
- b4 – User TPA address (0 = yes; 1 = no)
- b5 – Reset (0 = no; 1 = yes)
- b6 – BusReset (0 = yes; 1 = no)
- b7 – Reboot (0 = no; 1 = yes)

**F2B7H, 1** – Version number (usually 10H = v1.0).

**F2B8H, 1** – Number of the current directory entry.

**F2B9H~F2BFH** – ?

**F2C0H, 5** – Second hook of the interrupt routine (used by Disk-ROM).

**F2C5H, 2** – Mapping table address.

**F2C7H, 1** – Current logical page of the mapper on physical page 0.

**F2C8H, 1** – Current logical page of the mapper on physical page 1.

**F2C9H, 1** – Current logical page of the mapper on physical page 2.

**F2CAH, 1** – Current logical page of the mapper on physical page 3  
(cannot be changed).

**F2CBH, 1** – Copy of F2C7H during the execution of BDOS routines.

**F2CCH, 1** – Copy of F2C8H during the execution of BDOS routines.

**F2CDH, 1** – Copy of F2C9H during the execution of BDOS routines.

**F2CEH, 1** – Copy of F2CAH during the execution of BDOS routines.

**F2CFH, 1** – Number of the last available 16K block of the memory mapper. During the execution of the BDOS routines, the blocks are exchanged on page 2 (buffer segment).

**F2D0H, 1** – Number of the last available 16K block of the memory mapper. During the execution of the BDOS routines, the blocks are exchanged on page 0 (code segment).

**F2D1H~F2D4H** – ?

**F2D5H, 5** – Second EXTBIO hook (hook routine FCALL [FFCAH]).

**F2DAH, 4** – Address of the second BDOS ROM for handling functions.

**F2DEH, 4** – BDOS ROM address for handling functions.

**F2E2H~F2E5H** – ?

**F2E6H, 2** – Buffer used for temporary storage of register IX.

**F2E8H, 2** – Buffer used for temporary storage of the SP register.

**F2EAH, 1** – Status of the primary slots after the execution of a BDOS function.

**F2EBH, 1** – Same as F2EAH, but for secondary slots

**F2ECH, 1** – Flag for checking the disk status. (00H = Off, FFH = On).

**F2EDH~F2FAH** – ?

**F2FBH, 2** – Pointer to a temporary buffer when interpreting an error code.

**F2FDH, 1** – Drive from which MSXDOS2.SYS should be loaded.  
(01H = A :, 02H = B :, etc).

**F2FEH, 2** – Address of the top of the DOS buffer stack.

**F300H, 1** – Check flag (00H = Off, FFH = On).

**F301H~F30CH** – ?

**F30DH, 1** – Check disk flag (00H = Off, FFH = On).

**F30EH~F312H** – ?

**F313H, 1** – Contains the version of the ROM of the MSXDOS.  
00H = version 1.x; 20H = version 2.0; 21H = 21H = version 2.1; etc.  
Obs: The Nextor returns 99H.

**F314H~F322H** – ?

**F323H, 2** – DISKVE  
Disk error handler address.

**F325H, 2 – BREAKV**

Address of the CTRL+C key handler.

**F327H~F33CH – ?****F33DH, 3 – Jump to the BASIC 'LEN' command (random file access).****F341H, 1 – RAMD0**

Slot of page 0 of RAM (format equal to 'RDSLT' – 000CH / Main).

**F342H, 1 – RAMD1**

Slot of page 1 of RAM (format equal to 'RDSLT' – 000CH / Main).

**F343H, 1 – RAAD2**

Slot of page 2 of RAM (format equal to 'RDSLT' – 000CH / Main).

**F344H, 1 – RAAD3**

Slot of page 3 of RAM (format equal to 'RDSLT' – 000CH / Main).

**F345H, 1 – ?****F346H, 1 – MSXDOS**

Flag to indicate whether the system was booted from MSXDOS on a floppy disk. (0 = no; other value, yes)

**F347H, 1 – ?****F348H, 1 – MASTER**

Primary DOS Kernel slot ID (master). In the case of DOS2 it is the primary interface that contains the DOS2 ROM. The format is the same as RDSLT – 000CH /BIOS).

**7.2.6 – Pointers and buffers (FAT, DTA, FCB, DPB)****F349H, 2 – HIMSAV**

Pointer to a copy of the FAT of the last connected logical drive (1.5 Kbytes) followed by a copy of the FAT of the penultimate connected logical drive (1.5 Kbytes) and so on, up to drive A :. It also indicates the highest area of memory available to the user.

**F34BH~F34CH – ?****F34DH, 2 – SECBUF**

Pointer to a copy of the FAT of the default drive (1.5 Kbytes).

**F34FH, 2 – BUFFER**

Pointer to a 512-byte area used as the DTA of the Disk-BASIC.

**F351H, 2 – DIRBUF**

Pointer to a 512-byte buffer used for transferring sectors of the disk.

**F353H, 2 – FCBBASE**

Point to the FCB of the current file.

**F355H, 16 – DPBLIST**

List of pointers to the DPBs of all eight possible drives, reserving two bytes for each one.

|                     |                     |
|---------------------|---------------------|
| F355H, 2 – Drive A: | F35DH, 2 – Drive E: |
| F357H, 2 – Drive B  | F35FH, 2 – Drive F: |
| F359H, 2 – Drive C: | F361H, 2 – Drive G: |
| F35BH, 2 – Drive D: | F363H, 2 – Drive H: |

**F364H~F377H – ?****7.2.7 – System jumps****F378H – BLDCHK+1**

Routine address of the 'BLOAD' command handler.

**F37AH, 3 – Secondary jump to the system segment on p. 0.****F37DH – BDOS**

Jump to the BDOS function handler.

\*\*\* See also addresses F85FH to F87EH and FB20H to FB34H.

**7.3 – INTER-SLOT SUBROUTINES****RDPRIM (F380H)**

Function: Reads a byte from any address in any slot.

Input: A – Primary slot to be read  
D – Current return slot

Output: E – Byte read

Code: F380H RDPRIM: OUT (0A8H), A  
F382H LD E, (HL)  
F383H JR WRPRM1

**WRPRIM (F385H)**

Function: Writes a byte to any address in any slot.

Input: A – Primary slot to be read  
 D – Current return slot  
 E – Byte to be written

Output: None

Code: F385H WRPRIM: OUT (0A8H), A  
 F387H LD (HL), E  
 F388H WRPRM1: LD A, D  
 F389H OUT (0A8H), A

**CLPRIM (F38CH)**

Function: Calls an address in any slot.

Input: A – Primary slot containing the routine  
 IX – Address to be called  
 PUSH AF – Current return slot (in A)

Output: Depends on the routine called

Code: F38CH CLPRIM: OUT (0A8H), A  
 F38EH EX AF, AF'  
 F38FH CALL CLPRM1  
 F392H EX AF, AF'  
 F393H POP AF  
 F394H OUT (0A8H), A  
 F396H EX AF, AF'  
 F397H RET  
 F398H CLPRM1: JP (IX)

**7.4 – USR FUNCTION AND TEXT MODES****USRTAB (F39AH, 20)**

Initial value: FCERR

Content: There are ten system variables of two bytes each that point to the execution address of an assembly routine to be called by the USR function. The first position points to USR0, the second to USR1, and so on. The initial value points to the error generator routine.

**LINL40 (F3AEH, 1)**

Initial value: 39

Content: Screen width in Screen 0 text mode.

**LINL32** (F3AFH, 1)

Initial value: 29

Content: Screen width in Screen 1 text mode.

**LINLEN** (F3B0H, 1)

Initial value: 39

Content: Current width of the text screen.

**CRTCNT** (F3B1H, 1)

Initial value: 24

Content: Number of lines in text modes.

**CLMSLT** (F3B2H, 1)

Initial value: 14

Content: Horizontal location in the case of items divided by commas in the PRINT command.

**7.5 – AREA USED BY THE SCREEN****7.5.1 – Screen 0****TXTNAM** (F3B3H, 2)

Initial value: 0000H

Content: Address in the VRAM of the pattern name table.

**TXTCOL** (F3B5H, 2) – Variable not used**TXTCGP** (F3B7H, 2)

Initial value: 0800H

Content: Address in the VRAM of the character pattern table.

Note: In this variable lies the only bug found on MSX2 computers. When Screen 0 is given the command WIDTH up to 40, the value will be correct. However, if the WIDTH command is 41 to 80, the correct value will be 1000H, but this variable will continue with 0800H value. In this case, when working with an assembly program from BASIC, an ADD HL,HL instruction must be used to correct the value. In the MSX2+ and MSX turbo R models, the correct value for this variable is 0000H, so that the instruction shown does not affect compatibility, despite the fact that this bug does not exist in these models.

**TXATTR** (F3B9H, 2) – Variable not used

**TXTPAT** (F3BBH, 2)

Initial value: 0000H

Content: Variable not used

### 7.5.2 – Screen 1

**T32NAM** (F3BDH, 2)

Initial value: 1800H

Content: Address of the pattern name table.

**T32COL** (F3BFH, 2)

Initial value: 2000H

Content: Address in the VRAM of the color table.

**T32CGP** (F3C1H, 2)

Initial value: 0000H

Content: Address in VRAM of the pattern table.

**T32ATR** (F3C3H, 2)

Initial value: 1B00H

Content: Address in VRAM of the table of attributes of the sprites.

**T32PAT** (F3C5H, 2)

Initial value: 3800H

Content: Address in VRAM of the sprite pattern table.

### 7.5.3 – Screen 2

**GRPNAM** (F3C7H, 2)

Initial value: 1800H

Content: Address in the VRAM of the pattern name table.

**GRPCOL** (F3C9H, 2)

Initial value: 2000H

Content: Address in the VRAM of the color table.

**GRPCGP** (F3CBH, 2)

Initial value: 0000H

Content: Address in the VRAM of the pattern table.



**GRPATR** (F3CDH, 2)

Initial value: 1B00H

Content: Address in VRAM of the table of attributes of the sprites.

**GRPPAT** (F3CFH, 2)

Initial value: 3800H

Content: Address in VRAM of the sprite pattern table.

**7.5.4 – Screen 3****MLTNAM** (F3D1H, 2)

Initial value: 0800H

Content: Address of the pattern name table.

**MLTCOL** (F3D3H, 2) – Variable not used.**MLTCGP** (F3D5H, 2)

Initial value: 0000H

Content: Address in VRAM of the standards table.

**MLTATR** (F3D7H, 2)

Initial value: 1B00H

Content: Address in VRAM of the table of attributes of the sprites.

**MLTPAT** (F3D9H, 2)

Initial value: 3800H

Content: Address in VRAM of the sprite standards table.

**7.5.4 – Other Screen Values****CLIKSW** (F3DBH, 1)

Initial value: 1

Content: Turn key click on / off (0 = Off; other value, on). It can be changed by the SCREEN command.

**CSRY** (F3DCH, 1)

Initial value: 1

Content: Y (vertical) coordinate of the cursor in text modes.

**CSRX** (F3DDH, 1)

Initial value: 1

Content: X (horizontal) coordinate of the cursor in text modes.

**CNSDFG** (F3DEH, 1)

Initial value: 0

Content: Turns on / off the display of the function keys (0 = On, other value, off). It can be changed by the KEY ON/OFF command.

## 7.6 – VDP REGISTERS AREA

**RG0SAV** (F3DFH, 1)

Content: Copy of the VDP register R#0.

**RG1SAV** (F3E0H, 1)

Content: Copy of the VDP register R#1.

**RG2SAV** (F3E1H, 1)

Content: Copy of the VDP register R#2.

**RG3SAV** (F3E2H, 1)

Content: Copy of the VDP register R#3.

**RG4SAV** (F3E3H, 1)

Content: Copy of the VDP register R#4.

**RG5SAV** (F3E4H, 1)

Content: Copy of the VDP register R#5.

**RG6SAV** (F3E5H, 1)

Content: Copy of the VDP register R#6.

**RG7SAV** (F3E6H, 1)

Content: Copy of the VDP register R#7.

**STATFL** (F3E7H, 1)

Content: Copy of the VDP status register. On MSX2 or upper, stores the contents of the S#0 register.

### 7.6.1 – Area used for the V9938

**RG8SAV** (FFE7H, 1)

Content: Copy of the VDP register R#8.

**RG9SAV** (FFE8H, 1)

Content: Copy of the VDP register R#9.

**R10SAV** (FFE9H, 1)

Content: Copy of the VDP register R#10.

**R11SAV** (FFEAH, 1)

Content: Copy of the VDP register R#11.

**R12SAV** (FFEBH, 1)

Content: Copy of the VDP register R#12.

**R13SAV** (FFECH, 1)

Content: Copy of the VDP register R#13.

**R14SAV** (FFEDH, 1)

Content: Copy of the VDP register R#14.

**R15SAV** (FFEEH, 1)

Content: Copy of the VDP register R#15.

**R16SAV** (FFEFH, 1)

Content: Copy of the VDP register R#16.

**R17SAV** (FFF0H, 1)

Content: Copy of the VDP register R#17.

**R18SAV** (FFF1H, 1)

Content: Copy of the VDP register R#18.

**R19SAV** (FFF2H, 1)

Content: Copy of the VDP register R#19.

**R20SAV** (FFF3H, 1)

Content: Copy of the VDP register R#20.

**R21SAV** (FFF4H, 1)

Content: Copy of the VDP register R#21.

**R22SAV** (FFF5H, 1)

Content: Copy of the VDP register R#22.

**R23SAV** (FFF6H, 1)

Content: Copy of the VDP register R#23.

## 7.6.2 – Area used for the V9958

**R25SAV** (FFFAH, 1)

Content: Copy of the VDP register R#25 (V9958).

**R26SAV** (FFFBH, 1)

Content: Copy of the VDP register R#26 (V9958).

**R27SAV** (FFFCH, 1)

Content: Copy of the VDP register R#27 (V9958).

## 7.7 – MISCELLANEOUS

**TRGFLG** (F3E8H, 1)

Initial value: 11 110 001B

Content: Status of the joystick buttons. (0 = pressed, 1 = not pressed).  
This variable is constantly updated by the interrupt handler.

**FORCLR** (F3E9H, 1)

Initial value: 15

Content: Front and character color. It can be changed by the COLOR command.

**BAKCLR** (F3EAH, 1)

Initial value: 4

Content: Background color. It can be changed by the COLOR command.

**BDRCLR** (F3EBH, 1)

Initial value: 7

Content: Color of the border. It can be changed by the COLOR command.

**MAXUPD** (F3ECH, 3)

Initial value: JP 0000H (C3H, 00H, 00H)

Content: Used internally by the CIRCLE command.

**MINUPD** (F3EFH, 3)

Initial value: JP 0000H (C3H, 00H, 00H)

Content: Used internally by the CIRCLE command.

**ATRBYT** (F3F2H, 1)

Initial value: 15

Content: Color code used for graphics.

**7.8 – AREA USED BY PLAY COMMAND****QUEUES** (F3F3H, 2)

Initial value: QUETAB (F959H)

Content: Pointer to the PLAY command execution queue.

**FRCNEW** (F3F5H, 1)

Initial value: 255

Content: Used internally by the BASIC interpreter.

**MCLTAB** (F956H, 2)

Content: Address of the command table to be used by the PLAY and DRAW macro commands.

**MCLFLG** (F958H, 1)Content: Flag to indicate which command is being processed  
(0 = DRAW; not zero, PLAY).**QUETAB** (F959H, 24)

Content: This table contains the data for the three musical queues and RS232C queue, with six bytes reserved for each one.

+0: position to place

+1: position to get

+2: return indication

+3: size of the buffer in the queue

+4: address of the buffer in the queue (high)

+5: address of the buffer in the queue (low)

F959H = voz A  
 F95FH = voz B  
 F965H = voz C  
 F96AH = RS232C

**QUEBAK** (F971H, 4)

Content: Used for replacement characters table of queues

F971H +0 – Voice A  
 +1 – Voice B  
 +2 – Voice C  
 +3 – RS232C (MSX1 only)

**VOICAQ** (F975H, 128)

Initial value: DEFS 128 (00H ..... 00H)

Content: Queue for voice A.

**VOICBQ** (F9F5H, 128)

Initial value: DEFS 128 (00H ..... 00H)

Content: Queue for voice B.

**VOICCQ** (FA75H, 128)

Initial value: DEFS 128 (00H ..... 00H)

Content: Queue for voice C.

**7.24 – AREA USED BY THE PLAY COMMAND****PRSCNT** (FB35H, 1)

Content: Used internally by the PLAY command to count the number of operands completed. Bit 7 will be turned on after each of the three operands is analyzed.

**SAVSP** (FB36H, 2)

Content: Saves the value of the SP register before executing the PLAY command.

**VOICEN** (FB38H, 1)

Content: Number of the voice currently being processed (0, 1 or 2).

**SAVVOL** (FB39H, 2)

Content: Saves the volume when generating a pause.

**MCLLEN** (FB3BH, 1)

Content: Length of the string being analyzed.

**MCLPTR** (FB3CH, 2)

Content: Address of the operand being analyzed.

**QUEUEN** (FB3EH, 1)

Content: Used by the interrupt handler to contain the number of the musical queue that is currently being processed.

**MUSICF** (FB3FH, 1)

Content: Flag to indicate which musical queues will be used.

**PLYCNT** (FB40H, 1)

Content: Number of strings stored in the PLAY command queue.

**7.8.1 – Offset for PLAY buffer parameter control**

|        |           |                                     |
|--------|-----------|-------------------------------------|
| METREX | (+00, 2)  | Duration counter                    |
| VCXLEN | (+02, 1)  | String length                       |
| VCXPTR | (+03, 2)  | String address                      |
| VCXSTP | (+05, 2)  | Data address on the stack           |
| QLENGX | (+07, 1)  | Size of the musical packet in bytes |
| NTICSX | (+08, 2)  | Music package                       |
| TONPRX | (+10, 2)  | Tone period                         |
| AMPRX  | (+12, 1)  | Volume and envelope                 |
| ENVPRX | (+13, 2)  | Envelope period                     |
| OCTAVX | (+15, 1)  | Octave                              |
| NOTELX | (+16, 1)  | Tone length                         |
| TEMPOX | (+17, 1)  | Time                                |
| VOLUMX | (+18, 1)  | Volume                              |
| ENVLPX | (+19, 14) | Envelope waveform                   |
| MCLSTX | (+33, 3)  | Reserved for the battery            |
| MCLSEX | (+36, 1)  | Initialization of the stack         |
| VCBSIZ | (+37, 1)  | Size of the parameter buffer        |

## 7.8.2 – Data area for the parameter buffer

### **VCBA** (FB41H, 37)

Content: Parameters for voice A.

- +00, 2 – Duration counter
- +02, 1 – String length
- +03, 2 – String address
- +05, 2 – Data address on the stack
- +07, 1 – Music package size
- +08, 7 – Music package
- +15, 1 – Eighth
- +16, 1 – Length
- +17, 1 – Weather
- +18, 1 – Volume
- +19, 2 – Wrap period
- +21,16 – Stack data space

### **VCBB** (FB66H, 37)

Content: Parameters for voice B.

(Structure identical to voice A).

### **VCBC** (FB8BH, 37)

Content: Parameters for the C voice.

(Structure identical to voice A).

## 7.9 – KEYBOARD AREA

### **SCNCNT** (F3F6H, 1)

Initial value: 1

Content: Interval for scanning the keys.

### **REPCNT** (F3F7H, 1)

Initial value: 50

Content: Delay time for the start of the auto-repeat of the keys.

### **PUTPNT** (F3F8H, 2)

Initial value: KEYBUF (FBF0H)

Content: Points to the write address of the keyboard buffer.



**GETPNT** (F3FAH, 2)

Initial value: KEYBUF (FBF0H)

Content: Points to the reading address of the keyboard buffer.

**7.10 – AREA USED BY CASSETTE****CS1200** (F3FCH, 5)

Initial value: +0 → 53H – first half for bit 0

+1 → 5CH – second half for bit 0

+2 → 26H – first half for bit 1

+3 → 2DH – second half for bit 1

+4 → 0FH – cycle count for short header

[cycles = (0F400H) \* 2/256]

Content: Parameters for the 1200 baud cassette.

**CS2400** (F401H, 5)

Initial value: +0 → 25H – first half for bit 0

+1 → 2DH – second half for bit 0

+2 → 0EH – first half for bit 1

+3 → 16H – second half for bit 1

+4 → 1FH – count of cycles for short header

[cycles = (0F405H) \* 4/256]

Content: Parameters for the cassette for 2400 baud.

**LOW** (F406H, 2)

Initial value: +0 → 53H – first half for bit 0

+1 → 5CH – second half for bit 0

Content: Width for bit 0 of the current baud rate.

**HIGH** (F408H, 2)

Initial value +0 → 26H – first half for bit 1

+1 → 2DH – second half for bit 1

Content: Width for bit 1 of the current baud rate.

**HEADER** (F40AH, 1)

Initial value: 0FH

Content: Count of cycles for current short header.

## 7.11 – AREA USED BY CIRCLE COMMAND

### **ASPCT1** (F40BH, 2)

Content: 256 / aspect ratio. Can be changed by SCREEN command for use of the CIRCLE command.

### **ASPCT2** (F40DH, 2)

Content: 256 \* aspect ratio. Can be changed by SCREEN command for use of the CIRCLE command.

### **ASPECT** (F931H, 2)

Initial value: 0

Content: Aspect ratio.

### **CENCNT** (F933H, 2)

Initial value: 0

Content: Count of points of the final angle.

### **CLINEF** (F935H, 1)

Initial value: 0

Content: Flag used to indicate the drawing of a line from the center of the circle. Bit 0 will be turned on if a line is required from the start angle and bit 7 will be turned on if the line is required from the end angle.

### **CNPNTS** (F936H, 2)

Initial value: 0

Content: Number of points within a 45 degree segment of the circumference.

### **CPLOT** (F938H, 1)

Content: Used internally by the CIRCLE command.

### **CPCNT** (F939H, 2)

Content: Y coordinate within the current 45 degree segment of the circumference.

### **CPCNT8** (F93BH, 2)

Initial value: 0

Content: Total point count of the current position.

**CPCSUM** (F93DH, 2)

Initial value: 0

Content: Counter of points for computation.

**CSTCNT** (F93FH, 2)

Initial value: 0

Content: Count of points of the initial angle of the circumference.

**CSCLXY** (F941H, 1)

Initial value: 0

Content: Scales between X and Y. Used by the CIRCLE command.

**CSAVEA** (F942H, 2)

Content: Area reserved for ADVGRP.

**CSAVEM** (F944H, 1)

Content: Area reserved for ADVGRP.

**CXOFF** (F945H, 2)

Content: X coordinate from the center of the circle.

**CYOFF** (F947H, 2)

Content: Y coordinate from the center of the circle.

**7.12 – AREA INTERNALLY USED BY BASIC****ENDPRG** (F40FH, 5)

Initial value: ":", 00H; 00H; 00H; 00H.

Content: False end of program line for RESUME and NEXT commands.

**ERRFLG (F414H, 1)**

Initial value: 00H

Content: Area to save the error number.

**LPTPOS** (F415H, 1)

Initial value: 00H

Content: Stores the current position of the printer head.

**PRTFLG** (F416H, 1)

Initial value: 00H

Content: Flag to select output for screen or printer (0 = Screen; other value, printer).

**NTMSXP** (F417H, 1)

Initial value: 00H

Content: Flag to select the type of printer. (0 = Standard MSX printer; other value, non-MSX printer). It can be changed by the SCREEN command.

**RAWPRT** (F418H, 1)

Initial value: 00H

Content: Flag to determine whether the graphic control characters will be modified when sent to the printer (0 = modify; another value, does not modify).

**VLZADR** (F419H, 2)

Initial value: 0000H

Content: Character address for the VAL function.

**VLZDAT** (F41BH, 1)

Initial value: 00H

Content: Character that should be replaced by 0 with VAL function.

**CURLIN** (F41CH, 2)

Initial value: FFFFH

Content: Current line number of the BASIC interpreter. The FFFFH value indicates direct mode.

**7.12.1 – BASIC text buffers****KBFMIN** (F41EH, 1)

Initial value: ":"

Content: This byte is a fictitious prefix for the tokenized text contained in KBUF.

**KBUF** (F41FH, 318)

Initial value: 00H, 00H,... 00H

Content: This buffer stores the collected tokenized BASIC line by the interpreter.

**BUFMIN** (F55DH, 1)

Initial value: ":"

Content: Fictitious prefix for the text contained in BUF. It is used to synchronize the INPUT instruction handler when it starts analyzing the collected text.

**BUF** (F55EH, 258)

Initial value: 00H, 00H,... 00H

Content: This buffer stores, in ASCII format, the characters collected from the keyboard by the standard INLIN routine.

**ENDBUF** (F660H, 1)

Initial value: 00H

Content: Byte to prevent BUF overflow (F55EH).

**7.12.2 – General data****TTYPOS** (F661H, 1)

Content: Used by the PRINT command to store the virtual cursor position.

**DIMFLG** (F662H, 1)

Content: Used internally by the DIM command.

**VALTYP** (F663H, 1)

Content: Stores the type of variable contained in DAC (F3F6H):  
2 = Integer; 3 = String; 4 = Simple precision; 8 = Double prec.

**DORES** (F664H, 1)

Content: Used internally by the DATA command to keep the text in ASCII format.

**DONUM** (F665H, 1)

Content: Flag used internally by BASIC.

**CONTXT** (F666H, 2)

Content: Stores the address of the text used by the CHRGTTR routine.

**CONSAV** (F668H, 1)

Content: Stores the token of a numeric constant; used by the GHRGTTR routine.

**CONTYP** (F669H, 1)

Content: Stores the type of a numeric constant found in the BASIC program text. It is used by the prairie CHRGTTR routine.

**CONLO** (F66AH, 8)

Content: Stores a numeric constant used by standard routine CHRGTTR.

**MEMSIZ** (F672H, 2)

Content: Highest memory address that can be used by BASIC.

**STKTOP** (F674H, 2)

Content: Address of the top of the Z80 stack. Used internally by BASIC.

**TXTTAB** (F676H, 2)

Initial value: 8000H

Content: Starting address of the BASIC text area.

**TEMPPT** (F678H, 2)

Initial value: TEMPST (F67AH)

Content: Address of the next free position in TEMPST.

**TEMPST** (F67AH, 30)

Initial value: DEFS 30 (00H, ..... 00H)

Content: Buffer used to store string descriptors.

**DSCTMP** (F698H, 3)

Initial value: 00H, 00H, 00H

Content: Saves the string descriptor during processing.

**FRETOP** (F69BH, 2)

Initial value: F168H

Content: Address of the next free position in the string area.

**TEMP3** (F69DH, 2)

Initial value: 0000H

Content: Used internally by the interpreter for temporary storage of multiple routines.

**ENDFOR** (F6A1H, 2)

Initial value: 0000H

Content: Address for the FOR command.

**DATLIN** (F6A3H, 2)

Initial value: 00H

Content: Line number of the DATA command to use the READ command.

**SUBFLG** (F6A5H, 1)

Initial value: 00H

Content: Flag used to control the processing of indexes when searching for matrix type variables.

**FLGINP** (F6A6H, 1)

Initial value: 00H

Content: Flag used by the INPUT and READ commands (0 = INPUT; another value, READ).

**TEMP** (F6A7H, 2)

Content: Used internally by the interpreter.

**7.12.3 – BASIC lines control at runtime****PTRFLG** (F6A9H, 1)

Content: Used internally by the interpreter to convert line numbers to pointers (0 = Operand not converted; another value, operand converted).

**AUTFLG** (F6AAH, 1)

Content: Flag for the AUTO command (0 = AUTO command inactive; another value, AUTO command active).

**AUTLIN** (F6ABH, 2)

Content: Number of the last BASIC line entered.

**AUTINC** (F6ADH, 2)

Initial value: 10

Content: Increment value for the AUTO function.

**SAVTEXT** (F6AFH, 2)

Content: Stores the current BASIC text address during the execution.

**SAVSTK** (F6B1H, 2)

Content: Stores the current address of the Z80 stack. Used by the error handler and the RESUME statement.

**ERRLIN** (F6B3H, 2)

Content: BASIC line number where an error occurred.

**DOT** (F6B5H, 2)

Content: Last line number during processing. Used internally by the interpreter and the error handler.

**ERRTXT** (F6B7H, 2)

Content: Address of the BASIC text where an error occurred. Used by the RESUME command.

**ONELIN** (F6B9H, 2)

Content: Address of the program line that must be executed when an error occurs. Set by the ON ERROR GOTO command.

**ONEFLG** (F6BBH, 1)

Content: Flag to indicate the execution of an error routine (0 = not executing; another value, routine in execution).

**TEMP2** (F6BCH, 2)

Content: Used internally by the interpreter.

**OLDLIN** (F6BEH, 2)

Content: Stores the last line executed by the program. It is updated by the END and STOP commands to be used by the CONT command.

#### 7.12.4 – BASIC text storage addresses

**OLDTXT** (F6C0H, 2)

Content: Stores the address of the last instruction in the BASIC text.

**VARTAB** (F6C2H, 2)

Content: Address of the first byte of the storage area of the BASIC variables.



**ARYTAB** (F6C4H, 2)

Content: Address of the first byte of the storage area of the BASIC arrays.

**STREND** (F6C6H, 2)

Content: Address of the first byte after the storage area for arrays, variables or BASIC text.

**DATPTR** (F6C8H, 2)

Content: Address of the current DATA command to use the READ command.

**DEFTBL** (F6CAH, 26)

Content: Storage area of the variable type by names in alphabetical order. They can be changed by the command group "DEF xxx".

**7.12.5 – Area for user functions****PRMSTK** (F6E4H, 2)

Content: Previous definition of the block in the Z80 stack.

**PRMLEN** (F6E6H, 2)

Content: Length of the current "FN" parameter block in PARM1.

**PARM1** (F6E8H, 100)

Content: Buffer for storing the variables of the "FN" function being evaluated.

**PRMPRV** (F74CH, 2)

Initial value: PRMSTK (F6E4H)

Content: Address of the previous "FN" parameter block.

**PRMLN2** (F74EH, 2)

Content: Length of the "FN" parameter block being assembled in PARM2.

**PARM2** (F750H, 100)

Content: Buffer used for the variables of the current "FN" function.

**PRMFLG** (F7B4H, 1)

Content: Flag to indicate when PARM1 is being searched.

**ARYTA2** (F7B5H, 2)

Content: Last address to search for a variable.

**NOFUNS** (F7B7H, 1)

Content: Flag to indicate to the "FN" function the existence of local variables (0 = there are no variables; another value, there are variables).

**TEMP9** (F7B8H, 2)

Content: Used internally by the interpreter.

**FUNACT** (F7BAH, 2)

Content: Number of "FN" functions currently active.

**SWPTMP** (F7BCH, 8)

Content: Buffer used to contain the first operand of a SWAP command.

**TRCFLG** (F7C4H, 1)

Content: Flag for the TRACE command (0 = TRACE OFF, another value, TRACE ON).

### 7.12.6 – Interpreter data area

**FNKSTR** (F87FH, 160)

Content: Reserved area to store the content of the function keys (10 positions with 16 characters each).

**CGPNT** (F91FH, 3)

Content: Address of the character font. The first byte is the slot ID and the other two is the address.

**NAMBAS** (F922H, 2)

Content: Address of the name table in the current text mode.

**CGPBAS** (F924H, 2)

Content: Address of the pattern generator table in the current text mode.

**PATBAS** (F926H, 2)

Content: Current address of the sprite generator table.

**ATRBAS** (F928H, 2)

Content: Current address of the sprites attribute table.

**CLOC** (F92AH, 2)

Content: Used internally by graphic routines.

**CMASK** (F92CH, 1)

Content: Used internally by graphic routines.

**MINDEL** (F92DH, 2)

Content: Used internally by the LINE command.

**MAXDEL** (F92FH, 2)

Content: Used internally by the LINE command.

### 7.13 – MATH-PACK AREA

**FBUFFR** (F7C5H, 43)

Content: Used internally by MATH-PACK.

**DECTMP** (F7F0H, 2)

Content: Used to transform an integer into a floating point number.

**DECTM2** (F7F2H, 2)

Content: Used internally by the division routine.

**DECCNT** (F7F4H, 1)

Content: Used internally by the division routine.

**DAC** (F7F6H, 16)

Content: Primary accumulator that contains a number during a mathematical operation.

**HOLD8** (F806H, 48)

Initial value: 00H, 00H... 00H

Content: Storage area for decimal multiplication.

**HOLD2** (F836H, 8)

Initial value: 00H, 00H... 00H

Content: Used internally by MATH-PACK.

**HOLD** (F83EH, 8)

Initial value: 00H, 00H... 00H

Content: Used internally by MATH-PACK.

**ARG** (F847H, 16)

Content: Secondary accumulator that contains the number to be calculated with DAC (F7F6H).

**RNDX** (F857H, 8)

Content: Stores the last double-precision random number. Used by the RND function.

**7.14 – DISK SYSTEM DATA AREA****MAXFIL** (F85FH, 1)

Content: Number of existing I/O buffers. It can be changed by the MAXFILES statement.

**FILTAB** (F860H, 2)

Content: Initial address of the data area of the files.

**NULBUF** (F862H, 2)

Content: Points to the buffer used by commands SAVE and LOAD.

**PTRFIL** (F864H, 2)

Content: Address of the data of the currently active file.

**RUNFLG** (F866H, 0)

Content: Non-zero, if any programs have been loaded and executed. Used by the ", R" operand of the LOAD command.

**FILNAM** (F866H, 11)

Content: Area for storing a filename.

**FILNM2** (F871H, 11)

Content: Area for storing a filename to be compared with FILNAM.

**NLONLY** (F87CH, 1)

Content: Flag to indicate whether a program is being loaded or not (0 = program is not being loaded; another value, program is being loaded).

**SAVEND** (F87DH, 2)

Content: Used by the BSAVE command to contain the final address of the assembly program to be saved.

**HOKVLD** (FB20H, 1)

Initial value: 01H

Content: Bit 0 of this byte indicates the presence of an extended BIOS. (0 = No Extended BIOS, 1 = There is at least one BIOS that can be called at address 0FFCAh (EXTBIO)).

**DRVINV** (FB21H, 9)

Initial value: variable

Content: Slot ID and num of drives connected to the disk interfaces.

DRVINV +0 = Number of drives connected to the primary disk interface.

+1 = Master disk interface slot ID.

+2 = Number of units connected to the master disk interface.

+3 = 2nd disk interface slot ID.

+4 = Number of units connected to the 2nd disk interface.

+5 = 3rd disk interface slot ID.

+6 = Number of units connected to the 3rd disk interface.

+7 = Slot ID of the 4th disk interface.

+8 = Number of units connected to the 4th disk interface.

**DRVINT** (FB29H, 12)

Initial value: variable

Content: Slot ID and address of each disk interface interrupt handler (3 \* 4 bytes).

DRVINT +0 = slot ID of each interrupt handler on the main interface.

- +1, +2 = Address of the interrupt handler of the main interface.
- +3 = Slot ID of each interrupt handler on the 2nd interface.
- +4, +5 = 2nd interface interrupt handler address.
- +6 = Slot ID of each interrupt handler on the 3rd interface.
- +7, +8 = 3rd interface interrupt handler address.
- +9 = Slot ID of each interrupt handler on the 4rd interface.
- +10 = 4th interface interrupt handler address.

## 7.15 – AREA USED BY PAINT COMMAND

### **LOHMSK** (F949H, 1)

Initial value: 0

Content: Leftmost position of the LH tour.

### **LOHDIR** (F94AH, 1)

Initial value: 0

Content: Painting direction required by the LH tour.

### **LOHADR** (F94BH, 2)

Initial value: 0000H

Content: Leftmost position of the LH tour.

### **LOHCNT** (F94DH, 2)

Initial value: 0

Content: Size of the LH tour.

### **SKPCNT** (F94FH, 2)

Initial value: 0

Content: Hop counter returned by SCANR (012CH).

### **MOVCNT** (F951H, 2)

Initial value: 0

Content: Movement counter returned by SCANR (012CH).

### **PDIREC** (F953H, 1)

Content: Painting direction: 40H, down; C0H, upward; 00H, finish.

**LFPROG** (F954H, 1)

Content: Flag used by the PAINT command to indicate whether there was progress to the left (0 = there was no progress; another value, there was progress).

**RTPROG** (F955H, 1)

Content: Flag used by the PAINT command to indicate whether there has been progress on the right (0 = there was no progress; another value, there was progress).

**7.16 – ADDED AREA FOR MSX2****DPPAGE** (FAF5H, 1)

Initial value: 0

Content: Video page that is currently being displayed.

**ACPAGE** (FAF6H, 1)

Initial value: 0

Content: Active page for receiving commands.

**AVCSAV** (FAF7H, 1)

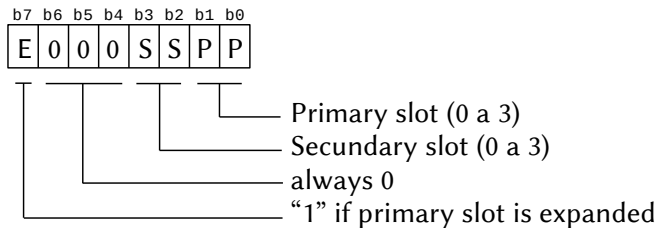
Initial value: 0

Content: Used by the AV control port.

**EXBRSA** (FAF8H, 1)

Initial value: 10 000 111B

Content: Sub-ROM slot, in the format below:

**CHRCNT** (FAF9H, 1)

Initial value: 0

Content: Character counter in the buffer. Used for Roman-Kana transition (0, 1 or 2).

**ROME** (FAFAH, 2)

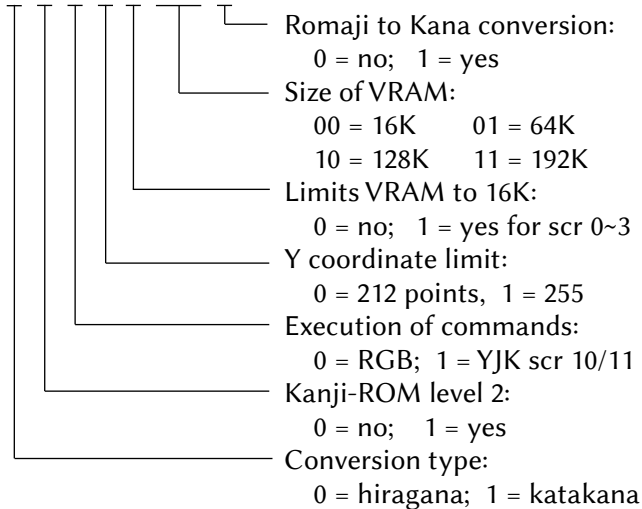
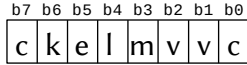
Initial value: 0

Content: Stores the character of the buffer for the Roman-Kana transition (Japanese version only).

**MODE** (FAFCH, 1)

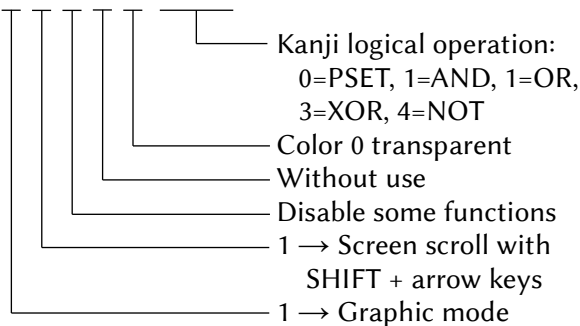
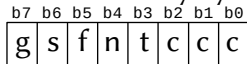
Initial value: 1000 1001B

Content: VRAM size and mode flag:

**NORUSE** (FAFDH, 1)

Initial value: 00H

Content: Used internally by the Kanji-driver.





**XSAVE** (FAFEH, 2)

Initial value: 00 000 000B, 00 000 000B

| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| L  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | X  | X  | X  | X  | X  | X  | X  | X  |

**YSAVE** (FB00H, 2)

Initial value: 00 000 000B, 00 000 000B

| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| L  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | Y  | Y  | Y  | Y  | Y  | Y  | Y  | Y  |

L = 1 → lightpen interruption request

0 000 000 = meaningless

XXXXXXXXXX = X coordinate

YYYYYYYYYY = Y coordinate

**LOGOPR** (FB02H, 1)

Initial value: 00H

Content: Logical operation code for the VDP.

**7.17 – AREA USED BY RS232C****RSTMP** (FB03H, 1)

Initial value: 00H

Content: Temporary storage for the RS232C.

Note: Same address as TOCNT.

**TOCNT** (FB03H, 1)

Initial value: 00H

Content: Counter used by the RS232C interface.

Note: Same address as RSTMP.

**RSFCB** (FB04H, 2)

Initial value: 0000H

Content: FCB address of RS232C.

**RSIQLN** (FB06H, 1)

Initial value: 00H

Content: Used internally by the RS232C.

**MEXBIH** (FB07H, 5)

Initial value: C9H, C9H, C9H, C9H, C9H

Content: Used internally by the RS232C.

FB07H +0: RST 030H

+1: Slot ID byte

+2: Address (low)

+3: Address (high)

+4: RET

**OLDSTT** (FB0CH, 5)

Initial value: C9H, C9H, C9H, C9H, C9H

Content: Used internally by the RS232C.

FB0CH +0: RST 030H

+1: Slot ID byte

+2: Address (low)

+3: Address (high)

+4: RET

**OLDINT** (FB12H, 5)

Initial value: C9H, C9H, C9H, C9H, C9H

Content: Used internally by the RS232C.

FB12H +0: RST 030H

+1: Slot ID byte

+2: Address (low)

+3: Address (high)

+4: RET

**DEVNUM** (FB17H, 1)

Content: Offset byte.

**DATCNT** (FB18H, 3)

Content: FB18H+0: Slot ID

+1: Pointer

+2: Pointer

**ERRORS** (FB1BH, 1)

Initial value: 00H

Content: RS232C error code.

**FLAGS** (FB1CH, 1)

Initial value: 00000011B

Content: Flags used by the RS232C.

**ESTBLS** (FB1DH, 1)

Initial value: FFH

Content: Boolean bit for use with RS232C.

**COMMSK** (FB1EH, 1)

Initial value: C1H

Content: RS232C mask.

**LSTCOM** (FB1FH, 1)

Initial value: E8H

Content: Used internally by the RS232C.

**7.18 – GENERAL DATA AREA****ENSTOP** (FBB0H, 1)

Content: Flag to enable a forced output for the interpreter when detecting the CTRL + SHIFT + GRAPH + CODE keys pressed together (0 = Disabled; another value, enabled).

**BASROM** (FBB1H, 1)

Initial value: 00H

Content: Location of BASIC text (0 = RAM; other value, ROM).

**LINTTB** (FBB2H, 24)

Content: There are 24 flags to indicate whether each line of text screen has advanced to the next line (0 = Advanced; another value, has not advanced).

**FSTPOS** (FBCAH, 2)

Content: First location of the character collected by the BIOS INLIN (00B1H) routine.

**CODSAV** (FBCCH, 1)

Initial value: 00H

Content: Character replaced by the cursor in the text screens.

**FNKSW1** (FBCDH, 1)

Initial value: 01H

Content: Flag to indicate which function keys are shown when enabled by KEY ON (1 = F1 to F5; 0 = F6 to F10).

**FNKFLG** (FBCEH, 10)

Content: Flags to enable, inhibit or stop the execution of a line defined by the ON KEY GOSUB command. They are modified by KEY (n) ON / OFF / STOP (0 = KEY (n) OFF / STOP; 1 = KEY (n) ON).

**ONGSBF** (FBD8H, 1)

Content: Flag to indicate whether any device required a program interruption (0 = normal; another value = Active interrupt).

**CLIKFL** (FBD9H, 1)

Content: Click flag of the keys. Used by the interrupt handler.

**OLDKEY** (FBDAH, 11)

Content: Previous state of the keyboard matrix.

**NEWKEY** (FBE5H, 11)

Content: New state of the keyboard matrix.

**KEYBUF** (FBF0H, 40)

Content: Circular buffer containing the decoded keyboard chars.

**LINWRK** (FC18H, 40)

Content: Buffer used by the BIOS to contain a full line of characters on the screen.

**PATWRK** (FC40H, 8)

Content: Buffer used by the BIOS to contain an 8x8 char pattern.

**BOTTOM** (FC48H, 2)

Content: Lowest address used by the interpreter, usually 8000H.

**HIMEM** (FC4AH, 2)

Content: Highest available RAM address for BASIC interpreter. It can be modified by the CLEAR command.

**TRPTBL** (FC4CH, 78)

Content: This table contains the current state of the interrupting devices. Each device allocates three bytes in the table. The first byte contains the state of the device (bit 0 = On; bit 1 = stopped; bit 2 = Active). The other two bytes contain the address of the program line to be executed in the interruption event.

|               |                        |                   |
|---------------|------------------------|-------------------|
| FC4CH / FC69H | (3 x 10 bytes)         | ON KEY GOSUB      |
| FC6AH / FC6CH | (3 x 1 byte)           | ON STOP GOSUB     |
| FC6DH / FC6FH | (3 x 1 byte)           | ON SPRITE GOSUB   |
| FC70H / FC7EH | (3 x 5 bytes)          | ON STRIG GOSUB    |
| FC7FH / FC81H | (3 x 1 byte)           | ON INTERVAL GOSUB |
| FC82H / FC99H | Reserved for expansion |                   |

**RTYCNT** (FC9AH, 1)

Content: Used internally by BASIC.

**INTFLG** (FC9BH, 1)

Content: If CTRL+STOP are pressed, this variable is set to 03H and processing is interrupted; if STOP is pressed, the value is 04H.

**PADY** (FC9CH, 1)

Content: Y coordinate of the paddle.

**PADX** (FC9DH, 1)

Content: X coordinate of the paddle.

**JIFFY** (FC9EH, 2)

Content: This variable is continuously incremented by the interrupt handler. Its value can be read or assigned by the TIME function. It is also used internally by the PLAY command.

**INTVAL** (FCA0H, 2)

Initial value: 0000H

Content: Duration of the interval used by ON INTERVAL GOSUB.

**INTCNT** (FCA2H, 2)

Initial value: 0000H

Content: Counter for the ON INTERVAL GOSUB instruction.

**LOWLIM** (FCA4H, 1)

Initial value: 31H

Content: Minimum duration for the starting bit when reading the cassette.

**WINWID** (FCA5H, 1)

Initial value: 22H

Content: Duration of the discrimination of the high / low cycle during the reading of the cassette.

**GRPHED** (FCA6H, 1)Content: Flag for sending a graphic character.  
(0 = normal; 1 = graphic character).**ESCCNT** (FCA7H, 1)

Content: Escape codes counting area.

**INSFLG** (FCA8H, 1)

Content: Flag to indicate the insertion mode (0 = normal; another value, insertion mode)

**CSRSW** (FCA9H, 1)

Content: Flag to indicate whether the cursor will be shown (0 = no; another value, yes). Can be modified by the LOCATE command.

**CSTYLE** (FCAAH, 1)

Content: Cursor shape (0 = block; other value, sub-aligned).

**CAPST** (FCABH, 1)

Content: Status of the CAPS LOCK key (0 = Off; another value, on).

**KANAST** (FCACH, 1)

Content: Status of the KANA key (0 = Off; another value, on).

**KANAMD** (FCADH, 1)

Content: Flag used only on Japanese machines.

**FLBMEM** (FCAEH, 1)

Content: Flag to indicate program loading in BASIC (0 = Is loading; another value, is not).

**SCRMOD** (FCAFH, 1)

Content: Number of the current screen mode.

**OLDSCR** (FCB0H, 1)

Content: Screen mode of the last text mode.

**CASPRV** (FCB1H, 1)

Initial value: 00H

Content: Used by the cassette on MSX1, MSX2 and MSX2+. In the MSX turbo R, it stores the A7H port value.

**BDRATR** (FCB2H, 1)

Content: Color code of the border. Used by PAINT.

**GXPOS** (FCB3H, 2)

Content: Graphic X coordinate.

**GYPOS** (FCB5H, 2)

Content: Graphic Y coordinate.

**GRPACX** (FCB7H, 2)

Content: Graphic accumulator for the X coordinate.

**GRPACY** (FCB9H, 2)

Content: Graphic accumulator for the Y coordinate.

**DRWFLG** (FCBBH, 1)

Content: Flag used by the DRAW command.

**DRWSCL** (FCBCH, 1)

Content: Scale factor for the DRAW command. A value of 0 indicates that the scale will not be used.

**DRWANG** (FCBDH, 1)

Content: Angle for the DRAW command.

**RUNBNF** (FCBEH, 1)

Content: Flag to indicate whether the BLOAD or BSAVE command is running (disk system only).

**SAVENT** (FCBFH, 2)

Initial value: 0000H

Content: Initial address for the BSAVE and BLOAD commands (disk system only)

**7.19 – BIOS EXPANSION ROUTINES****EXTBIO** (FFCAH)

Purpose: To directly expand the system BIOS.

Input: A – Always 0.

D – device identifier (device number 0 is used to get installed extensions).

E – function to be called.

Note: Refer to the “EXTENDED BIOS ROUTINES” section for more details.

**DISINT** (FFCFH)

Purpose: Called by function 2 of the “broadcast”.

Input: None.

**ENAIN** (FFD4H)

Purpose: Called by function 2 of the “broadcast”.

Input: None.

**FFD9H~FFE6H** → Contains the code for the DISINT and ENAIN routines.**7.20 – DATA AREA FOR SLOTS AND PAGES****EXPTBL** (FCC1H, 4)

Initial value: Variable.

Content: Table of flags to indicate whether the primary slots are expanded:

FCC1H → primary slot 0 (Main-ROM slot).

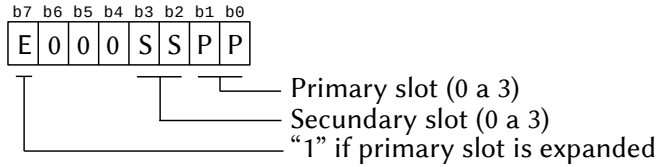
FCC2H → primary slot 0 (Main-ROM slot).

FCC3H → primary slot 0 (Main-ROM slot).

FCC4H → primary slot 0 (Main-ROM slot).

The structure of each flag is described below:





### SLTTBL (FCC5H, 4)

Content: These four bytes contain the possible state of the four primary slot registers, in case the slot is expanded.

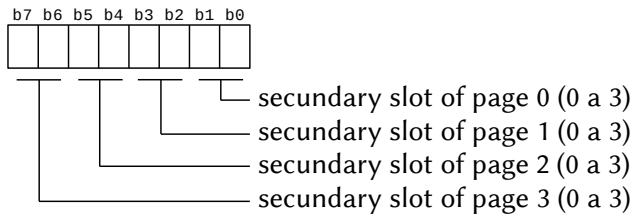
FCC5H → status for primary slot 0

FCC6H → status for primary slot 1

FCC7H → status for primary slot 2

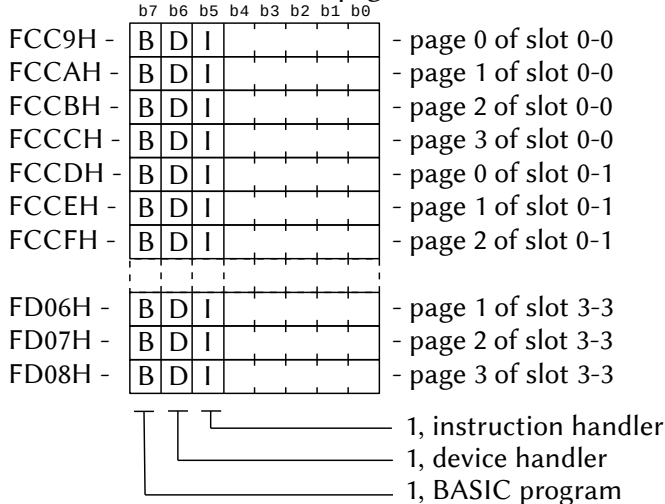
FCC8H → status for primary slot 3

The structure of each flag is described below:



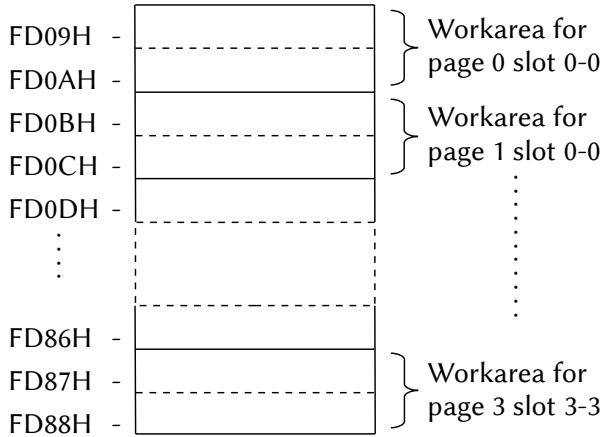
### SLTATR (FCC9H, 64)

Content: Table of attributes for each page of each slot.



**SLTWRK** (FD09H, 128)

Content: This table allocates two bytes as a working area for each page of each slot.

**PROCNM** (FD89H, 16)

Content: Stores the name of an expanded instruction (CALL command) or device expansion (OPEN command). The end of the name is marked with a byte 0.

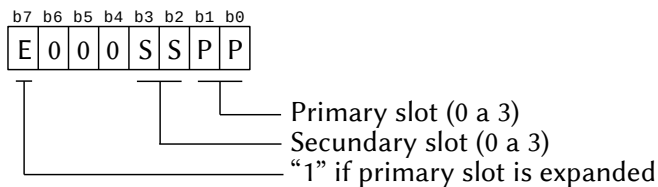
**DEVICE** (FD99H, 1)

Content: Stores the device ID in a cartridge (0 to 3).

**FD9AH~FFC9H** → Hook area (listed further ahead)

**7.20.1 – Main-ROM slot****MINROM** (FFF7H, 1)

Content: Main-ROM slot, in the format below:



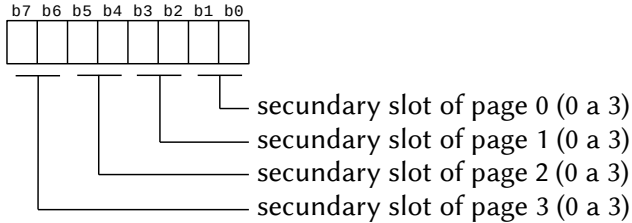
FFF8H~FFF9H → Not used

FFFDH~FFFEH → Not used

## 7.20.2 – Secondary slot register

**SLTSL** (FFFFH, 1)

Content: Secondary slot register, in the format below:



## 7.21 – HOOKS DESCRIPTION

**HKEYI** (FD9AH)

Called by: Beginning of the interrupt handler (KEYINT, 0038H).

Purpose: Add interrupt handling routines. It can also be used to test when the interruption is caused by a device other than VDP.

**HTIMI** (FD9FH)

Called by: The interrupt routine (KEYINT, 0038H) immediately after reading the VDP status register 0.

Purpose: Add interrupt handling routines. It can also be used to synchronize the graphical display, adding graphics during Vblank.

**HCHPU** (FDA4H)

Called by: Beginning of the CHPUT routine (00A2H).

Purpose: Connect other console devices besides the screen. Reg. A contains the character code when this hook is called.

**HDSPC** (FDA9H)

Called by: Beginning of the DSPSCR routine (with cursor).

Purpose: Connect other console devices besides the screen.

**HERAC** (FDAEH)

Called by: Beginning of the ERASCR routine (deletes cursor).

Purpose: Connect other console devices besides the screen.

**HDSPF** (FDB3H)

Called by: Beginning of the DSPFNK routine (features function keys).

Purpose: Connect other console devices besides the screen.

**HERAF** (FDB8H)

Called by: Beginning of routine ERAFNK (clears function keys).

Purpose: Connect other console devices besides the screen.

**HTOTE** (FDBDH)

Called by: Beginning of TOTEXT routine (forces screen to text mode).

Purpose: Connect other console devices besides the screen.

**HCHGE** (FDC2H)

Called by: Beginning of the CHGET routine (takes a character).

Purpose: Connect other console devices besides the keyboard.

**HINIP** (FDC7H)

Called by: Beginning of the INIPAT routine (initialization of the character patterns).

Purpose: To use another character table.

**HKEYC** (FDCCH)

Called by: Beginning of the routine KEYCOD (keyboard character decoder).

Purpose: To change the keyboard configuration. When this hook is called, register A contains: (line number)  $\times$  8 + column number of the key pressed in the keyboard matrix.

**HKEYA** (FDD1H)

Called by: Beginning of MSXIO NMI (KEY EASY)

Purpose: To change the way a key is interpreted.

**HNMI** (FDD6H)

Called by: Beginning of the non-masking interrupt handler (NMI, 0066H).

Purpose: NMI is disabled on a standard MSX; therefore, this hook has no use.

**HPINL** (FDDBH)

Called by: Beginning of PINLIN routine (get a line)

Purpose: To use other input devices and / or methods, such as 80 columns of text or other input devices in addition to the keyboard.

**HQINL** (FDE0H)

Called by: Beginning QINLIN routine (get a line with "?").

Purpose: To use other input devices and / or methods, such as 80 columns of text or other input devices in addition to the keyboard.

**HINLI** (FDE5H)

Called by: Beginning of the INLIN routine.

Purpose: To use other input devices and / or methods, such as 80 columns of text or other input devices in addition to the keyboard.

**HONGO** (FDEAH)

Called by: Beginning of the handler of the ON GOTO and ON GOSUB commands.

Purpose: To divert access to these BASIC instructions.

**HDSKO** (FDEFH)

Called by: Beginning of BASIC command "DSKO\$".

Purpose: Used by Disk-ROM to write a sector to the disk.

**HSETS** (FDF4H)

Called by: Beginning of the BASIC "SET" command.

Purpose: On MSX1, the instruction SET has no other effect than calling this hook and returning an error. On MSX2 or newer, the instructions SET SCREEN, SET ADJUST, SET TIME, etc call this hook to be treated.

**HNAME** (FDF9H)

Called by: Beginning of BASIC command "NAME".

Purpose: To connect disk devices.

**HKILL** (FDFEH)

Called by: Beginning of the BASIC "KILL" command.

Purpose: To connect disk devices.

**HIPL** (FE03H)

Called by: Beginning of BASIC command "IPL" (Initial Program Loading).

Purpose: Reserved. There is no known use for this instruction, but this hook can be used to add functions to the IPL instruction.

**HCOPY** (FE08H)

Called by: Beginning of the BASIC "COPY" command.

Purpose: To connect disk devices.

**HCMD** (FE0DH)

Called by: Beginning of BASIC command "CMD" (Expanded Commands).

Purpose: Reserved. There is no known use for this instruction, but this hook can be used to add functions to the CMD instruction.

**HDSKF** (FE12H)

Called by: Beginning of BASIC command "DSKF".

Purpose: To connect disk devices.

**HDSKI** (FE17H)

Called by: Beginning of BASIC command "DSKI\$".

Purpose: To connect disk devices.

**HATTR** (FE1CH)

Called by: Beginning of the BASIC "ATTR\$" command handler.

Purpose: To connect disk devices.

**HLSET** (FE21H)

Called by: Beginning of the BASIC "LSET" command handler.

Purpose: To connect disk devices.

**HRSET** (FE26H)

Called by: Beginning of the BASIC "RSET" command handler.

Purpose: To connect disk devices.

**HFIEL** (FE2BH)

Called by: Beginning of the FIELD command handler.

Purpose: To connect disk devices.

**HMKI\$** (FE30H)

Called by: Beginning of the MKI\$ command handler.

Purpose: To connect disk devices.

**HMKS\$** (FE35H)

Called by: Beginning of the MKS\$ command handler.

Purpose: To connect disk devices.

**HMKD\$** (FE3AH)

Called by: Beginning of the MKD\$ command handler.

Purpose: To connect disk devices.

**HCVI** (FE3FH)

Called by: Beginning of the CVI command handler.

Purpose: To connect disk devices.

**HCVS** (FE44H)

Called by: Beginning of the CVS command handler.

Purpose: To connect disk devices.

**HCVD** (FE49H)

Called by: Beginning of the CVD command handler.

Purpose: To connect disk devices.

**HGETP** (FE4EH)

Called by: Find FCB (get file pointer).

Purpose: To connect disk devices.

**HSETP** (FE53H)

Called by: Find FCB (set file pointer).

Purpose: To connect disk devices.

**HNOFO** (FE58H)

Called by: OPEN command handler (OPEN without FOR).

Purpose: To connect disk devices.

**HNULO** (FE5DH)

Called by: OPEN command handler (open unused file).

Purpose: To connect disk devices.

**HNTFL** (FE62H)

Called by: Close I/O buffer 0.

Purpose: To connect disk devices.

**HMERG** (FE67H)

Called by: Beginning of the MERGE and LOAD commands handler.

Purpose: To connect disk devices.

**HSAVE** (FE6CH)

Called by: Beginning of the SAVE command handler.

Purpose: To connect disk devices.

**HBINS** (FE71H)

Called by: Beginning of the SAVE command handler (in binary).

Purpose: To connect disk devices.

**HBINL** (FE76H)

Called by: Beginning of the LOAD command handler (in binary).

Purpose: To connect disk devices.

**HFILE** (FE7BH)

Called by: Beginning of the FILES command handler.

Purpose: To connect disk devices.

**HDGET** (FE80H)

Called by: Beginning of the GET and PUT commands handler.

Purpose: To connect disk devices.

**HFILO** (FE85H)

Called by: Sequential output handler.

Purpose: To connect disk devices.

**HINDS** (FE8AH)

Called by: Sequential input handler.

Purpose: To connect disk devices.

**HRSLF** (FE8FH)

Called by: Handler for pre-selection of the drive.

Purpose: To connect disk devices.



**HSAVD** (FE94H)

Called by: Reserve current disk (LOC and LOF commands).

Purpose: To connect disk devices.

**HLOC** (FE99H)

Called by: Beginning of the LOC function handler.

Purpose: To connect disk devices.

**HLOF** (FE9EH)

Called by: Beginning of the LOF function handler.

Purpose: To connect disk devices.

**HEOF** (FEA3H)

Called by: Beginning of the EOF function handler.

Purpose: To connect disk devices.

**HFPOS** (FEA8H)

Called by: Beginning of the FPOS function handler.

Purpose: To connect disk devices.

**HBAKU** (FEADH)

Called by: Beginning of the LINEINPUT # instruction handler.

Purpose: To connect disk devices.

**HPARD** (FEB2H)

Called by: Beginning of the routine that analyzes the device name.

Purpose: To expand or add device names.

**HNODE** (FEB7H)

Called by: Beginning of the NODEVN routine, which is called when no name was found in the device name table.

Purpose: To assign the default device name to another device.

**HPOSD** (FEBCH)

Called by: Analyze device name (SPCDEV POSDSK).

Purpose: To connect disk devices.

**HDEVN** (FEC1H)

Called by: Process device name.

Purpose: To expand logical device name.

**HGEND** (FEC6H)

Called by: Beginning of the routine that assigns the device name.

Purpose: To expand logical device name.

**HRUNC** (FECBH)

Called by: Beginning of the routine that initializes the interpreter variables for the RUN and NEW commands.

Purpose: Allows to assign new functions to the commands.

**HCLEA** (FED0H)

Called by: Initialize interpreter variables for CLEAR command.

Purpose: Allows to assign new functions to the command or prevent accidental deletion of variables.

**HLOPD** (FED5H)

Called by: Initialize interpreter variables (general).

Purpose: To use other default values for variables.

**HSTKE** (FEDAH)

Called by: Beginning of the STKERR (stack error) routine, used by the BASIC CLEAR command.

Purpose: This hook is called after checking the executable ROMs in each slot when starting MSX, just before the system starts the BASIC or DOS environment. Therefore, it allows you to automatically re-run the ROM after installing the disks.

**HISFL** (FEDFH)

Called by: Beginning of the ISFLIO routine, which tests whether the file should be written or read.

**HOUTD** (FEE4H)

Called by: Beginning of the OUTDO routine, which sends a character to the screen or to the printer.

**HCRDO** (FEE9H)

Called by: Beginning of the routine that sends CR + LF to the OUTDO routine.

Purpose: Allows you to use a printer with automatic line feed, for example.

**HDSKC** (FEEEH)

Called by: Disk attribute entry.

**HDOGR** (FEF3H)

Called by: Beginning of the internal DOGRPH routine, used by the BASIC graphical instructions (LINE, CIRCLE, etc.)

Purpose: To change or expand the graphical instructions.

**HPRGE** (FEF8H)

Called by: End of the execution of a BASIC program.

Purpose: Add a routine to be executed after the BASIC program ends.

**HERRP** (FEFDH)

Called by: Beginning of the routine of presenting error messages.

Purpose: Add or change error messages.

**HERRF** (FF02H)

Called by: End of the error message routine.

Purpose: Add routine to be executed after the error message is presented.

**HREAD** (FF07H)

Called by: "Ok" from the main loop (interpreter ready).

Purpose: Add a routine to be executed after the prompt is presented ("Ok").

**HMAIN** (FF0CH)

Called by: Beginning of the interpreter's BASIC text execution main loop.

Purpose: Add a routine to be executed whenever the BASIC interpreter is accessed.

**HDIRD** (FF11H)

Called by: Beginning of direct command execution (direct declaration).

Purpose: Add routine or prevent executions.

**HFINI** (FF16H)

Called by: Beginning of the FININT routine, which starts the interpretation of a BASIC commands.

Purpose: To change the processing of BASIC instructions.

**HFINE** (FF1BH)

Called by: End of the FININT routine, which initiates the interpretation of a BASIC statement.

**HCRUN** (FF20H)

Called by: Beginning of the CRUNCH routine (42B9H), which converts a BASIC text from ASCII form to tokenized form.

**HCRUS** (FF25H)

Called by: Beginning of the CRUSH routine (4353H), which looks for a reserved word in the alphabetical list of the ROM.

**HISRE** (FF2AH)

Called by: Beginning of the ISRESV routine (437CH), when a reserved word is found by the CRUSH routine.

**HNTFN** (FF2FH)

Called by: Beginning of the NTFN2 routine (43A4H), when a reserved word is followed by a line number.

**HNOTR** (FF34H)

Called by: Beginning of the NOTRSV routine (44EBH), when the sequence of characters examined by the CRUNCH routine is not a reserved word.

**HSNGF** (FF39H)

Called by: Beginning of the FOR command handler.

**HNEWS** (FF3EH)

Called by: Beginning of the interpreter's NEWSTT routine (4601H), which executes an tokenized BASIC text.

**HGONE** (FF43H)

Called by: Beginning of the GONE2 routine, used by the jump instructions (GOTO, THEN, etc.).

**HCHRG** (FF48H)

Called by: Beginning of the CHRGET routine (character entry via the keyboard).

Purpose: Use another keyboard.

**HRETU** (FF4DH)

Called by: Beginning of the RETURN command handler.

**HPTRF** (FF52H)

Called by: Beginning of the PRINT command handler.

**HCOMP** (FF57H)

Called by: Beginning of the internal COMPRT routine (4A94H), used by the PRINT handler.

**HFINP** (FF5CH)

Called by: Beginning of the routine that clears PRTFLG and PRTFIL to end the PRINT command.

Purpose: Add a routine to be executed after the PRINT command.

**HTRMN** (FF61H)

Called by: Beginning of the error handler of the READ and INPUT commands.

Purpose: Error processing.

**HFRME** (FF66H)

Called by: Routine FRMEVL (4C64H) – Expression Evaluator.

Purpose: Allows you to add new mathematical functions.

Input: HL = pointer to BASIC text

Output: HL = pointer to the expression found

VALTYP (F663H) = value type of expression

DAC (F7F6H) = value found

**HNTPL** (FF6BH)

Called by: Routine FRMEVL (4CA6H) – Expression Evaluator.

Purpose: Allows you to add new mathematical functions.

**HEVAL** (FF70H)

Called by: Factor Evaluator (4DD9H)

Purpose: Allows you to add new mathematical functions.

**HOKNO** (FF75H)

Called by: Beginning of the BASIC interpreter's transcendental function routine (hook removed on MSX turbo R. It was replaced by HMDIN).

Purpose: Allows you to add new mathematical functions.

**HMDIN** (FF75H)

Called by: Beginning of the MIDI interface interruption handling routine (MSX turbo R with internal MIDI only).

Purpose: Add or change functionality of the MIDI interface.

**HFING** (FF7AH)

Called by: Factor evaluator.

**HISMI** (FF7FH)

Called by: Beginning of the MID\$ command handler.

**HWIDT** (FF84H)

Called by: Beginning of the WIDTH command handler.

**HLIST** (FF89H)

Called by: Beginning of the LIST command handler.

**HBUFL** (FF8EH)

Called by: De-symbolize for LIST command (532DH).

**HFRQI** (FF93H)

Called by: Convert to integer (543FH). Hook removed on MSX turbo R. Replaced by HMDTM.

**HMDTM** (FF93H)

Called by: Beginning of the MIDI interface timer manipulation routine (MSX turbo R with internal MIDI only).

Purpose: Add or change functionality of the MIDI interface.

**HSCNE** (FF98H)

Called by: Beginning of the BASIC interpreter routine SCNEX2 (5514H) (converting a line number to a memory address and vice versa)

**HFRET** (FF9DH)

Called by: Searches for a free place to store the next descriptor of an alphanumeric variable (string).

**HPTRG** (FFA2H)

Called by: Beginning of the PTRGET routine (5EA9H) of the BASIC interpreter, which obtains the pointer of a variable.

Purpose: Use another default value for the variables.

**HPHYD** (FFA7H)

Called by: Beginning of routine PHYDIO (physical disk input-output).

Purpose: To connect disk devices.

**HFORM** (FFACH)

Called by: Beginning of the FORMAT (format disk) routine.

Purpose: To connect disk devices.

**HERRO** (FFB1H)

Called by: Beginning of the error handler.

Purpose: Error handling by application programs.

**HLPTO** (FFB6H)

Called by: Beginning of the LPTOUT routine (00A5H).

Purpose: Use other printer models.

**HLPTS** (FFBBH)

Called by: Beginning of the LPTSTT routine (00A5H).

Purpose: Use other printer models.

**HSCRE** (FFC0H)

Called by: Beginning of the SCREEN command handler.

Purpose: To expand the SCREEN command.

**HPLAY** (FFC5H)

Called by: Beginning of the PLAY command handler.

Purpose: To expand the PLAY command.

## 8 – BIOS ROUTINES

This appendix provides a description of the BIOS routines available to the user.

There are several types of BIOS routines, the ones in the Main-ROM, the ones in the Sub-ROM, the Math-Pack routines and the extension routines accessed by EXTBIO on the workarea, in addition to several others made available by cartridges of expansion and of the BASIC interpreter routines.

The notation for routines is as follows:

**LABEL** (Routine address / location)

Function: describes the function of the routine.

Input: describes the parameters for calling the routine.

Output: describes the parameters for returning the routine.

Registers: lists the registers modified by the routine.

### 8.1 – Main-ROM ROUTINES

#### 8.1.1 – RST Routines

##### **CHKRAM** (0000H / Main)

Function: Tests RAM and initializes system variables. A call to this routine will cause a software reset.

Input: None.

Output: None.

Registers: All

##### **SYNCHR** (0008H / Main)

Function: Tests if the character pointed by (HL) is the one specified. If not, it generates "Syntax error"; otherwise it calls CHRGR (0010H).

Input: The character to be tested must be in (HL) and the character for comparison after the RST instruction (online parameter), as shown in the example below:

```
LD HL, CARACT
RST 008H
DEFB 'A'
|
CARACT: DEFB 'B'
```



Output: HL is incremented by 1 and A receives (HL). When the tested character is numeric, the CY flag is set; the end of declaration (00H or 3AH) sets the Z flag.

Registers: AF, HL.

### **RDSLT** (000CH / Main)

Function: Reads a memory byte in the slot specified in A. Interrupts are disabled during reading.

Input: A – 

|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
| E  | 0  | 0  | 0  | S  | S  | P  | P  |

Primary slot (0 a 3)

Secondary slot (0 a 3)

“1” if primary slot is expanded

HL – memory address to be read.

Output: A – contains the value of the read byte.

Registers: AF, BC, DE.

### **CHRGTR** (0010H / Main)

Function: Get a character (token) from the BASIC text.

Input: HL – address of the character to be read.

Output: HL is incremented by 1 and A receives (HL). When the character is numeric, the CY flag is set; the end of the declaration (00H or 3AH) sets the Z flag.

Registers: AF, HL.

### **WRSLT** (0014H / Main)

Function: Writes a memory byte in the slot specified in A. Interrupts are disabled during writing.

Input: A – slot indicator (same as RDSLT – 000CH).

HL – address for writing the byte.

E – byte to be written.

Output: None

Registers: AF, BC, D.

### **OUTDO** (0018H / Main)

Function: Sends a byte to the current device.

Input: A – Byte to be sent. If PRTFLG (F416H) is different of 0, the byte will sent to the printer; if PTRFIL (F864H) is

different of 0, the byte will sent to the file specified by PTRFIL.

Output: None.

Registers: None.

### **CALSLT** (001CH)

Function: Calls a routine in any slot (called an inter-slot).

Input: IY – the slot ID must be specified in the highest 8 bits in the same format as RDSLT (000CH).

IX – address of the routine to be called.

Output: It depends on the called routine.

Registers: It depends on the routine called.

### **DCOMPR** (0020H)

Function: Compare HL with DE.

Input: HL, DE.

Output: Set the Z flag if HL = DE; set the flag CY if HL < DE.

Registers: AF.

### **ENASLT** (0024H)

Function: Enables a page in any slot. Only pages 1 and 2 can be enabled by this routine; 0 and 3 do not. Interruptions are deactivated during enabling.

Input: A – Slot indicator (same as RDSLT – 000CH).

Output: None.

Registers: All.

### **GETYPR** (0028H / Main)

Function: Gets the type of operand contained in DAC.

Input: None

Output: Flags CY, S, Z and P / V, as shown in the table below:

|                   |      |      |      |        |
|-------------------|------|------|------|--------|
| Integer:          | C=1  | S=1* | Z=0  | P/V=1  |
| Simple precision: | C=1  | S=0  | Z=0  | P/V=0* |
| Double precision: | C=0* | S=0  | Z=0  | P/V=1  |
| String:           | C=1  | S=0  | Z=1* | P/V=1  |

Note: The types can be recognized by using only by the flags marked with “\*”.

Registers: AF.

**CALLF** (0030H / Main)

Function: Calls a routine in any slot using inline parameters. Very useful for calling routines through system hooks. The call sequence is as follows:

```
RST 030H ; calls CALLF
DEFB n   ; n is slot ID (same as RDSLT)
DEFW nn  ; nn is the address to be called
RET      ; return to the system
```

Input: By the method described.

Output: It depends on the called routine.

Registers: Depends on the called routine (plus AF).

**KEYINT** (0038H / Main)

Function: Performs the routine of interrupting and scanning the keyboard.

Input: None.

Output: None.

Registers: None.

**8.1.2 – Routines for I/O initialization****HOME** (0000H / Main)

Function: Initializes the input and output devices.

Input: None.

Output: None.

Registers: All.

**INIFNK** (003EH / Main)

Function: Initializes the contents of the function keys.

Input: None.

Output: None.

Registers: All.

**8.1.3 – Routines for accessing the VDP****DISSCR** (0041H / Main)

Function: Disables the screen presentation.

Input: None.

Output: None.

Registers: AF, BC.

**ENASCR** (0044H / Main)

Function: Enables the screen presentation.

Input: None.

Output: None.

Registers: AF, BC.

**WRTVDP** (0047H / Main)

Function: Writes a byte of data to a VDP register.

Input: C – register that will receive the data. It can vary from 0 to 7 for MSX1, from 0 to 23/32 to 46 for MSX2 and from 0 to 23/25 to 27/32 to 46 for MSX2+ or higher.

B – data byte

Output: None.

Registers: AF, BC.

**RDVRM** (004AH / Main)

Function: Read a VRAM byte. This routine reads only the lowest 14 address bits (16K for MSX1's TMS9918). To access the entire VRAM it is necessary to use the routine NRDVRM (0174H).

Input: HL – VRAM address to be read.

Output: A – byte read.

Registers: AF.

**WRTVRM** (004DH / Main)

Function: Writes a VRAM byte. This routine writes only the lowest 14 address bits (16K for MSX1's TMS9918). To access the entire VRAM, it is necessary to use the routine NWRVRM (0177H).

Input: HL – VRAM address to be written.

A – Byte to be written.

Output: None.

Registers: AF.

**SETRD** (0050H / Main)

Function: Prepare the VRAM for sequential reading using the VDP address auto-increment function. It is a faster means of reading than using a loop with the RDVRM routine (004AH). This routine accesses only the lowest 14 address bits (16K for MSX1's TMS9918). To access the entire VRAM, it is necessary to use the NSETRD routine (016EH).

Input: HL – Address at VRAM to start reading

Output: None.

Registers: AF.

### **SETWRT** (0053H / Main)

Function: Prepare the VRAM for sequential writing using the VDP address auto-increment function. The characteristics are the same as for SETRD (0050H). To access the entire VRAM it is necessary to use the routine NSTWRT (0171H).

Input: HL – VRAM address to start reading.

Output: None.

Registers: AF.

### **FILVRM** (0056H / Main)

Function: Fills an area of the VRAM with a single byte of data. This routine accesses only the lowest 14 address bits (16K for MSX1's TMS9918). To access the entire VRAM, it is necessary to use the BIGFIL routine (016BH).

Input: HL – VRAM address to start writing.

BC – Number of bytes to be written.

A – Byte to be written.

Output: None.

Registers: AF, BC.

### **LDIRMV** (0059H / Main)

Function: Copies a block of data from VRAM to RAM. All 16 address bits are valid.

Input: HL – Source address at VRAM.

DE – Destination address in RAM.

BC – Block size (length).

Output: None.

Registers: All.

### **LDIRVM** (005CH / Main)

Function: Copies a block of data from RAM to VRAM.

Input: HL – Source address in RAM.

DE – Destination address at VRAM.

BC – Block size (length).

Note: All 16 address bits are valid.

Output: None.

Registers: All.

### **CHGMOD** (005FH / Main)

Function: Switches the screen modes. This routine does not initialize the color palette. For this, it is necessary to use the routine CHGMDP (01B5H / Sub-ROM).

Input: A – 0 to 3 for MSX1, 0 to 8 for MSX2 or 0 to 12 for MSX2+ or higher (Note: Mode 9 is only valid for Korean machines).

Output: None.

Registers: All.

### **CHGCLR** (0062H / Main)

Function: Change the colors of the screen.

Input: FORCLR (F3E9H) – Front color

BAKCLR (F3EAH) – Background color

BDRCLR (F3EBH) – Border color

Output: None.

Registers: All.

### **NMI** (0066H / Main)

Function: Executes the NMI (Non-Maskable Interrupt) routine. On a standard MSX machine, it just makes a call to the HNMI hook (FDD6H) and returns without any processing.

Input: None.

Output: None.

Registers: None.

### **CLRSR** (0069H / Main)

Function: Initializes all sprites. The sprite pattern table is cleared (filled with zeros), the sprite numbers are initialized with the series 0 ~ 31 and the color of the sprites is equal to the background color. The vertical location of the sprites is set to 209 (for Screens 0 to 3) or 217 (for Screens 4 to 8 / 10 to 12).

Input: SCRMOD (FCAFH) – Screen mode.

Output: None.

Registers: All.

**INITXT** (006CH / Main)

Function: Initializes the screen in text mode (Screen 0). The color palette is not initialized. To initialize it, it is necessary to call the routine INIPLT (0141H / Sub-ROM).

Input: TXTNAM (F3B3H) – Name table address  
 TXTCGP (F3B7H) – Pattern table address  
 LINL40 (F3AEH) – Number of characters per line

Output: None.

Registers: All.

**INIT32** (006FH / Main)

Function: Initializes the screen in graphical mode 1 (Screen 1). The color palette is not initialized. To initialize it, it is necessary to call the routine INIPLT (0141H / Sub-ROM).

Input: T32NAM (F3BDH) – Address of the character name table.  
 T32COL (F3BFH) – Address of the character color table.  
 T32CGP (F3C1H) – Address of the character pattern table.  
 T32ATR (F3C3H) – Address of the sprites attribute table.  
 T32PAT (F3C5H) – Address of the sprites standards table.

Output: None.

Registers: All.

**INIGRP** (0072H / Main)

Function: Initializes the screen in the high resolution graphic mode of MSX1 (Screen 2). The color palette is not initialized. To initialize it, it is necessary to call the routine INIPLT. (0141H / Sub-ROM).

Input: GRPNAM (F3C7H) – Address of the pattern name table.  
 GRPCOL (F3C9H) – Address of the color table.  
 GRPCGP (F3CBH) – Address of the pattern generator table.  
 GRPATR (F3CDH) – Address of the sprites attribute table.  
 GRPPAT (F3CFH) – Address of the sprite standards table.

Output: None.

Registers: All.

**INIMLT** (0075H / Main)

Function: Initializes the screen in the MSX1 multicolor mode (Screen 3). The color palette is not initialized. To initialize it, it is necessary to call the routine INIPLT (0141H / Sub-ROM).

Input: MLTNAM (F3D1H) – Address of the pattern name table.  
 MLTCOL (F3D3H) – Address of the color table.  
 MLTCGP (F3D5H) – Adr of the pattern generator table.  
 MLTATR (F3D7H) – Address of the sprites attribute table.  
 MLTPAT (F3D9H) – Address of the sprites standards table.

Output: None.

Registers: All.

**SETTXT** (0078H / Main)

Function: Puts only the VDP in text mode (Screen 0).

Input: Same as INITXT (006CH).

Output: None.

Registers: All.

**SETT32** (007BH / Main)

Function: Puts only the VDP in graphical mode 1 (Screen 1).

Input: Same as INIT32 (006FH).

Output: None.

Registers: All.

**SETGRP** (007EH / Main)

Function: Puts only the VDP in graphical mode 2 (Screen 2).

Input: Same as INIGRP (0072H).

Output: None.

Registers: All.

**SETMLT** (0081H / Main)

Function: Puts only the VDP in multicolour mode (Screen 3).

Input: Same as INIMLT (0075H).

Output: None.

Registers: All.

**CALPAT** (0084H / Main)

Function: Returns the address of the sprite pattern generator table.

Input: A – Sprite number.

Output: HL – Address at VRAM.

Registers: AF, DE, HL.



**CALATR** (0087H / Main)

Function: Returns the address of a sprite's attribute table.

Input: A – Sprite number.

Output: HL – Address at VRAM.

Registers: AF, DE, HL.

**GSPSIZ** (008AH / Main)

Function: Returns the current size of the sprites.

Input: None.

Output: A – Size of the sprite in bytes. The CY flag is set if the size is 16 x 16 and reset otherwise.

Registers: AF.

**GRPPRT** (008DH / Main)

Function: Displays a character on a graphic screen.

Input: A – Character ASCII code. When the screen is 5 to 8 or 10 to 12, it is necessary to specify the logical operation code in LOGOPR (FB02H).

Output: None.

Registers: None.

**8.1.4 – Routines for access to PSG****GICINI** (0090H / Main)

Function: Initializes the PSG and sets the initial values for the PLAY command.

Input: None.

Output: None.

Registers: All.

**WRTPSG** (0093H / Main)

Function: Writes a byte of data to a PSG register.

Input: A – PSG registrar number.

E – Byte of data to be written.

Output: None.

Registers: None.

**RDPSG** (0096H / Main)

Function: Reads the contents of a PSG register.

Input: A – PSG registrar number.

Output: A – Byte read.

Registers: None.

**STRTMS** (0099H / Main)

Function: Tests whether the PLAY command is being executed. If not, start execution.

Input: None.

Output: None.

Registers: All.

**8.1.5 – Routines for accessing keyboard, screen and printer****CHSNS** (009CH / Main)

Function: Checks the keyboard buffer.

Input: None.

Output: If the Z flag is set, the buffer is empty; otherwise, the Z flag will be reset.

Registers: AF.

**CHGET** (009FH / Main)

Function: Input of a character by the keyboard with waiting.

Input: None.

Output: A – Character ASCII code.

Registers: AF.

**CHPUT** (00A2H / Main)

Function: Displays a character on the text screen.

Input: A – ASCII code of the character to be displayed.

Output: None.

Registers: None.

**LPTOUT** (00A5H / Main)

Function: Send a character to the printer.

Input: A – ASCII code of the character to be sent.

Output: If it fails, CY returns set.

Registers: F.

**LPTSTT** (00A8H / Main)

Function: Tests the status of the printer.

Input: None.

Output: A = 255 (and Z flag = 0) → printer ready.

A = 0 (and Z flag = 1) → printer is not ready.

Registers: AF.

**CNVCHR** (00A8H / Main)

Function: Tests the graphic header and converts if necessary.

Input: A – ASCII code of the character.

Output: CY = 0 – There is no graphic header.

CY = 1 and Z = 1 – The converted code is placed in A.

CY = 1 and Z = 0 – The unconverted code returns to A.

Registers: AF.

**PINLIN** (00AEH / Main)

Function: Collect a line of text and store it in a buffer until the RETURN or STOP key is pressed.

Input: None.

Output: HL – initial address of the buffer minus 1.

CY – set if the STOP key was pressed.

Registers: All.

**INLIN** (00B1H / Main)

Function: Same as PINLIN(00AEH), except that AUTFLG(F6AAH) is set.

Input: None.

Output: HL – initial address of the buffer minus 1.

CY – set if the STOP key was pressed.

Registers: All.

**QINLIN** (00B4H / Main)

Function: Executes INLIN (00B1H) presenting “?” and a space.

Input: None.

Output: HL – initial address of the buffer minus 1.

CY – set if the STOP key was pressed.

Registers: All.

**BREAKX** (00B7H / Main)

Function: Tests whether CTRL+STOP are pressed together. During verification, interruptions are disabled.

Input: None.

Output: CY – Set if CTRL+STOP are pressed.

Registers: AF.

**BEEP** (00C0H / Main)

Function: Generates a beep.

Input: None.

Output: None.

Registers: All.

**CLS** (00C3H / Main)

Function: Clears the screen.

Input: The Z flag must be set.

Output: None.

Registers: AF, BC, DE.

**POSIT** (00C6H / Main)

Function: Moves the cursor to a specific coordinate.

Input: H – X coordinate (horizontal)

L – Y coordinate (vertical)

Output: None.

Registers: AF.

**FNKSB** (00C9H / Main)

Function: Tests whether the commands associated with the function keys are being displayed on the screen by checking the FNKFLG (FBCEH) flag and inverts the display status (if the flag is on, off and if it is off, on).

Input: FNKFLG (FBCEH).

Output: None.

Registers: All.

**ERAFNK** (00CCH / Main)

Function: Turn off the display of the function keys.

Input: None.

Output: None.

Registers: All.

**DSPFNK** (00CFH / Main)

Function: Turns on the display of the function keys.

Input: None.

Output: None.

Registers: All.

**TOTEXT** (00D2H / Main)

Function: Forces the screen to text mode (Screen 0 or 1).

Input: None.

Output: None.

Registers: All.

**8.1.6 – I/O access routines for games****GTSTCK** (00D5H / Main)

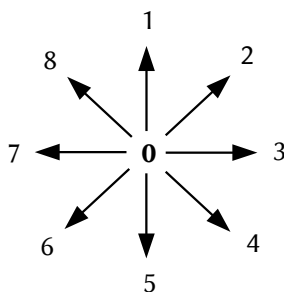
Function: Returns the state of the joystick or the cursor keys.

Input: A – 0 = Cursor keys.

1 = joystick on port 1.

2 = joystick on port 2.

Output: A – direction of the joystick or function keys as shown in the illustration below.



Registers: All.

**GTTRIG** (00D8H / Main)

Function: Returns the state of the mouse, joystick or keyboard bar buttons.

Input: A – 0 = Space bar.

1 = joystick on port 1, button A.

2 = joystick on port 2, button A.

3 = joystick on port 1, button B.

4 = joystick on port 2, button B.

Output: A – 0 = tested button is not pressed.  
           255 = tested button is pressed.

Registers: AF, BC.

### **GTPAD** (00DBH / Main)

Function Returns the state of a touch pad, trackball or mouse connected to one of the joystick connectors.

Input: A – 0 – Check touch pad on port 1 (255 if connected)  
           1 – Returns the X coordinate (horizontal).  
           2 – Returns the Y (vertical) coordinate.  
           3 – Returns the key state (255 if pressed).  
           4 – Check touch pad on port 2 (255 if connected).  
           5 – Returns the X (horizontal) coordinate.  
           6 – Returns the Y (vertical) coordinate.  
           7 – Returns the key state (255 if pressed).  
           8 – Check lightpen (255 if connected or touching pad).  
           9 – Returns the X (horizontal) coordinate.  
           10 – returns the Y (vertical) coordinate.  
           11 – returns the key state (255 if pressed).  
           12 – check mouse on port 1 (255 if connected).  
           13 – returns X coordinate offset (horizontal).  
           14 – returns Y coordinate offset (vertical).  
           15 – always 0.  
           16 – check mouse on port 2 (255 if connected).  
           17 – returns X coordinate offset (horizontal).  
           18 – returns Y coordinate offset (vertical).  
           19 – always 0.  
           20 – checks 2nd lightpen (255 if connected or touching the pad).  
           21 – returns the X coordinate (horizontal).  
           22 – returns the Y (vertical) coordinate.  
           23 – returns the key state (255 if pressed).

Output: A – state or value, as described above.

Registers: All.

Note: For function codes 8 to 23, call NEWPAD (01ADH) in SubROM. For the MSX turbo R, the pen functions (8 to 11) have been eliminated.

**GTPDL** (00DEH / Main)

Function: Returns the values of paddles connected to the joystick connectors.

Input: A – paddle identification (1 to 12).  
           1, 3, 5, 7, 9, 11 – Paddles connected to port 1.  
           2, 4, 6, 8, 10, 12 – Paddles connected to port 2.

Output: A – value read (0 to 255).

Registers: All.

Note: This routine was eliminated in the MSX turbo R.

**8.1.7 – I/O access routines for cassette register****TAPION** (00E1H / Main)

Function: Read the tape header after starting the cassette motor.

Input: None.

Output: If it fails, the CY flag returns set.

Registers: All.

Note: This routine was eliminated in the MSX turbo R.

**TAPIN** (00E4H / Main)

Function: Read data from the tape.

Input: None.

Output: A – byte read.  
           CY – set if the reading fails.

Registers: All.

Note: This routine was eliminated in the MSX turbo R.

**TAPIOF** (00E7H / Main)

Function: For reading the tape.

Input: None.

Output: None.

Registers: None.

Note: This routine was eliminated in the MSX turbo R.

**TAPOON** (00EAH / Main)

Function: Writes the header on the tape after starting the cas motor.

Input: A – 0 = Short header; another value = long header.

Output: If it fails, the CY flag returns set.

Registers: All.

Note: This routine was eliminated in the MSX turbo R.

**TAPOUT** (00EDH / Main)

Function: Writes data to the tape.

Input: A – Byte to be written.

Output: If it fails, the CY flag returns set.

Registers: All.

Note: This routine was eliminated in the MSX turbo R.

**TAPOOF** (00F0H / Main)

Function: For writing on the tape.

Input: None.

Output: If it fails, the CY flag returns set.

Registers: All.

Note: This routine was eliminated in the MSX turbo R.

**STMOTR** (00F3H / Main)

Function: Turns the cassette motor on or off.

Input: A – 0 = power on the motor

1 = power off the motor

255 = Inverts the state of the motor

Output: None.

Registers: AF.

Note: This routine was eliminated in the MSX turbo R.

**8.1.8 – Routines for the PSG queue****LFTQ** (00F6H / Main)

Function: Returns the number of free bytes in a PSG musical queue.

Input: A – queue number (0, 1 or 2).

Output: HL – free space left in the queue.

Registers: AF, BC, HL.

**PUTQ** (00F9H / Main)

Function: Place a byte in one of the PSG's musical queues.

Input: A – queue number (0, 1 or 2).

E – data byte.

Output: Flag Z set if the queue is full.

Registers: AF, BC, HL.



**GETVCP** (0150H / Main)

Function: Returns the address of byte 2 in the PSG's voice buffer.

Input: A – Voice number (0, 1 or 2)

Output: HL – address in the voice buffer.

Registers: AF, HL.

**GETVC2** (0153H / Main)

Function: Returns the address of any bytes in the PSG's voice buffer.

Input: VOICEN (FB38H) – Voice number (0, 1 or 2).

L – Byte number (0 to 36).

Output: HL – Address in the voice buffer.

Registers: AF, HL.

**8.1.9 – Routines for MSX1 graphics screens****RIGHTC** (00FCH / Main)

Function: Shifts the current pixel one position to the right.

Input: None.

Output: None.

Registers: AF.

**LEFTC** (00FFH / Main)

Function: Shifts the current pixel one position to the left.

Input: None.

Output: None.

Registers: AF.

**UPC** (0102H / Main)

Function: Shifts the current pixel one position up.

Input: None.

Output: None.

Registers: AF.

**TUPC** (0105H / Main)

Function: Tests the position of the current pixel and, if possible, moves it up one position.

Input: None.

Output: CY = 1 if the pixel could not be moved because it exceeds the upper limit of the screen.

Registers: AF.

**DOWNC** (0108H / Main)

Function: Shifts the current pixel down one position.

Input: None.

Output: None.

Registers: AF.

**TDOWNC** (010BH / Main)

Function: Tests the position of the current pixel and, if possible, moves it down one position.

Input: None.

Output: CY = 1 if the pixel could not be moved because it exceeds the lower limit of the screen.

Registers: AF.

**SCALXY** (010EH / Main)

Function: Limits the pixel coords to the visible area of the screen.

Input: BC – X coordinate (horizontal).

DE – Y coordinate (vertical).

Output: BC – X coordinate limited to the border.

DE – Y coordinate limited to the border.

CY = 1 if the coordinates are limited.

Registers: AF.

**MAPXYC** (0111H / Main)

Function: Converts a pair of graphic coordinates to the physical address of the current pixel (places the "cursor" on the coord).

Input: BC – X coordinate (horizontal).

DE – Y coordinate (vertical).

Output: None.

Registers: AF, D, HL.

**FETCHC** (0114H / Main)

Function: Returns the physical address of the current pixel.

Input: None.

Output: A ← content of CMASK (F92CH).

HL ← content of CLOC (F92AH).

Registers: A, HL.

**STOREC** (0117H / Main)

Function: Establishes the physical address of the current pixel.

Input: A is copied to CMASK (F92CH).

HL is copied to CLOC (F92AH).

Output: None.

Registers: None.

**SETATR** (011AH / Main)

Function: Establishes the color for the SETC (0120H) and NSETCX (0123H) routines.

Input: A – Color code (0 to 15).

Output: CY – Set if the color code is invalid.

Registers: F.

**READC** (011DH / Main)

Function: Returns the color code of the current pixel.

Input: None.

Output: A – Color code of the current pixel (0 to 15).

Registers: AF, EI.

**SETC** (0120H / Main)

Function: Establishes the color of the current pixel.

Input: ATRBYT (F3F2H) – Color code (0 to 15), established by SETATR (011AH).

Output: None.

Registers: AF, EI.

**NSETCX** (0123H / Main)

Function: Sets the color of multiple horizontal pixels starting from the current pixel, to the right.

Input: ATRBYT (F3F2H) – Color code (0 to 15), established by SETATR (011AH).

HL – Number of pixels to color.

Output: None.

Registers: AF, EI.

**GTASPC** (0126H / Main)

Function: Returns the aspect ratios of the CIRCLE statement.

Input: None.

Output: DE – Contents of ASPCT1 (F40BH).

HL – Contents of ASPCT2 (F40DH).

Registers: DE, HL.

**PNTINI** (0129H / Main)

Function: Establishes the outline color for the PAINT instruction.

Input: A – outline color code (0 to 15).

Output: CY – 1 if the color code is invalid.

Registers: AF.

**SCANR** (012CH / Main)

Function: Used by the PAINT instruction handler to scan an area, from left to right, starting from the current pixel until a color code equal to BDRATR (FCB2H) is found or the edge of the screen is reached.

Input: B – 0 = Does not fill the area covered.

255 = Fills the area covered.

DE – number of hops (pixels of the same color ignored).

Output: HL – number of pixels covered.

DE – number of hops remaining.

Registers: AF, BC, DE, HL, EI.

**SCANL** (012FH / Main)

Function: Same as SCANR (012CH), except that the route will be from right to left and the area will always be filled.

Input: None.

Output: HL – number of pixels covered.

Registers: AF, BC, DE, HL, EI.

**8.1.10 – Miscellaneous****CHGCAP** (0132H / Main)

Function: Changes the LED status of Caps Lock.

Input: A = 0 turns off the LED; another value, turn on the LED.

Output: None.

Registers: AF.

**CHGSND** (0135H / Main)

Function: Changes the state of the sound-generating 1-bit port.

Input: A = 0 turns the bit off, another value turns the bit on.

Output: None.

Registers: AF.

**RSLREG** (0138H / Main)

Function: Reads the contents of the primary slot register.

Input: None.

Output: A – Value read.

Registers: A.

**WSLREG** (013BH / Main)

Function: Writes to the primary slot register.

Input: A – Value to be written.

Output: None.

Registers: None.

**RDVDP** (013EH / Main)

Function: Read the VDP status register.

Input: None.

Output: A – Value read.

Registers: A.

**SNSMAT** (0141H / Main)

Function: Reads the value of a line from the keyboard matrix.

Input: A – Line to be read.

Output: A – Value read (the bit corresponding to a key pressed is 0).

Registers: AF, C.

**ISFLIO** (014AH / Main)

Function: Tests whether a device I/O operation is being performed.

Input: None.

Output: A = 0 if the device is active (I/O operation is being performed); another value the device is inactive.

Registers: AF.

**OUTDLP** (014DH / Main)

Function: Formatted output for the printer. It differs from LPTOUT in the following points:

- If the character sent is a TAB (09H) spaces will be sent until reaching a multiple of 8;
- For non-MSX printers, hiraganas are converted to katakanas and graphic characters are converted to 1-byte characters;
- If there is a failure, an I/O error will occur.

Input: A – Character to be sent.

Output: None.

Registers: F.

### **KILBUF** (0156H / Main)

Function: Clears the keyboard buffer.

Input: None.

Output: None.

Registers: HL.

### **CALBAS** (0159H / Main)

Function: Performs an inter-slot call to any BASIC interpreter routine.

Input: IX – Address to be called.

Output: It depends on the routine called.

Registers: It depends on the routine called.

## **8.1.11 – Routines for accessing the disk system**

### **PHYDIO** (0144H / Main)

Function: Read or write one or more sectors on the specified drive.

Input: CY – 0 = Reading.

1 = writing.

A – drive number (0 = A:, 1 = B:, etc).

B – number of sectors to read or write.

C – Disk formatting ID:

F0H – 63 sectors per track (for HD's)

F8H – 80 tracks, 9 sectors per track, single face.

F9H – 80 tracks, 9 sectors per track, double sided.

FAH – 80 tracks, 8 sectors per track, single face.

FBH – 80 tracks, 8 sectors per track, double sided.

FCH – 40 tracks, 9 sectors per track, single face.

FDH – 40 tracks, 9 sectors per track, double sided.

DE – Number of the first sector to be read or written.

HL – RAM address from which the sectors to be read from the disk will be written or the sectors to be written to the disk will be removed.

Output: CY – set if there was a reading or writing error.

A – error code if CY = 1:

0 – Write-protected.

2 – Not ready.

4 – Data error.

6 – Seek error.

8 – Sector not found.

10 – Writing error.

12 – Invalid parameters.

14 – Insufficient memory.

16 – Undefined error.

B – Number of sectors actually read or written.

Registers: All.

Note: In some HD interfaces, when bit 7 of register C is set, a 23-bit addressing scheme will be used and bits 0-6 of register C must contain bits 23-16 of the number of the sector.

#### **FORMAT** (0147H / Main)

Function: Format a floppy disk. When called, a series of questions will be presented that must be answered to start the formatting. There is no standard for these questions; they can be different for each drive interface.

Input: None.

Output: None.

Registers: None.

### **8.1.12 – Routines added for MSX2**

#### **SUBROM** (015CH / Main)

Function: Performs an inter-slot call to SubROM.

Input: IX – Address to be called (at the same time put IX on the stack).

Output: It depends on the called routine.

Registers: IY, AF', BC', DE', HL', and the registers modified by the called routine.

#### **EXTROM** (015FH / Main)

Function: Performs an inter-slot call to SubROM.

Input: IX – Address to be called.

Output: It depends on the called routine.

Registers: IY, AF', BC', DE', HL', and the registers modified by the called routine.

### **CHKSLZ** (0162H / Main)

Function: Searches for slots for the SubROM.

Input: None.

Output: None.

Registers: All.

### **CHKNEW** (0165H / Main)

Function: Tests the screen mode.

Input: None.

Output: CY = 1 if screen is 5, 6, 7 or 8.

Registers: AF.

### **EOL** (0168H / Main)

Function: Erase until the end of the line.

Input: H – X coordinate of the cursor.

L – Y coordinate of the cursor.

Output: None.

Registers: All.

### **BIGFIL** (016BH / Main)

Function: Fills an area of RAM with a single byte of data. The Screens 0 to 3 are not tested and filling can exceed the 16K limit of these screens. See FILVRM (0056H) on Main ROM.

Input: L – VRAM address to start writing.

BC – Number of bytes to be written.

A – Byte to be written.

Output: None.

Registers: AF, BC.

### **NSETRD** (016EH / Main)

Function: Prepares VRAM for sequential reading using the VDP address auto-increment function.

Input: HL – VRAM address from which the data will be read. All bits are valid.

Output: None.

Registers: AF.



**NSTWRT** (0171H / Main)

Function: Prepares VRAM for sequential writing using the VDP address auto-increment function.

Input: HL – VRAM address from which the data will be written.  
All bits are valid.

Output: None.

Registers: AF.

**NRDVRM** (0174H / Main)

Function: Read the content of one byte of the VRAM.

Input: HL – VRAM address to be read.

Output: A – byte read.

Registers: AF.

**NWRVRM** (0177H / Main)

Function: Writes one byte of data to VRAM.

Input: HL – VRAM address to be written.  
A – byte to be written.

Output: None.

Registers: AF.

**8.1.13 – Routines added for MSX2+****RDRES** (017AH / Main)

Function: Returns the reset status.

Input: None.

Output: A – b7 = 0 indicates total reset (by hardware)  
b7 = 1 indicates partial reset (by software)

Registers: A.

Note: In the total reset (by hardware) the RAM content is cleared and the MSX logo appears at startup. In the partial reset (by software) the RAM content is not erased (only the desktop is initialized) and the MSX logo does not appear at startup.

**WRRES** (017DH / Main)

Function: Modifies the reset status.

Input: A – b7 = 0 for total reset (by hardware)  
b7 = 1 for partial reset (by software)

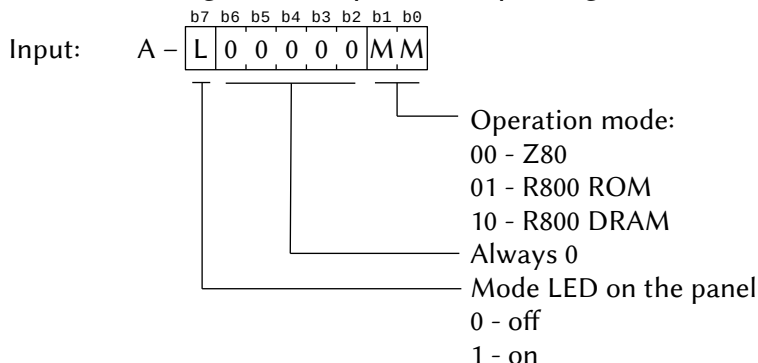
Output: None.

Registers: None.

### 8.1.14 – Routines added for the MSX turbo R

#### CHGCPU (0180H / Main)

Function: Change the microprocessor (operating mode).



Output: None.

Registers: AF.

#### GETCPU (0183H / Main)

Function: Returns in which mode the computer is operating.

Input: None.

Output: A – 0 = Z80; 1 = R800 ROM; 2 = R800 DRAM.

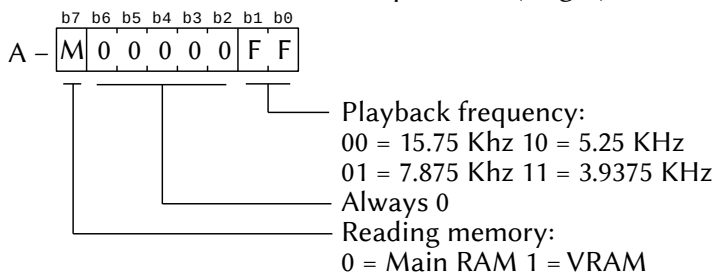
Registers: AF.

#### PCMPY (0186H / Main)

Function: Play sounds through the PCM.

Input: EHL – Address to start reading.

DBC – Size of the block to be reproduced (length).



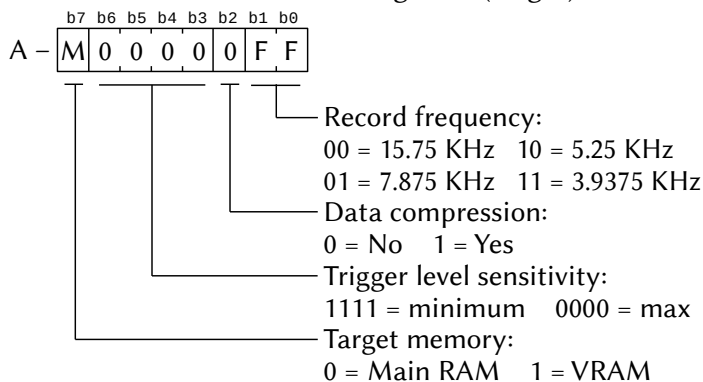
Note: The 15.75 KHz frequency can only be used in the R800 DRAM mode.

Output: CY – 0 → Playback OK.  
           1 → Playback error.  
           Cause of error:  
       A – 0 → error in specifying the frequency.  
           1 → interruption by CTRL+STOP.  
       EHL – Address as far as it actually reproduced.  
 Registers: All.

### PCMREC (0189H / Main)

Function: Digitize sounds through the PCM.

Input: EHL – Address to start reading.  
       DBC – Size of the block to be digitized (length).



Note: The 15.75 KHz frequency can only be used in R800 DRAM mode.

Output: CY – 0 → Record OK.  
           1 → Record error.  
           Cause of error:  
       A – 0 → error in specifying the frequency.  
           1 → interruption by CTRL+STOP.  
       EHL – Address as far as it actually recorded.  
 Registers: All.

## 8.1.15 – Inter-slot work area routines

### RDPRIM (F380H / Work Area)

Function: Reads a byte from any address in any slot.

Input: A – Primary slot to be read.  
       D – Current return slot.

Output: E – Byte read.

**WRPRIM** (F385H / Work Area)

Function: Writes a byte to any address in any slot.

Input: A – Primary slot to be read.

D – Current return slot.

E – Byte to be written.

Output: None

**CLPRIM** (F38CH / Work Area)

Function: Calls an address in any slot

Input: A – Primary slot containing the routine

IX – Address to be called

PUSH AF – Current return slot (in A)

Output: Depends of the called routine

**8.2 – SubROM ROUTINES****8.2.1 – Routines for BASIC graphical functions****PAINT** (0069H / SubROM) – BASIC Command

Function: Paints an area on a graphic screen.

Input: HL – Pointer to the beginning of the BASIC text  
(parameters of the PAINT command).

Output: HL – Points to the end of the command parameters.

Registers: All.

**PSET** (006DH / SubROM) – BASIC Command

Function: Draws a point on a graphic screen.

Input: HL – Pointer to the beginning of the BASIC text  
(parameters of the PSET command).

Output: HL – Points to the end of the command parameters.

Registers: All.

**ATRSCN** (0071H / SubROM) – BASIC Command

Function: Returns color attributes.

Input: HL – Pointer to the beginning of the BASIC text.

Output: HL – Points to the end of the command parameters.

Registers: All.

**GLINE** (0075H / SubROM) – BASIC Command

Function: Draws a line on a graphic screen.

Input: HL – Pointer to the beginning of the BASIC text.

Output: HL – Points to the end of the command parameters.

Registers: All.

**DOBOXF** (0079H / SubROM) – BASIC Command

Function: Draws a filled rectangle on a graphic screen.

Input: HL – Pointer to the beginning of the BASIC text.

Output: HL – Points to the end of the command parameters.

Registers: All.

**DOLINE** (007DH / SubROM) – BASIC Command

Function: Draws a line on a graphic screen.

Input: HL – Pointer to the beginning of the BASIC text  
(parameters of the LINE command).

Output: HL – Points to the end of the command parameters.

Registers: All.

**BOXLIN** (0081H / SubROM) – BASIC Command

Function: Draws a rectangle on a graphic screen.

Input: HL – Pointer to the beginning of the BASIC text.

Output: HL – Points to the end of the command parameters.

Registers: All.

**PUTSPR** (0151H / SubROM) – BASIC Command

Function: Displays a sprite on a graphical screen.

Input: HL – Pointer to the beginning of the BASIC text.

Output: HL – Points to the end of the command parameters.

Registers: All.

**COLOR** (0155H / SubROM) – BASIC command

Function: Change the colors of the screen, sprites or palette.

Input: HL – Pointer to the beginning of the BASIC text.

Output: HL – Points to the end of the command parameters.

Registers: All.

**SCREEN** (0159H / SubROM) – BASIC command

Function: Switches the screen modes.

Input: HL – Pointer to the beginning of the BASIC text.

Output: HL – Points to the end of the command parameters.

Registers: All.

**WIDTH** (015DH / SubROM) – BASIC Command

Function: Changes the number of characters per line in text mode.

Input: HL – Pointer to the beginning of the BASIC text.

Output: HL – Points to the end of the command parameters.

Registers: All.

**VDP** (0161H / SubROM) – BASIC Command

Function: Writes data to a VDP register.

Input: HL – Pointer to the beginning of the BASIC text.

Output: HL – Points to the end of the command parameters.

Registers: All.

**VDPF** (0165H / SubROM) – BASIC Command

Function: Reads data from a VDP register.

Input: HL – Pointer to the beginning of the BASIC text.

Output: HL – Points to the end of the command parameters.

Registers: All.

**BASE** (0169H / SubROM) – BASIC command

Function: Writes data to the VDP base register.

Input: HL – Pointer to the beginning of the BASIC text.

Output: HL – Points to the end of the command parameters.

Registers: All.

**BASEF** (0169H / SubROM) – BASIC Command

Function: Reads data from the VDP base register.

Input: HL – Pointer to the beginning of the BASIC text.

Output: HL – Points to the end of the command parameters.

Registers: All.

## 8.2.2 – Routines for graphical functions

### **DOGRPH** (0085H / SubROM)

Function: Draws a line on a graphic screen.

Input: BC – Initial X coordinate.

HL – Initial Y coordinate.

GXPOS (FCB3H) – Final X coordinate.

GYPOS (FCB5H) – Y coordinate at the end.

ATRBYT (F3F2H) – Attributes.

LOGOPR (FB02H) – Logical operation code.

Output: None.

Registers: AF.

### **GRPPRT** (0089H / SubROM)

Function: Prints a character on a graphical MSX2 screen.

Input: A – Character ASCII code.

ATRBYT (F3F2H) – Attributes.

LOGOPR (FB02H) – Logical operation code.

Output: None.

Registers: All.

### **SCALXY** (008DH / SubROM)

Function: Limits the pixel coordinates to the screen visible area.

Input: BC – X coordinate (horizontal).

DE – Y coordinate (vertical).

Output: BC – X coordinate limited to the border.

DE – Y coordinate limited to the border.

CY = 1 if the coordinates was limited.

Registers: AF.

### **MAPXYC** (0091H / SubROM)

Function: Converts a pair of graphic coordinates to the physical address of the current pixel (puts the "cursor" on the coord).

Input: BC – X coordinate (horizontal).

DE – Y coordinate (vertical).

Output: Screen 3: HL, CLOC (F92AH) – Address at VRAM.

A, CMASK (F92CH) – Mask.

Screen 5~12: HL, CLOC (F92AH) – X coordinate.

A, CMASK (F92CH) – Y coordinate.

Registers: F.

**READC** (0095H / SubROM)

Function: Read the attributes of a pixel.

Input: CLOC (F92AH) – X Coordinate.  
CMASK (F92CH) – Y coordinate.

Output: A – Attribute.

Registers: AF.

**SETATR** (0099H / SubROM)

Function: Defines attribute in ATRBYT (F3F2H).

Input: A – Attribute.

Output: CY = 1 if there is an error in the attribute.

Registers: F.

**SETC** (009DH / SubROM)

Function: Defines pixel attribute.

Input: CLOC (F92AH) – X Coordinate.  
CMASK (F92CH) – Y coordinate.  
ATRBYT (F3F2H) – Attribute.

Output: None.

Registers: AF.

**TRIGHT** (00A1H / SubROM)

Function: Moves one pixel to the right.

Input: CLOC (F92AH) – X Coordinate.  
CMASK (F92CH) – Y coordinate.

Output: CLOC (F92AH) – New X coordinate.  
CMASK (F92CH) – New Y coordinate.  
CY = 1 if the edge of the screen is reached.

Registers: AF.

Note: Only for Screen 3.

**RIGHTC** (00A5H / SubROM)

Function: Moves one pixel to the right.

Input: CLOC (F92AH) – X Coordinate.  
CMASK (F92CH) – Y coordinate.

Output: CLOC (F92AH) – New X coordinate.  
CMASK (F92CH) – New Y coordinate.

Registers: AF.

Note: Only for Screen 3. This routine is the same as TRIGHT (00A1H) except for the absence of the return of the CY flag.



**TLEFTC** (00A9H / SubROM)

Function: Moves one pixel to the left.

Input: Same as TRIGHT (00A1H / SubROM).

Output: Same as TRIGHT (00A1H / SubROM).

Registers: AF.

Note: Only for Screen 3.

**LEFTC** (00ADH / SubROM)

Function: Moves one pixel to the left.

Input: Same as RIGHTC (00A5H / SubROM).

Output: Same as RIGHTC (00A5H / SubROM).

Registers: AF.

Note: Only for Screen 3. This routine is the same as TLEFTC (00A9H) except for the absence of the return of the CY flag.

**TDOWNC** (00B1H / SubROM)

Function: Moves down one pixel.

Input: Same as TRIGHT (00A1H / SubROM).

Output: Same as TRIGHT (00A1H / SubROM).

Registers: AF.

Note: Only for Screen 3.

**DOWNC** (00B5H / SubROM)

Function: Moves down one pixel.

Input: Same as RIGHTC (00A5H / SubROM).

Output: Same as RIGHTC (00A5H / SubROM).

Registers: AF.

Note: Only for Screen 3. This routine is the same as TDOWNC (00A9H) except for the absence of the return of the CY flag.

**TUPC** (00B9H / SubROM)

Function: Moves up one pixel.

Input: Same as TRIGHT (00A1H / SubROM).

Output: Same as TRIGHT (00A1H / SubROM).

Registers: AF.

Note: Only for Screen 3.

**UPC** (00BDH / SubROM)

Function: Moves up one pixel.

Input: Same as RIGHTC (00A5H / SubROM).

Output: Same as RIGHTC (00A5H / SubROM).

Registers: AF.

Note: Only for Screen 3. This routine is the same as TUPC (00B9H) except for the absence of the CY flag return.

**SCANR** (00C1H / SubROM)

Function: Scan pixels, from left to right, starting from the current pixel until a color code equal to BDRATR (FCB2H) is found or the edge of the screen is reached.

Input: B – 0 = Does not fill the area covered.  
255 = Fills the area covered.

C – counter to the edge.

Output: DE – counter to the edge.

C – modified pixel flag.

Registers: All.

**SCANL** (00C5H / SubROM)

Function: Scan pixels, from right to left, starting from the current pixel until a color code equal to BDRATR (FCB2H) is found or the edge of the screen is reached.

Input: DE – counter to the edge.

Output: DE – counter to the edge.

C – modified pixel flag.

Registers: All.

**NVBXLN** (00C9H / SubROM)

Function: Draws a lined rectangle.

Input: BC – Initial X coordinate.

HL – Initial Y coordinate.

GXPOS (FCB3H) – Final X coordinate.

GYPOS (FCB5H) – Y coordinate at the end.

ATRBYT (F3F2H) – Attributes.

LOGOPR (FB02H) – Logic operation code.

Output: None.

Registers: All.

**NVBXFL** (00CDH / SubROM)

Function: Draws a filled rectangle.

Input: Same as NVBXLN (00C9H / SubROM).

Output: None.

Registers: All.

**8.2.3 – Duplicate routines (same as MainROM)****CHGMOD** (00D1H / SubROM)

Function: Switches the screen modes.

Input: A – 0 to 3 for MSX1, 0 to 8 for MSX2 or 0 to 12 for MSX2+ or higher (Mode 9 is valid only for Korean computers).

Output: None.

Registers: All.

**INITXT** (00D5H / SubROM)

Function: Initializes the screen in text mode (Screen 0).

Input: TXTNAM (F3B3H) – Name table address.

TXTCGP (F3B7H) – Pattern table address.

LINL40 (F3AEH) – Number of characters per line.

Output: None.

Registers: All.

**INIT32** (00D9H / SubROM)

Function: Initializes the screen in Screen 1 mode.

Input: T32NAM (F3BDH) – Characters name table address.

T32COL (F3BFH) – Characters color table address.

T32CGP (F3C1H) – Characters patterns table address.

T32ATR (F3C3H) – Sprites attributes table address.

T32PAT (F3C5H) – Sprites patterns table address.

Output: None.

Registers: All.

**INIGRP** (00DDH / SubROM)

Function: Initializes the screen in Screen 2 mode.

Input: GRPNAM (F3C7H) – Patterns name table address.

GRPCOL (F3C9H) – Color table address.

GRPCGP (F3CBH) – Patterns generator table address.

GRPATR (F3CDH) – Sprites attributes table address.

GRPPAT (F3CFH) – Sprites patterns table address.

Output: None.

Registers: All.

### **INIMLT** (00E1H / SubROM)

Function: Initializes the screen in the multicolor mode (Screen 3).

Input: MLTNAM (F3D1H) – Patterns name table address.

MLTCOL (F3D3H) – Color table address.

MLTCGP (F3D5H) – Patterns generator table address.

MLTATR (F3D7H) – Sprites attributes table address.

MLTPAT (F3D9H) – Sprites patterns table address.

Output: None.

Registers: All.

### **SETTXT** (00E5H / SubROM)

Function: Puts only the VDP in text mode (Screen 0).

Input: Same as INITXT (00D5H / SubROM).

Output: None.

Registers: All.

### **SETT32** (00E9H / SubROM)

Function: Puts only the VDP in graphical mode 1 (Screen 1).

Input: Same as INIT32 (00D9H / SubROM).

Output: None.

Registers: All.

### **SETGRP** (00EDH / SubROM)

Function: Puts only the VDP in graphical mode 2 (Screen 2).

Input: Same as INIGRP (00E1H / SubROM).

Output: None.

Registers: All.

### **SETMLT** (00F1H / SubROM)

Function: Puts only the VDP in multicolour mode (Screen 3).

Input: Same as INIMLT (0075H).

Output: None.

Registers: All.

**CLRSPR** (00F5H / SubROM)

Function: Initializes all sprites. The sprite pattern table is cleared (filled with zeros), the sprite numbers are initialized with the series 0 ~ 31 and the color of the sprites is equal to the background color. The vertical location of the sprites is set to 209 (Screens 0 to 3) or 217 (Screens 4 to 9 or 10 to 12).

Input: SCRMOD (FCAFH) – Screen mode.

Output: None.

Registers: All.

**CALPAT** (00F9H / SubROM)

Function: Returns the address of the sprite pattern generator table.

Input: A – Sprite number.

Output: HL – Address at VRAM.

Registers: AF, DE, HL.

**CALATR** (00FDH / SubROM)

Function: Returns the address of a sprite's attribute table.

Input: A – Sprite number.

Output: HL – Address at VRAM.

Registers: AF, DE, HL.

**GSPSIZ** (0101H / SubROM)

Function: Returns the current size of the sprites.

Input: None.

Output: A – Size of the sprite in bytes. The CY flag is set if the size is 16 x 16 and reset otherwise.

Registers: AF.

**8.2.4 – Various routines for MSX2 or higher****GETPAT** (0105H / SubROM)

Function: Returns the pattern of a character.

Input: A – ASCII code of the character.

Output: PATWRK (FC40H) – Character standard.

Registers: All.

**WRTVRM** (0109H / SubROM)

Function: Writes one byte of data to VRAM.

Input: HL – Address of VRAM.

A – byte to be written.

Output: None.

Registers: AF.

**RDVRM** (010DH / SubROM)

Function: Read the content of one byte of VRAM.

Input: HL – VRAM address to be read.

Output: A – byte read.

Registers: AF.

**CHGCLR** (0111H / SubROM)

Function: Change the colors of the screen.

Input: FORCLR (F3E9H) – Front color

BAKCLR (F3EAH) – Background color

BDRCLR (F3EBH) – Border color

Output: None.

Registers: All.

**CLSSUB** (0115H / SubROM)

Function: Clear the screen.

Input: None.

Output: None.

Registers: All.

**CLRTXT** (0119H / SubROM)

Function: Clear text screen.

Input: None.

Output: None.

Registers: All.

**DSPFNK** (011DH / SubROM)

Function: Displays the content of the function keys.

Input: None.

Output: None.

Registers: All.

**DELLNO** (0121H / SubROM)

Function: Deletes a line in text mode.

Input: L – Number of the line to be deleted.

Output: None.

Registers: All.

**INSLNO** (0125H / SubROM)

Function: Adds a line in text mode.

Input: L – Line number to be added.

Output: None.

Registers: All.

**PUTVRM** (0129H / SubROM)

Function: Place a character on a text screen.

Input: H – Y coordinate.

L – X coordinate.

Output: None.

Registers: AF.

**WRTVDP** (012DH / SubROM)

Function: Writes a byte of data to a VDP register.

Input: C – number of the registrar that will receive the data.

B – data byte.

Output: None.

Registers: AF, BC.

**VDPSTA** (0131H / SubROM)

Function: Read the contents of a VDP register.

Input: A – Number of the register to be read (0 to 9).

Output: A – Value read.

Registers: F.

**KYKLOK** (0135H / SubROM)

Function: Control of the KANA key and the KANA LED on Japanese computers.

Input: ?

Output: ?

Registers: ?

**PUTCHR** (0139H / SubROM)

Function: Take a key code, convert it to KANA and put it in a buffer (on Japanese computers).

Input: CY = 0 – Make conversion.

CY = 1 – Does not convert.

Output: ?

Registers: All.

**SETPAG** (013DH / SubROM)

Function: Defines the video pages.

Input: DPPAGE (FAF5H) – Page shown on the screen.

ACPAGE (FAF6H) – Active page for receiving commands.

Output: None.

Registers: AF.

**NEWPAD** (01ADH / SubROM)

Function: Returns the state of the mouse or the lightpen.

Input: A – function code:

0 to 7 – No effect.

8 – Check lightpen (255 if connected/touching screen).

9 – Returns the X (horizontal) coordinate.

10 – Returns the Y (vertical) coordinate.

11 – Returns the button state (255 if pressed).

12 – Check mouse on port 1 (255 if connected).

13 – Returns X coordinate offset (horizontal).

14 – Returns Y coordinate offset (vertical).

15 – Always 0.

16 – Check mouse on port 2 (255 if connected).

17 – Returns X coordinate offset (horizontal).

18 – Returns Y coordinate offset (vertical).

19 – Always 0.

20 – Check 2nd lightpen (255 if connected/touch screen).

21 – Returns the X coordinate (horizontal).

22 – Returns the Y (vertical) coordinate.

23 – Returns the button state (255 if pressed).

Output: A – state or value, as described above.

Registers: All.



**CHGMDP** (01B5H / SubROM)

Function: Switches the screen modes and initializes the color palette.

Input: A – 0 to 3 for MSX1, 0 to 8 for MSX2 or 0 to 12 for MSX2+ or higher (Mode 9 is valid only for Korean computers).

Output: None.

Registers: All.

**KNJPRT** (01BDH / SubROM)

Function: Writes a Kanji character on a graphic screen (Screens 5 to 8 or 10 to 12). This routine is present only in machines with Kanji ROM.

Input: BC – Kanji character JIS code.

A – Presentation mode:

0 – All lines on the screen.

1 – Even lines.

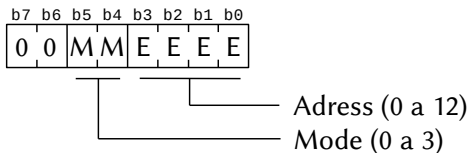
2 – Odd lines.

Registers: AF.

**REDCLK** (01F5H / SubROM)

Function: Reads a nibble of data from the clock memory (Clock-IC).

Input: C – SRAM address of the clock, as shown below:



Output: A – Nibble read (4 bits lower).

Registers: AF.

**WRTCLK** (01F9H / SubROM)

Function: Writes a data nibble to the clock's memory

Input: C – SRAM address of the clock (equal to REDCLK).

A – Nibble to be written (4 bits lower).

Output: None.

Registers: F.

## 8.2.5 – Color palette handling routines

### **INIPLT** (0141H / SubROM)

Function: Initializes the color palette (the current palette will be saved in VRAM).

Input: None.

Output: None.

Registers: AF, BC, DE.

### **RSTPLT** (0145H / SubROM)

Function: Retrieves the color palette saved in VRAM.

Input: None.

Output: None.

Registers: AF, BC, DE.

### **GETPLT** (0149H / SubROM)

Function: Returns the color levels of the palette.

Input: A – color number in the palette (0 to 15).

Output: B – 4 bits high for the red level.

B – 4 bits low for the blue level.

C – 4 bits low for green level.

Registers: AF, DE.

### **SETPLT** (014DH / SubROM)

Function: Modifies the color levels of the palette.

Input: D – color number in the palette (0 to 15).

A – 4 bits high for the red level.

A – 4 bits low for the blue level.

E – 4 bits low for green level.

Output: None.

Registers: AF.

## 8.2.6 – Various routines used by BASIC

### **VPOKE** (0171H / SubROM) – BASIC Command

Function: Writes one byte of data to VRAM.

Input: HL – Pointer to the beginning of the BASIC text.

Output: HL – Points to the end of the command parameters.

Registers: All.

**VPEEK** (0175H / SubROM) – BASIC Command

Function: Reads one byte of VRAM data.

Input: HL – Pointer to the beginning of the BASIC text.

Output: HL – Points to the end of the command parameters.

Registers: All.

**SETS** (0179H / SubROM) – BASIC Command

Function: Executes the parameters of the BEEP, ADJUST, TIME and DATE commands

Input: HL – Pointer to the beginning of the BASIC text.

Output: HL – Points to the end of the command parameters.

Registers: All.

**BEEP** (017DH / SubROM) – BASIC Command

Function: Generates a beep.

Input: HL – Pointer to the beginning of the BASIC text.

Output: HL – Points to the end of the command parameters.

Registers: All.

**PROMPT** (0181H / SubROM) – BASIC Command

Function: Displays the BASIC prompt ("Ok" by default).

Input: HL – Pointer to the beginning of the BASIC text.

Output: HL – Points to the end of the command parameters.

Registers: All.

**SDFSCR** (0185H / SubROM) – BASIC command

Function: Retrieves the screen parameters of the Clock-IC. When CY = 1, the content of the function keys will be displayed.

Input: CY = 0 after calling MSXDOS.

Output: ?

Registers: All.

**SETSCR** (0189H / SubROM) – BASIC Command

Function: Retrieves the screen parameters of the Clock-IC and displays a welcome message.

Input: ?

Output: ?

Registers: All.

**SCOPY** (018DH / SubROM) – BASIC Command

Function: Executes copies between VRAM, BASIC matrices and files on disk.

Input: HL – Pointer to the beginning of the BASIC text.

Output: HL – Points to the end of the command parameters.

Registers: All.

**GETPUT** (01B1H / SubROM) – BASIC command

Function: Executes the parameters of the GET TIME, GET DATE and PUT KANJI commands.

Input: HL – Pointer to the beginning of the BASIC text.

Output: HL – Points to the end of the command parameters.

Registers: All.

**8.2.7 – Block transfer routines (bit-blit)****BLTVV** (0191H / SubROM)

Function: Transfer data from one VRAM area to another.

Input: HL – Must contain the value F562H.

SX – (F562H, 2) – X coordinate of the source.

SY – (F564H, 2) – Y coordinate of the source.

DX – (F566H, 2) – X coordinate of destination.

DY – (F568H, 2) – Y coordinate of destination.

NX – (F56AH, 2) – Number of pixels in the X direction.

NY – (F56CH, 2) – Number of pixels in the Y direction.

CDUMMY – (F56EH, 1) – Dummy (no data required).

ARGT – (F56FH, 1) – Selects the direction and the expanded VRAM (equal to R # 45 of the VDP).

LOGOP – (F570H, 1) – Logical operation code (same as VDP codes).

Output: CY = 0.

Registers: All.

**BLTVM** (0195H / SubROM)

Function: Transfer data from Main RAM to VRAM.

Input: HL – Must contain the value F562H.

DPTR – (F562H, 2) – Source address in RAM.

DUMMY – (F564H, 2) – Dummy (no data required).

DX – (F566H, 2) – X coordinate of destination.  
 DY – (F568H, 2) – Y coordinate of destination.  
 NX – (F56AH, 2) – Number of pixels in the X direction  
 (no data required; already filled in).  
 NY – (F56CH, 2) – Number of pixels in the Y direction  
 (no data required; already filled in).  
 CDUMMY– (F56EH, 1) – Dummy (no data required).  
 ARGV – (F56FH, 1) – Selects the direction and the  
 expanded VRAM (equal to R # 45 of the VDP).  
 LOGOP – (F570H, 1) – Logical operation code (same as  
 VDP codes).

Output: CY = 0 – Transfer successful.  
 CY = 1 – Transfer error.

Registers: All.

Note: The memory space to be allocated, in bytes, must obey the following formulas:

Screen 6:  $(NX * NY) / 4 + 4$

Screens 5 and 7:  $(NX * NY) / 2 + 4$

Screens 8, 10, 11 and 12:  $(NX * NY) + 4$

### **BLTMV** (0199H / SubROM)

Function: Transfer data from VRAM to Main RAM.

Input: HL – Must contain the value F562H.  
 SX – (F562H, 2) – X coordinate of the source.  
 SY – (F564H, 2) – Y coordinate of the source.  
 DPTR – (F566H, 2) – Destination address in RAM.  
 DUMMY – (F568H, 2) – Dummy (no data required).  
 NX – (F56AH, 2) – Number of pixels in the X direction.  
 NY – (F56CH, 2) – Number of pixels in the Y direction.  
 CDUMMY– (F56EH, 1) – Dummy (no data required).  
 ARGV – (F56FH, 1) – Selects the direction and the  
 expanded VRAM (equal to R # 45 of the VDP).

Output: CY = 0.

Registers: All.

Note: The memory space to be allocated, in bytes, must obey the following formulas:

Screen 6:  $(NX * NY) / 4 + 4$

Screens 5 and 7:  $(NX * NY) / 2 + 4$

Screens 8, 10, 11 and 12:  $(NX * NY) + 4$

**BLTVD** (019DH / SubROM)

Function: Transfer data from disk to VRAM.

Input: HL – Must contain the value F562H.

FNPTR – (F562H, 2) – Address of the filename.

DUMMY – (F564H, 2) – Dummy (no data required).

DX – (F566H, 2) – X coordinate of destination.

DY – (F568H, 2) – Y coordinate of destination.

NX – (F56AH, 2) – Number of pixels in the X direction  
(no data required; already filled in).

NY – (F56CH, 2) – Number of pixels in the Y direction  
(no data required; already filled in).

CDUMMY– (F56EH, 1) – Dummy (no data required).

ARGT – (F56FH, 1) – Selects the direction and the  
expanded VRAM (same as R # 45 of the VDP).

LOGOP – (F570H, 1) – Logical operation code (same as  
VDP codes).

Output: CY = 0 – Transfer successful.

CY = 1 – Error in the transfer or in the parameters.

Registers: All.

**BLTDV** (01A1H / SubROM)

Function: Transfer data from VRAM to disk.

Input: HL – Must contain the value F562H.

SX – (F562H, 2) – X coordinate of the source.

SY – (F564H, 2) – Y coordinate of the source.

FNPTR – (F566H, 2) – Address of the filename.

DUMMY – (F568H, 2) – Dummy (no data required).

NX – (F56AH, 2) – Number of pixels in the X direction.

NY – (F56CH, 2) – Number of pixels in the Y direction.

CDUMMY– (F56EH, 1) – Dummy (no data required).

Output: CY = 0.

Registers: All.

**BLTMD** (01A5H / SubROM)

Function: Transfer data from disk to Main RAM.

Input: HL – Must contain the value F562H.

FNPTR – (F562H, 2) – Address of the filename.

DUMMY – (F564H, 2) – Dummy (no data required).

SPTR – (F566H, 2) – Initial data address

EPTR – (F568H, 2) – Final data address

Output: CY = 0

Registers: All.

### **BLTDM** (01A9H / SubROM)

Function: Transfer data from Main RAM to disk.

Input: HL – Must contain the value F562H.

SPTR – (F562H, 2) – Initial data address.

EPTR – (F564H, 2) – Final data address.

FNPTR – (F566H, 2) – Address of the filename.

Output: CY = 0

Registers: All.

## **8.3 – MATH-PACK ROUTINES**

### **8.3.1 – Floating point mathematical functions**

|               |       |                 |                      |
|---------------|-------|-----------------|----------------------|
| <b>DECSUB</b> | 268CH | DAC – DAC – ARG | } (double precision) |
| <b>DECADD</b> | 269AH | DAC – DAC + ARG |                      |
| <b>DECMUL</b> | 27E6H | DAC – DAC * ARG |                      |
| <b>DECDIV</b> | 289FH | DAC – DAC / ARG |                      |
| <b>COS</b>    | 2993H | DAC – COS (DAC) |                      |
| <b>SIN</b>    | 29ACH | DAC – SIN (DAC) |                      |
| <b>TAN</b>    | 29FBH | DAC – TAN (DAC) |                      |
| <b>ATN</b>    | 2A14H | DAC – ATN (DAC) |                      |
| <b>LOG</b>    | 2A72H | DAC – LOG (DAC) | } (single-precision) |
| <b>SQR</b>    | 2AFFH | DAC – SQR (DAC) |                      |
| <b>EXP</b>    | 2B4AH | DAC – EXP (DAC) | } (double-precision) |
| <b>SGNEXP</b> | 37C8H | DAC – DAC ^ ARG |                      |
| <b>DBLEXP</b> | 37D7H | DAC – DAC ^ ARG |                      |

### **8.3.2 – Operations with integer numbers**

|               |       |                |                           |
|---------------|-------|----------------|---------------------------|
| <b>UMULT</b>  | 314AH | DE – BC * DE   | (Unsigned multiplication) |
| <b>ISUB</b>   | 3167H | HL – DE – HL   |                           |
| <b>IADD</b>   | 3172H | HL – DE + HL   |                           |
| <b>IMULT</b>  | 3193H | HL – DE * HL   |                           |
| <b>IDIV</b>   | 31E6H | HL – DE / HL   |                           |
| <b>IMOD</b>   | 323AH | HL – DE mod HL |                           |
|               |       | DE – DE / HL   |                           |
| <b>INTEXP</b> | 383FH | DAC – DE ^ HL  |                           |

### 8.3.3 – Special functions

|  |       |  |
|--|-------|--|
| <b>DECNRM</b>  | 26FAH | Normalises DAC, removing excessive zeros from the mantissa. (Ex. 0.00123 → 0.123E-2).  |
| <b>DECROU</b>  | 273CH | Rounds DAC   |
| <b>RND</b>   | 2BDFH | Generates a random number from the number contained in DAC, returning it in DAC.       |
| <b>SIGN</b>  | 2E71H | A – Sign of DAC.   |
| <b>ABSFN</b>   | 2E82H | Extracts the absolute value (module) of DAC and stores the result in DAC.              |
| <b>NEG</b>   | 2E8DH | Inverts the DAC signal.  |
| <b>SGN</b>   | 2E97H | DAC – Sign of DAC:<br>DAC +2, +3: 0000H = Zero<br>0001H = Positive<br>FFFFH = Negative |
| <b>FCOMP</b>   | 2F21H | Left: CBED    Right: DAC    single precision   |
| <b>ICOMP</b>   | 2F4DH | Left: DE       Right: HL       integer number  |
| <b>XDCOMP</b>  | 2F5CH | Left: ARG      Right: DAC      double precision  |
| Results will be in A register. Meanings of A register are: |       |  |
| A = 1  | →     | left < right   |
| A = 0  | →     | left = Right   |
| A = -1   | →     | left > right   |

### 8.3.4 – Movement

|               |       |             |                    |
|---------------|-------|-------------|--------------------|
| <b>MAF</b>    | 2C4DH | ARG ← DAC   | } Double precision |
| <b>MAM</b>    | 2C50H | ARG ← (HL)  |                    |
| <b>MOV8DH</b> | 2C53H | (DE) ← (HL) |                    |
| <b>MFA</b>    | 2C59H | DAC ← ARG   |                    |
| <b>MFM</b>    | 2C5CH | DAC ← (HL)  |                    |
| <b>MMF</b>    | 2C67H | (HL) ← DAC  |                    |
| <b>MOV8HD</b> | 2C6AH | (HL) ← (DE) |                    |
| <b>XTF</b>    | 2C6FH | (SP) ↔ DAC  |                    |
| <b>PHA</b>    | 2CC7H | ARG ← (SP)  |                    |
| <b>PHF</b>    | 2CCCH | DAC ← (SP)  |                    |
| <b>PPA</b>    | 2CDCH | (SP) ← ARG  |                    |
| <b>PPF</b>    | 2CE1H | (SP) ← DAC  |                    |



|               |       |                          |                    |
|---------------|-------|--------------------------|--------------------|
| <b>PUSHF</b>  | 2EB1H | DAC $\leftarrow$ (SP)    | } Single precision |
| <b>MOVFM</b>  | 2EBEH | DAC $\leftarrow$ (HL)    |                    |
| <b>MOVFR</b>  | 2EC1H | DAC $\leftarrow$ (CBED)  |                    |
| <b>MOVRF</b>  | 2ECCH | (CBED) $\leftarrow$ DAC  |                    |
| <b>MOVRMI</b> | 2ED6H | (CBED) $\leftarrow$ (HL) |                    |
| <b>MOVRM</b>  | 2EDFH | (BCDE) $\leftarrow$ (HL) |                    |
| <b>MOVMF</b>  | 2EE8H | (HL) $\leftarrow$ DAC    | } VALTYP           |
| <b>MOVE</b>   | 2EEBH | (HL) $\leftarrow$ (DE)   |                    |
| <b>VMOVAM</b> | 2EEFH | ARG $\leftarrow$ (HL)    |                    |
| <b>MOVVFM</b> | 2EF2H | (DE) $\leftarrow$ (HL)   |                    |
| <b>VMOVE</b>  | 2EF3H | (HL) $\leftarrow$ (DE)   |                    |
| <b>VMOVFA</b> | 2F05H | DAC $\leftarrow$ ARG     |                    |
| <b>VMOVFM</b> | 2F08H | DAC $\leftarrow$ (HL)    |                    |
| <b>VMOVAF</b> | 2F0DH | ARG $\leftarrow$ DAC     |                    |
| <b>VMOVMF</b> | 2F10H | (HL) $\leftarrow$ DAC    |                    |

### 8.3.5 – Conversions

|               |       |   |
|---------------|-------|---|
| <b>FRCINT</b> | 2F8AH | Converts DAC to a 2-byte integer (DAC+2,+3)   |
| <b>FRCSNG</b> | 2FB2H | Converts DAC to a single precision real number  |
| <b>FRCDBL</b> | 303AH | Converts DAC to a double precision real number  |
| <b>FIXER</b>  | 30BEH | DAC $\leftarrow$ SGN(DAC) * INT(ABS(DAC))   |
| <b>FIN</b>    | 3299H | Stores a string representing the floating-point number in DAC, converting it in real. |
| Input:        |       | HL $\leftarrow$ Starting address of the string  |
|               |       | A $\leftarrow$ First character of the string  |
| Output:       |       | DAC $\leftarrow$ Real number  |
|               |       | C $\leftarrow$ FFH: without decimal point<br>00H: with decimal point                  |
|               |       | B $\leftarrow$ Number of digits after the decimal point                               |
|               |       | D $\leftarrow$ Number of digits   |
| <b>FOUT</b>   | 3425H | Converts a real number contained in DAC to an unformatted string.                     |
| Input:        |       | A – Always 0  |
|               |       | B – Number of digits before the decimal point   |
|               |       | C – Number of digits after the decimal point, including this one.                     |
| Output:       |       | HL – Address of the first character of the string.                                    |

|               |  |  |
|---------------|--|--|
| <b>PUFOUT</b> | (3426H)  | Converts a real number contained in DAC to a formatted string.                     |
| Input:        | A ← Format:<br>bit 7 – 0: unformatted    1: formatted<br>bit 6 – 0: no commas    1: commas every 3 digits.<br>bit 5 – 0: meaningless    1: fill spaces with “*”<br>bit 4 – 0: meaningless    1: add “\$” before number<br>bit 3 – 0: meaningless    1: add “+” for positive numbers<br>bit 2 – 0: meaningless    1: sign after the number<br>bit 1 – 0: not used<br>bit 0 – 0: fixed point    1: floating point<br>B ← Number of digits before decimal point.<br>C ← Number of digits after decimal point, including this one. |  |
| Output:       | HL ← Starting address of the string.   |  |
| <b>FOUTB</b>  | (371AH)  | Converts an integer contained in DAC to a string expression in binary format.      |
| Input:        | DAC+2, +3 – Integer number.<br>VALTYP – 2.   |  |
| Output:       | HL – Initial address of the binary string.   |  |
| <b>FOUTO</b>  | (371EH)  | Converts an integer contained in DAC to a string expression in octal format.       |
| Input:        | DAC+2, +3 – Integer number.<br>VALTYP – 2.   |  |
| Output:       | HL – Initial address of the octal string.  |  |
| <b>FOUTH</b>  | (3722H)  | Converts an integer contained in DAC to a string expression in hexadecimal format. |
| Input:        | DAC+2, +3 – Integer number.<br>VALTYP – 2.   |  |
| Output:       | HL – Initial address of the hexadecimal string.  |  |
| <b>FIN</b>    | 3299H  | Converts a string representing a real number to BCD format and stores it in DAC.   |
| Input:        | HL – Address of the first character of the string.<br>A – First character of the string.   |  |

Output: DAC – Real number in BCD.  
 C  $\leftarrow$  FFH – Without decimal point;  
 0 – With decimal point.  
 B – Number of digits after the decimal point.  
 D – Total number of digits.

## 8.4 – BASIC INTERPRETER ROUTINES

### 8.4.1 – Execution routines

#### **READYR** (409BH / Main)

Function: Returns to the command level (BASIC hot start).

Input: None

Output: None

#### **CRUNCH** (42B2H / Main)

Function: Converts BASIC text from ASCII form to tokenized form.

Input: HL  $\leftarrow$  ASCII text address to be converted, ending with a 00H byte.

Output: KBUF (F41FH) – Converted BASIC text.

#### **NEWSTT** (4601H / Main)

Function: Executes a BASIC text. The text must be in tokenized form.

Input: HL  $\leftarrow$  pointer to the beginning of the text to be executed.  
 The text must be in the form illustrated below:

|     |     |     |     |
|-----|-----|-----|-----|
| 3AH | 94H | 00H | ... |
| :   | NEW |     | ... |

↑

(HL)

Output: None

#### **CHRGTR** (4666H / Main) – From 0010H

Function: Extracts a character from the BASIC text, starting with (HL) + 1. Spaces are ignored.

Input: HL  $\leftarrow$  starting address of the text

Output: HL  $\leftarrow$  extracted character address

A  $\leftarrow$  ASCII code of the extracted character

Z = "1" if it is the end of the line (00H or 3AH ":")

CY = "1" if it is a character from 0 to 9

**FRMEVL** (4C64H / Main)

Function: Evaluates an expression and returns the result.

Input: HL ← start address of the expression in the BASIC text.

Output: HL ← final address of the expression +1.

VALTYP (F663H) ← 2 – Integer variable

4 – Single precision variable

8 – Double precision variable

3 – String variable

DAC (F7F6H) – Result of the evaluated expression.

**GETBYT** (521CH / Main)

Function: Evaluates an expression and returns a 1-byte result. When the result extrapolates the value of 1 byte, an “Illegal Function Call” error will be generated and the execution will return to the command level.

Input: HL ← starting address of the expression to be evaluated

Output: HL ← final address of the expression +1.

A,E – Evaluation result (A and E contain the same value).

**FRMQNT** (542FH / Main)

Function: Evaluates an expression and returns a result of 2 bytes (integer). When the result extrapolates the value of 2 bytes, an “Overflow” error will be generated and the execution will return to the command level.

Input: HL ← starting address of the expression to be evaluated

Output: HL ← final address of the expression +1.

DE ← evaluation result

**SYNCHR** (558CH / Main) – 0008H

Function: Tests if the character pointed by (HL) is the one specified. If not, it generates “Syntax error”; otherwise it calls CHRGET (4666H / Main).

Input: HL ← points to the character to be tested

The character for comparison must be placed after an instruction “RST 0008H” in the form of a line parameter, as shown in the example below:

```
LD HL, CHAR
```

```
RST 008H
```

```
DEFB 'A'
```

```
|
```

```
CHAR: DEFB 'B'
```

Output: HL is incremented by one and A receives (HL). When the tested character is numeric, the CY flag is set. The end of the declaration (00H or 3AH “:”) sets the Z flag.

### **GETYPR** (5597H / Main) – 0028H

Function: Gets the type of operand contained in DAC.

Input: None

Output: Flags CY, S, Z and P / V, as shown in the table below:

|                   |       |       |       |         |
|-------------------|-------|-------|-------|---------|
| Integer:          | C=1   | S=1 * | Z=0   | P/V=1   |
| Simple precision: | C=1   | S=0   | Z=0   | P/V=0 * |
| Double precision: | C=0 * | S=0   | Z=0   | P/V=1   |
| String:           | C=1   | S=0   | Z=1 * | P/V=1   |

Note: The types can be recognized verifying only the flags marked with “\*”.

### **PTRGET** (5EA4H / Main)

Function: Gets the address for storing a variable or matrix. The address is also obtained when the variable has not been assigned. When the value of SUBFLG (F5A5H) is different from 0, the starting address of an array will be obtained; otherwise, the address of the array element will be obtained.

Input: HL ← starting address of the variable name in BASIC text  
SUBFLG (F6A5H) – 0: single variable, other value: matrix

Output: HL ← address after the variable name  
DE ← address of the content of the variable.

### **FRESTR** (67D0H / Main)

Function: Registers the result of a string obtained by FRMEVL (4C64H) and obtains the respective descriptor. When evaluating a string, this routine is usually combined with FRMEVL as described below:

```
CALL FRMEVL
PUSH HL
CALL FRESTR
EX DE, HL
POP HL
LD A, (DE)
```

Input: VALTYP (F663H) – Variable type (must be 3).  
DAC (F7F6H) – Pointer to the string descriptor.

Output: HL ← pointer to the string descriptor.

## 8.4.2 – Command and function routines

Command / Token Address Token Address

function function in the routine table

| Command/<br>Function | Token | Function<br>Token | Table<br>Address | Routine<br>Address |
|----------------------|-------|-------------------|------------------|--------------------|
| >                    | EEH   | –                 | Afat             | –                  |
| =                    | EFH   | –                 | Afat             | –                  |
| <                    | F0H   | –                 | Afat             | –                  |
| +                    | F1H   | –                 | Afat             | –                  |
| -                    | F2H   | –                 | Afat             | –                  |
| *                    | F3H   | –                 | Afat             | –                  |
| /                    | F4H   | –                 | Afat             | –                  |
| ^                    | F5H   | –                 | Afat             | –                  |
| \$                   | FCH   | –                 | Afat             | –                  |
| ABS                  | 06H   | FF86H             | 39E8H            | 2E82H              |
| AND                  | F6H   | –                 | Afat             | –                  |
| ASC                  | 15H   | FF95H             | 3A06H            | 680BH              |
| ATN                  | 0EH   | FF8EH             | 39F8H            | 2A14H              |
| ATTR\$               | E9H   | –                 | Afat             | 7C43H              |
| AUTO                 | A9H   | –                 | 3973H            | 49B5H              |
| BASE                 | C9H   | –                 | 39BEH            | 7B5AH              |
| BEEP                 | C0H   | –                 | 39ACH            | 00C0H              |
| BIN\$                | 1DH   | FF9DH             | 3A16H            | 6FFFH              |
| BLOAD                | CFH   | –                 | 39CAH            | 6EC6H              |
| BSAVE                | D0H   | –                 | 39CCH            | 6E92H              |
| CALL                 | CAH   | –                 | 39C0H            | 55A8H              |
| CDBL                 | 20H   | FFA0H             | 3A1CH            | 303AH              |
| CHR\$                | 16H   | FF96H             | 3A08H            | 681BH              |
| CINT                 | 1EH   | FF9EH             | 3A18H            | 2F8AH              |
| CIRCLE               | BCH   | –                 | 39A4H            | 5B11H              |
| CLEAR                | 92H   | –                 | 3950H            | 64AFH              |
| CLOAD                | 9BH   | –                 | 3962H            | 703FH              |
| CLOSE                | B4H   | –                 | 3994H            | 6C14H              |
| CLS                  | 9FH   | –                 | 396AH            | 00C3H              |
| CMD                  | D7H   | –                 | 39DAH            | 7C34H              |
| COLOR                | BDH   | –                 | 39A6H            | 7980H              |
| CONT                 | 99H   | –                 | 395EH            | 6424H              |
| COPY                 | D6H   | –                 | 39D8H            | 7C2FH              |

|        |     |       |       |       |
|--------|-----|-------|-------|-------|
| COS    | 0CH | FF8CH | 39F4H | 2993H |
| CSAVE  | 9AH | -     | 3960H | 6FB7H |
| CSNG   | 1FH | FF9FH | 3A1AH | 2FB2H |
| CSRLIN | E8H | -     | Afat  | 790AH |
| CVD    | 2AH | FFAAH | 3A30H | 7C70H |
| CVI    | 28H | FFA8H | 3A2CH | 7C66H |
| CVS    | 29H | FFA9H | 3A2EH | 7C6BH |
| DATA   | 84H | -     | 3934H | 485BH |
| DEF    | 97H | -     | 395AH | 501DH |
| DEFDBL | AEH | -     | 3988H | 4721H |
| DEFINT | ACH | -     | 3984H | 471BH |
| DEFSNG | ADH | -     | 3986H | 471EH |
| DEFSTR | ABH | -     | 3982H | 4718H |
| DELETE | A8H | -     | 397CH | 53E2H |
| DIM    | 86H | -     | 3938H | 5E9FH |
| DRAW   | BEH | -     | 39A8H | 5D6EH |
| DSKF   | 26H | FFA6H | 3A28H | 7C39H |
| DSKI\$ | EAH | -     | Afat  | 7C3EH |
| DSK0\$ | D1H | -     | 39CEH | 7C16H |
| ELSE   | A1H | 3AA1H | 396EH | 485DH |
| END    | 81H | -     | 396EH | 63EAH |
| EOF    | 2BH | FFABH | 3A32H | 6D25H |
| EQV    | F9H | -     | Afat  | -     |
| ERASE  | A5H | -     | 3976H | 6477H |
| ERL    | E1H | -     | Afat  | 4E0BH |
| ERR    | E2H | -     | Afat  | 4DFDH |
| ERROR  | A6H | -     | 3978H | 49AAH |
| EXP    | 0BH | FF8BH | 39F2H | 2B4AH |
| FIELD  | B1H | -     | 398EH | 7C52H |
| FILES  | B7H | -     | 39AAH | 6C2FH |
| FIX    | 21H | FFA1H | 3A1EH | 30BEH |
| FN     | DEH | -     | Afat  | 5040H |
| FOR    | 82H | -     | 3920H | 4524H |
| FP0\$  | 27H | FFA7H | 3A2AH | 6D39H |
| FRE    | 0FH | FF8FH | 39FAH | 69F2H |
| GET    | B2H | -     | 3990H | 775BH |
| GOSUB  | 8DH | -     | 3948H | 47B2H |
| GOTO   | 89H | -     | 393EH | 47E8H |
| GO TO  | 89H | -     | 393EH | 47E8H |
| HEX\$  | 1BH | FF9BH | 3A12H | 65FAH |
| IF     | 8BH | -     | 3942H | 49E5H |
| IMP    | FAH | -     | 3A20H | 7940H |

|         |     |       |       |       |
|---------|-----|-------|-------|-------|
| INKEY\$ | ECH | -     | Afat  | 7347H |
| INP     | 10H | FF90H | 39FCH | 4001H |
| INPUT   | 85H | -     | 3936H | 4B6CH |
| INSTR   | E5H | -     | 39F6H | 29FBH |
| INT     | 05H | FF85H | 39E6H | 30CFH |
| IPL     | D5H | -     | 39D6H | 7C2AH |
| KEY     | CCH | -     | 3964H | 786CH |
| KILL    | D4H | -     | 39D4H | 7C25H |
| LEFT\$  | 01H | FF81H | 39DEH | 6861H |
| LEN     | 12H | FF92H | 3A00H | 67FFH |
| LET     | 88H | -     | 393CH | 4880H |
| LFILES  | BBH | -     | 39A2H | 6C2AH |
| LINE    | AFH | -     | 398AH | 4B0EH |
| LIST    | 93H | -     | 3952H | 522EH |
| LLIST   | 9EH | -     | 3968H | 5229H |
| LOAD    | B5H | -     | 3996H | 6B5DH |
| LOC     | 2CH | FFACH | 3A34H | 6D03H |
| LOCATE  | D8H | -     | 39DCH | 7766H |
| LOF     | 2DH | FFADH | 3A36H | 6D14H |
| LOG     | 0AH | FF8AH | 39F0H | 2A72H |
| LPOS    | 1CH | FF9CH | 3A14H | 4FC7H |
| LPRINT  | 9DH | -     | 394CH | 4A1DH |
| LSET    | B8H | -     | 399CH | 7C48H |
| MAX     | CDH | -     | 39C6H | 7E4BH |
| MERGE   | B6H | -     | 3998H | 6B5EH |
| MID\$   | 03H | FF83H | 39E2H | 689AH |
| MKD\$   | 30H | FFB0H | 3A3CH | 7C61H |
| MKI\$   | 2EH | FFAEH | 3A38H | 7C57H |
| MKS\$   | 2FH | FFAFH | 3A3AH | 7C5CH |
| MOD     | FBH | -     | Afat  | -     |
| MOTOR   | CEH | -     | 39C8H | 73B7H |
| NAME    | D3H | -     | 39D2H | 7C20H |
| NEW     | 94H | -     | 3954H | 6286H |
| NEXT    | 83H | -     | 3932H | 6527H |
| NOT     | E0H | -     | Afat  | -     |
| OCT\$   | 1AH | FF9AH | 3A10H | 7C70H |
| OFF     | EBH | -     | 3A02H | 3A02H |
| ON      | 95H | -     | 3956H | 48E4H |
| OPEN    | B0H | -     | 398CH | 6AB7H |
| OR      | F7H | -     | Afat  | -     |
| OUT     | 9CH | -     | 3964H | 4016H |
| PAD     | 25H | FFA5H | 3A26H | 7969H |



|          |     |       |       |       |
|----------|-----|-------|-------|-------|
| PAINT    | BFH | -     | 39AAH | 59C5H |
| PDL      | 24H | FFA4H | 3A24H | 795AH |
| PEEK     | 17H | FF97H | 3A0AH | 541CH |
| PLAY     | C1H | -     | 39AEH | 73E5H |
| POINT    | EDH | -     | Afat  | 5803H |
| POKE     | 98H | -     | 395CH | 5423H |
| POS      | 11H | FF91H | 39FEH | 4FCCH |
| PRESET   | C3H | -     | 39B2H | 57E5H |
| PRINT    | 91H | -     | 394EH | 4A24H |
| PSET     | C2H | -     | 39B0H | 57EAH |
| PUT      | B3H | -     | 3992H | 7758H |
| READ     | 87H | -     | 393AH | 4B9FH |
| REM      | 8FH | 3A8FH | 394AH | 485DH |
| RENUM    | AAH | -     | 3980H | 5468H |
| RESTORE  | 8CH | -     | 3944H | 63C9H |
| RESUME   | A7H | -     | 397AH | 495DH |
| RETURN   | 8EH | -     | 3948H | 4821H |
| RIGHT\$  | 02H | FF82H | 39E0H | 6891H |
| RND      | 08H | FF88H | 39ECH | 2BDFH |
| RSET     | B9H | -     | 399EH | 7C4DH |
| RUN      | 8AH | -     | 3940H | 479EH |
| SAVE     | BAH | -     | 39A0H | 6BA3H |
| SCREEN   | C5H | -     | 39B6H | 79CCH |
| SET      | D2H | -     | 39D0H | 7C1BH |
| SGN      | 04H | FF84H | 39E4H | 2E97H |
| SIN      | 09H | FF89H | 39EEH | 29ACH |
| SOUND    | C4H | -     | 39B4H | 73CAH |
| SPACE\$  | 19H | FF99H | 3A0EH | 6848H |
| SPC(     | DFH | -     | Afat  | -     |
| SPRITE   | C7H | -     | 39BAH | 7A48H |
| SQR      | 07H | FF87H | 39EAH | 2AFFH |
| STEP     | DCH | -     | Afat  | -     |
| STICK    | 22H | FFA2H | 3A20H | 7940H |
| STOP     | 90H | -     | 394CH | 63E3H |
| STR\$    | 13H | FF93H | 3A02H | 6604H |
| STRIG    | 23H | FFA3H | 3A22H | 794CH |
| STRING\$ | E3H | -     | Afat  | 6829H |
| SWAP     | A4H | -     | 3974H | 643EH |
| TAB(     | DBH | -     | Afat  | -     |
| TAN      | 0DH | FF8DH | 39F6H | 29FBH |
| THEN     | DAH | -     | Afat  | -     |
| TIME     | CBH | -     | 39C2H | 7911H |

|        |     |       |       |       |
|--------|-----|-------|-------|-------|
| TO     | D9H | -     | Afat  | -     |
| TROFF  | A3H | -     | 3972H | 6439H |
| TRON   | A2H | -     | 3970H | 6438H |
| USING  | E4H | -     | Afat  | -     |
| USR    | DDH | -     | Afat  | 4FD5H |
| VAL    | 14H | FF94H | 3A04H | 68BBH |
| VARPTR | E7H | -     | 39FAH | 4E41H |
| VDP    | C8H | -     | 39BCH | 7B37H |
| VPEEK  | 18H | FF98H | 3A0CH | 7BF5H |
| VPOKE  | C6H | -     | 39B8H | 7BE2H |
| WAIT   | 96H | -     | 3958H | 401CH |
| WIDTH  | A0H | -     | 396CH | 51C9H |
| XOR    | F8H | -     | Afat  | -     |

## 8.5 – EXTENDED BIOS ROUTINES

### 8.5.1 – Extended BIOS Entry

#### EXTBIO (FFCAH/Work Area)

Function: Accesses extended BIOS functions. Only available if bit 0 of the HOKVLD system flag (FB20H) is set to 1.

Input: A ← Always 00H.

D ← Device ID:

00 – Internal commands (broadcast commands)

01~03 – Free

04 – DOS2 Mapped Memory Handling

05~07 – Free

08 – RS232C / MSX Modem

09 – Free

10 – MSX-Audio

11 – MSX MIDI

12~15 – Free

16 – MSX-JE

17 – Kanji Driver

18~33 – Free

34 – UNAPI

35~51 – Free

52 – MWMPLAY (MoonBlaster 4 Wave Replayer)

53~76 – Free

77 – Memman

78 – Nowind

79~204 – Free

205 – MCDRV (Micro Cabin BGM Replayer)

206~239 – Free

240 – MGSDRV (SCC music player)

241~254 – Free

255 – System Exclusive

E ← Function number (0 to 255).

Output: Depends of the device and function called.

CY = 1 if the specified device is not found.

Registers: All.

### 8.5.2 – Internal commands (broadcast commands)

#### EXTBIO (FFCAH/Work Area)

Function: Accesses extended BIOS functions.

Input: A – 00H.

D – 00H – Internal command.

E – 00H – Examines the devices present in the system, asks it to record its own number in the table, increments the pointer by 1 and moves to the next device.

B – ID of the slot where the table will be placed.

HL – Table address.

Output: B – Table slot ID.

HL – Table address.

CY = 1 if there are no devices.

Registers: All.

#### EXTBIO (FFCAH/Work Area)

Function: Accesses extended BIOS functions.

Input: A = 00H.

D = 00H ← Internal command.

E = 01H – Gets the number of MSX BASIC interrupt events. It internally manages up to 26 events, which are:

0~9 ON KEY GOSUB

10 ON STOP GOSUB

|       |                           |
|-------|---------------------------|
| 11    | ON SPITE GOSUB            |
| 12~16 | ON STRIG GOSUB            |
| 17    | ON INTERVAL GOSUB         |
| 18~23 | For expansion devices     |
| 24~25 | Reserved (prohibited use) |

Output: A – Number of active events.

Registers: All.

#### **EXTBIO** (FFCAH/Work Area)

Function: Accesses extended BIOS functions.

Input: A – 00H.

D – 00H ← Internal command.

E – 02H ← Declare interrupt prohibition (disables interrupts for the default time of 1 mS).

Output: None.

Registers: All.

#### **EXTBIO** (FFCAH/Work Area)

Function: Accesses extended BIOS functions.

Input: A – 00H.

D – 00H ← Internal command.

E – 03H ← Declare interrupt permission (enables interrupts blocked by the 02H function).

Output: None.

Registers: All.

### **8.5.3 –Memory Mapper**

#### **EXTBIO** (FFCAH/Work Area)

Function: Access extended BIOS functions

Input: A – 00H.

D – 04H ← MSXDOS2 Memory Mapped Handling Device.

E – 01H ← Returns the address of the Memory Mapper variable table.

Output: A – Primary mapper slot ID.

DE – Reserved.

HL – Starting address of the variable table, whose structure is as follows:

- +00H Primary Mapper Slot ID
- +01H Total number of 16K segments
- +02H Number of free 16K segments
- +03H Number of 16k segments allocated by the system (minimum 6 for primary mapper)
- +04H Number of 16K segments allocated to user
- +05H~+07H Reserved (Always 00H)
- +08H... Entries for other mappers in other slots. If there is none, it will contain 00H.

Registers: All.

### **EXTBIO** (FFCAH/Work Area)

Function: Access extended BIOS functions

Input: A – 00H.

D – 04H – Memory Mapper Handling Device.

E – 02H – Returns several parameters related to the Memory Mapper.

Output: A – Total number of segments (logical pages) for the primary mapper.

B – Primary mapper slot ID.

C – Number of free segments (logical pages) in the primary mapper.

DE – reserved.

HL – Start address of a mapper support subroutine call table. The format of this table is as follows:

+00H ALL\_SEG Allocates a 16K segment

+03H FRE\_SEG Releases a 16K segment

+06H RD\_SEG Read a byte from the address (A:HL) to A

+09H WR\_SEG Write the contents of E at the address (A:HL)

+0CH CAL-SEG Inter-segment call by address IYh:IX

+0FH CALLS Inter-segment call. Parameters in line after the CALL statement

+12H PUT\_PH Place a segment on the physical page (HL)

+15H GET\_PH Returns the current segment for the physical page (HL)

|             |   |
|-------------|---|
| +18H PUT_P0 | Place a segment on physical page 0                                      |
| +1BH GET_P0 | Returns the current page 0 segment.                                     |
| +1EH PUT_P1 | Place a segment on physical page 1                                      |
| +21H GET_P1 | Returns the current page 1 segment.                                     |
| +24H PUT_P2 | Place a segment on physical page 2                                      |
| +27H GET_P2 | Returns the current page 2 segment.                                     |
| +2AH PUT_P3 | Not supported as page 3 cannot be switched. If called, it just returns. |
| +2DH GET_P3 | Returns the current page 3 segment.                                     |

Registers: All.

### 8.5.3.1 – Memory Mapper Manipulation Routines

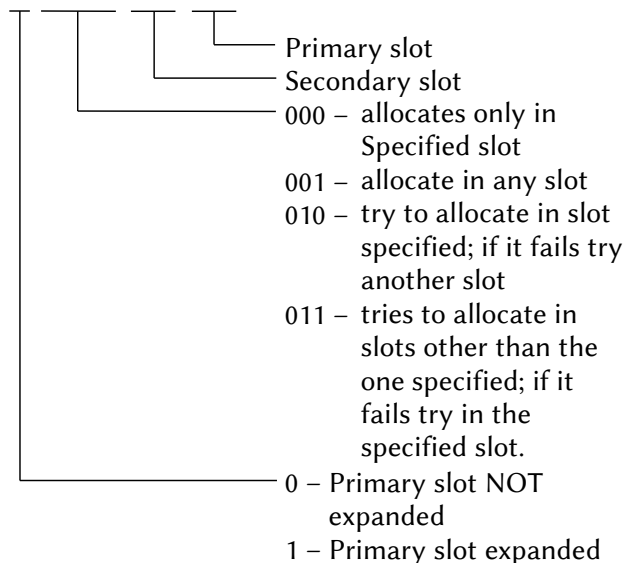
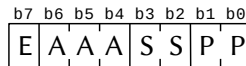
**ALL\_SEG** (HL+00H/ExtBIOS) – HL value obtained via EXTBIO

Function: Allocate a 16K segment of the mapper.

Input: A – 00H – Allocates a user segment

01H – Allocates a system segment

B – 00H – Allocates only to primary mapper



Output: CY = 1 → There are no free segments  
           0 → Segment allocated

A – Segment number

B – Segment slot ID

Note: A system segment will only be released using the FRE\_SEG routine. In the case of a user segment, whenever the program that uses it is closed, the segments are released, which is not the case with the system segments.

**FRE\_SEG** (HL+03H/ExtBIOS) – HL value obtained via EXTBIO

Function: Free a 16K segment from the mapper.

Input: A – segment number to be released

B – If it is 00H, it releases only on the primary mapper; if it is different from 00H it releases in any other mapper than the primary one (see ALL\_SEG).

Output: CY = 0 – Segment released  
           1 – Error in releasing the segment

**RD\_SEG** (HL+06H/ExtBIOS) – HL value obtained via EXTBIO

Function: Read a byte from the mapper.

Input: A – segment number from which the byte will be read.

HL – address to be read (0000H to 3FFFFH).

Output: A – byte read.  
         All other registers are preserved.

**WR\_SEG** (HL+09H/ExtBIOS) – HL value obtained via EXTBIO

Function: Write a byte to the mapper.

Input: A – segment number where the byte will be written.

HL – address to be written (0000H to 3FFFFH).

E – value to write.

Output: A – corrupted while writing.  
         All other registers are preserved.

**CAL\_SEG** (HL+0CH/ExtBIOS) – HL value obtained via EXTBIO

Function: Calls a routine in any area of the mapper.

Input: IYh – segment number to be called

IX – address to be called (0000H to FFFFFH)

AF, BC, DE and HL can contain parameters for the routine.

Do not use AF', BC', DE' and HL' as they are corrupted during the call

Output: AF, BC, DE, HL, IX and IY can contain valid return values.  
AF', BC', DE' and HL' return corrupted.

**CALLS** (HL+0FH/ExtBIOS) – HL value obtained via EXTBIO

Function: Calls a routine in any area of the mapper through inline parameters.

Input: AF, BC, DE and HL can contain parameters for the routine.  
Do not use AF', BC', DE' and HL' as they are corrupted during the call. The call string must be in the following format:

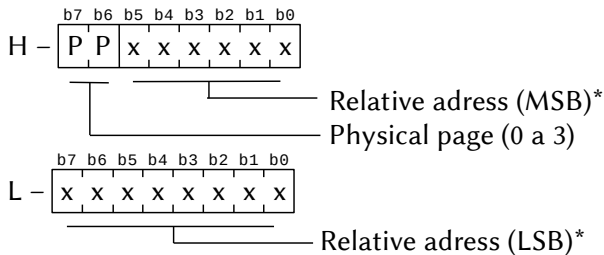
```
CALL CALLS
DEFB SEGMENT
DEFW ADDRESS
```

Output: AF, BC, DE, HL, IX and IY can contain valid return values.  
AF', BC', DE' and HL' return corrupted.

**PUT\_PH** (HL+12H/ExtBIOS) – HL value obtained via EXTBIO

Function: Enables a mapper segment on a physical page.

Input: A – Mapper segment

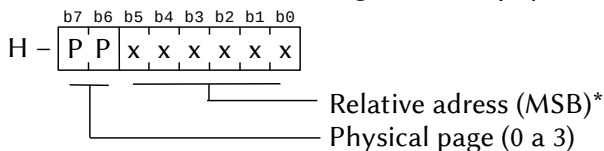


\* Relative address is optional.

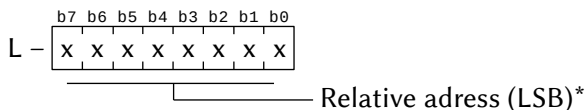
Output: None. All registers are preserved.

**GET\_PH** (HL+15H/ExtBIOS) – HL value obtained via EXTBIO

Function: Returns the current active segment on a physical page.







\* Relative address is optional.

Output: A – Segment number.  
All other registers are preserved.

**PUT\_P0** (HL+18H/ExtBIOS) – HL value obtained via EXTBIO

Function: Enables a mapper segment on physical page 0.

Input: A – segment number to be enabled

Output: None. All registers are preserved.

**GET\_P0** (HL+1BH/ExtBIOS) – HL value obtained via EXTBIO

Function: Returns the active segment on physical page 0.

Input: None

Output: A – active segment number  
All other registers are preserved.

**PUT\_P1** (HL+1EH/ExtBIOS) – HL value obtained via EXTBIO

Function: Enables a mapper segment on physical page 1.

Input: A – segment number to be enabled

Output: None. All registers are preserved.

**GET\_P1** (HL+21H/ExtBIOS) – HL value obtained via EXTBIO

Function: Returns the active segment on physical page 1.

Input: None

Output: A – active segment number  
All other registers are preserved.

**PUT\_P2** (HL+24H/ExtBIOS) – HL value obtained via EXTBIO

Function: Enables a mapper segment on physical page 2.

Input: A – segment number to be enabled

Output: None. All registers are preserved.

**GET\_P2** (HL+27H/ExtBIOS) – HL value obtained via EXTBIO

Function: Returns the active segment on physical page 2.

Input: None

Output: A – active segment number  
All other registers are preserved.

**PUT\_P3** (HL+2AH/ExtBIOS) – HL value obtained via EXTBIO

Function: Not supported since physical page 3 cannot be swapped.  
A call to this function has no effect.

**GET\_P3** (HL+2DH/ExtBIOS) – HL value obtained via EXTBIO

Function: Returns the active segment on physical page 0.

Input: None.

Output: A – Active segment number.  
All other registers are preserved.

**CALL\_MAP** (HL+30H/ExtBIOS) – HL value obtained via EXTBIO

Function: Calls a routine in any area of the mapped RAM.

Input: IYh – Slot number.

IYl – Segment number.

IX – Routine address, which must necessarily be on  
page 1 (4000H to 7FFFH).

AF, BC, DE, HL – Parameters for the routine. (Do not use  
AF', BC', DE' and HL' as they are corrupted in the call).

Output: AF, BC, DE, HL, IX, IY – May contain valid return values.  
AF', BC', DE' and HL' return corrupted.

Note: Exclusive routine for NEXTOR.

**RD\_MAP** (HL+33H/ExtBIOS) – HL value obtained via EXTBIO

Function: Reads a byte from a RAM segment.

Input: A – Slot number.

B – Segment number.

HL – Address to read (highest two bits will be ignored).

Output: A – Data byte read.  
F, BC, DE, HL, IX, IY return preserved.

Note: Exclusive routine for NEXTOR.

**CALL\_MAPI** (HL+36H/ExtBIOS) – HL value obtained via EXTBIO

Function: Calls routine on a mapped RAM segment (inline parameters).

Input: AF, BC, DE, HL – Parameters for the called routine. Do not  
use AF', BC', DE' and HL' as they are corrupted during the  
call. The call string must be in the following format:

```
CALL CALL_MAPI
```

```
DEFB SLOT
```

```
DEFB ADDRESS
```

```
DEFB SEGMENT NUMBER
```

```
; It is not necessary to use RET
```

Where

- SLOT – Is the slot to be called, from 0 to 3
- ADDRESS – Address to be called as an index of a table, which can vary from 0 to 63, where 0=4000H, 1=4003H, 2=4006H, etc.
- SEGMENT NUMBER – Can range from 0 to 255.

Output: AF, BC, DE, HL, IX, IY – Parameters returned by the routine.

Note: Exclusive routine for NEXTOR.

**WR\_MAP** (HL+39H/ExtBIOS) – HL value obtained via EXTBIOS

Function: Writes a byte to a mapped RAM segment.

Input: A – Slot number.

B – Segment number.

E – Byte to write.

HL – Address to be written (highest two bits are ignored).

Output: A – Data read from the specified address.

F, BC, DE, HL, IX, IY return preserved.

Note: Exclusive routine for NEXTOR.

## 8.5.4 – RS232C Serial Port and MSX Modem

**EXTBIO** (FFCAH/Work Area)

Function: Access extended BIOS functions

Input: A – 00H.

D – 08H – RS232C manipulation device.

E – 00H – Returns the address of the input address table of the RS232C routines.

B – Address table slot ID.

HL – Table address.

Output: CY = 1 → no RS232C interfaces.

0 → HL is incremented by 4 for each interface found and will point to the end of a table that reserves 4 bytes for each RS232C found. The original value of HL points to the beginning of the table, which has the following structure:

+00H Slot ID.

+01H Lowest address.

+02H Highest address.

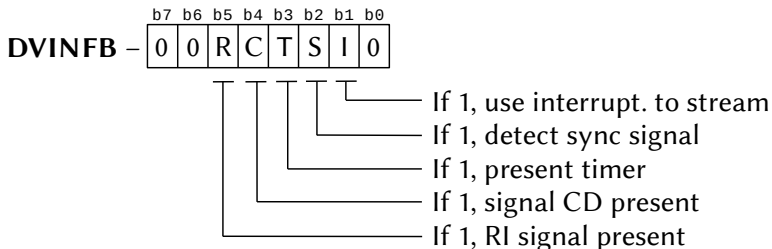
+03H Reserved for expansion.

The slot ID (+00H) and the address (+01H,+02H) will point to a table with the following structure:

|      |           |                        |
|------|-----------|------------------------|
| +00H | DB DVINFB | (optional)             |
| +01H | DB DVTYPE | (optional)             |
| +02H | DB 0      |                        |
| +03H | JP INIT   | Initialize RS232       |
| +06H | JP OPEN   | Opens an RS232 port    |
| +09H | JP STAT   | Returns various states |
| +0CH | JP GETCHR | Read a character       |
| +0FH | JP SNDCHR | Sends a character      |
| +12H | JP CLOSE  | Closes an RS232 port   |
| +15H | JP EOF    | Checks end of file     |
| +18H | JP LOC    | Returns the num. char. |
| +1BH | JP LOF    | Return free space      |
| +1EH | JP BACKUP | Save a character       |
| +21H | JP SNDBRK | Send break characters  |
| +24H | JP DTR    | On/off DTR line        |
| +27H | JP SETCHN | Select RS232 channel   |
| +2AH | JP NCUSTA | (MSX Modem)            |
| +2DH | JP SPKCNT | (MSX Modem)            |
| +30H | JP LINSEL | (MSX Modem)            |
| +33H | JP DIALST | (MSX Modem)            |
| +36H | JP DIALCH | (MSX Modem)            |
| +39H | JP DTMFST | (MSX Modem)            |
| +3CH | JP RDDTMF | (MSX Modem)            |
| +3FH | JP HOKCNT | (MSX Modem)            |
| +42H | JP CONFIG | (MSX Modem)            |
| +45H | JP SPCIAL | (MSX Modem)            |

Registers: All.

#### 8.5.4.1 – Parameter Bytes



**DVTYPE** – 0 → Multiple channels.  
 Another value → Single channel.

#### 8.5.4.2 – RS232C serial port manipulation routines

**INIT** (HL+03H/ExtBIOS) – HL value obtained via EXTBIO

Function: Initialize RS232C port.

Input: B – ID of the slot from the parameter table.

HL – Address of the parameter table, with the following structure (from +00H to +07H values must be in ASCII code):

+00H – Character length ("5", "6", "7" or "8")

+01H – Parity ("E", "O", "I" or "N")

+02H – Stop bits ("1", "2" or "3")

+03H – XON/XOFF ("X" or "N")

+04H – CTR-RTS hand shake ("H" or "N")

+05H – Auto LF reception ("A" or "N")

+06H – Auto LF transmission ("A" or "N")

+07H – SI/SO Control ("Y" or "N")

+08H – Receive speed (low)

+09H – Receive speed (high) (50 to 19 200 baud)

+0AH – Speed transmission (low)

+0BH – Speed transmission (high) (50 to 19 200 baud)

+0CH – Time counter (0 to 255)

Output: CY = 0 → RS232C successfully started.

1 → Parameter error.

Registers: AF.

**OPEN** (HL+06H/ExtBIOS) – HL value obtained via EXTBIO

Function: Opens an RS232C serial port using FCB.

Input: HL – FCB initial address (greater than 8000H).

C – Buffer size (32 to 254).

E – Open mode:

0 – Entry

2 – Exit

4 – Input/Output and RAW mode

Output: CY = 0 → Door opened successfully.

1 → Error in the opening process.

Registers: AF.

**STAT** (HL+09H/ExtBIOS) – HL value obtained via EXTBIOS

Function: Returns status or error data.

Input: None.

Output: HL – Data returned.

Bit 15: 0 – No buffer error. 1 – Buffer overflow.

Bit 14: 0 – No timing error. 1 – Time out.

Bit 13: 0 – Correct framing. 1 – Framing error.

Bit 12: 0 – Correct execution.

1 – Execution error (overrun error).

Bit 11: 0 – No parity error.

1 – Character parity error.

Bit 10: 0 – CTRL+STOP are not pressed.

1 – CTRL+STOP pressed together.

Bit 09: Reserved.

Bit 08: Reserved.

Bit 07: 0 – Clear to Send state is false.

1 – Clear to Send state is true.

Bit 06: 0 – Timer/Counter-2 not confirmed.

1 – Timer/Counter-2 confirmed.

Bit 05: Reserved.

Bit 04: Reserved.

Bit 03: 0 – Data Set Ready state is false.

1 – Data Set Ready state is true.

Bit 02: 0 – Stop not detected.

1 – Stop detected.

Bit 01: 0 – Ring indicator state is false.

1 – Touch indicator state is true.

Bit 00: 0 – Carrier not detected.

1 – Carrier detected.

**GETCHR** (HL+0CH/ExtBIOS) – HL value obtained via EXTBIOS

Function: Returns a character from the receive buffer.

Input: None.

Output: A – Character received.

CY = 1 → EOF (end of file).

S = 1 → Error.

Registers: F.

**SNDCHR** (HL+0FH/ExtBIOS) – HL value obtained via EXTBIO

Function: Sends a character to the RS232C serial port.

Input: A – Character to be sent.

Output: CY = 1 → CTRL+STOP were pressed together.

Z = 1 → Error.

Registers: F.

**CLOSE** (HL+12H/ExtBIOS) – HL value obtained via EXTBIO

Function: Close the RS232C serial port.

Input: None.

Output: CY = 1 → Error.

Registers: AF.

**EOF** (HL+15H/ExtBIOS) – HL value obtained via EXTBIO

Function: Checks for end of file.

Input: None.

Output: HL = -1 and CY = 1 → Next character is EOF (End of file).

HL = 0 and CY = 0 → Not end of file.

Registers: AF.

**LOC** (HL+18H/ExtBIOS) – HL value obtained via EXTBIO

Function: Returns the number of characters in the receive buffer.

Input: None.

Output: HL – Number of characters in buffer.

Registers: AF.

**LOF** (HL+1BH/ExtBIOS) – HL value obtained via EXTBIO

Function: Returns the free space in the receive buffer.

Input: None.

Output: HL – Free space in bytes.

Registers: AF.

**BACKUP** (HL+1EH/ExtBIOS) – HL value obtained via EXTBIO

Function: Saves a character in a special buffer. The previous character is lost.

Input: C – Character to be saved.

Output: None.

Registers: F.

**SNDBRK** (HL+21H/ExtBIOS) – HL value obtained via EXTBIO

Function: Sends the specified number of “break” characters.

Input: DE – Number of “break” characters to be sent.

Output: CY = 1 → CTRL+STOP were pressed together.

Registers: AF, DE.

**DTR** (HL+24H/ExtBIOS) – HL value obtained via EXTBIO

Function: Turns the DTR line on/off.

Input: A = 0 → Disconnect the DTR line.

A ≠ 0 → Connects the DTR line.

Output: None.

Registers: F.

**SETCHN** (HL+27H/ExtBIOS) – HL value obtained via EXTBIO

Function: Select the channel number (only for multi-channel interfaces).

Input: A – Channel number.

Output: CY = 1 → The channel does not exist on the interface.

Registers: AF, BC.

#### 8.5.4.3 – MSX Modem manipulation routines

**INIT** (HL+03H/ExtBIOS) – HL value obtained via EXTBIO

Function: Initializes MSX Modem.

Input: A – modem type.

0 – BELL 103 300 bps full duplex

1 – BELL 212 A 1200 bps full duplex

2 – CCITT V 21 300 bps full duplex

3 – CCITT V 22 1200 bps full duplex

4 – CCITT V22bis 2400 bps full duplex

5 – CCITT V 23 1200 bps half duplex

6 – CCITT V27ter 4800 bps half duplex

7 – CCITT V 29 9600 bps half duplex

8 – CCITT V32 9600 bps full duplex

9 to 254 – Reserved for future expansions.

255 – System default.



- C – Dialing mode:
    - 0 – DTMF (tone prompting)
    - 1 – Reserved for future expansions.
    - 2 – Pulses (20 pps)
    - 3 – Pulses (10 pps)
    - 4 – Automatic
    - 5 to 254 – Reserved for future expansions.
    - 255 – System default.
  - B – ID of the slot from the parameter table.
  - HL – Address of the parameter table, with the following structure (from +00H to +07H values must be in ASCII code):
    - +00H – Character length ("5", "6", "7" or "8")
    - +01H – Parity ("E", "O", "I" or "N")
    - +02H – Stop bits ("1", "2" or "3")
    - +03H – XON/XOFF ("X" or "N")
    - +04H – CTR-RTS hand shake ("H" or "N")
    - +05H – Auto LF reception ("A" or "N")
    - +06H – Auto LF transmission ("A" or "N")
    - +07H – SI/SO Control ("Y" or "N")
    - +08H~0BH – Not used
    - +0CH – Time counter (0 to 255)
- Output: CY = 0 → MSX Modem successfully started.  
 1 → Parameter error.

Registers: AF.

**NCUSTA** (HL+2AH/ExtBIOS) – HL value obtained via EXTBIOS

Function: Returns NCU status.

Input: None.

Output: HL – State.

- bit 15~bit 9 – Always 0.
- bit 8: 0 – No DTMF data.  
 1 – Receiving DTMF data
- bit 7: 0 – External telephone on hook  
 1 – External telephone off-hook
- bit 6: 0 – No ringing tone  
 1 – 400 Hz ring tone detected
- bit 5: locks line polarity inversion

b4,b3: 00 – Loop off  
           01 – DC loop (LB)  
           10 – DC loop (LA)  
           11 – Undefined  
 b2,b1: dialing mode  
           00 – DTMF  
           01 – Pulse (10 pps)  
           10 – Pulse (20 pps)  
           11 – Automatic  
 bit 0: 0 – No bell signal (ring)  
           1 – Bell signal (ring) present

Registers: All.

**SPKCNT** (HL+2DH/ExtBIOS) – HL value obtained via EXTBIO

Function: Turns the speaker on/off.

Input: A = 0 → Turns off the speaker.

A ≠ 0 → Turns on the speaker.

Output: CY = 1 if this function is not supported.

Registers: F.

**LINSEL** (HL+30H/ExtBIOS) – HL value obtained via EXTBIO

Function: Switch the line.

Input: A – bit 7~5 – Reserved (always 0).

b4~b3 – releases the line (puts the internal phone on the “hook”). Bit4 releases the speaker and bit3 releases the microphone).

b2~b1 – connect the built-in telephone to the modem to the outside line (bit2 = 1, connect speaker,

bit1 = 1, connect microphone).

bit 0 – Switches between modem and external telephone:

b0 = 0 → connect the internal modem;

b0 = 1 → connect the telephone connected to the “TEL” port of the modem.

Output: CY = 1 if there is an error in the parameters.

Registers: None.

**DIALST** (HL+33H/ExtBIOS) – HL value obtained via EXTBIO

Function: Connect the device to the line and “dial”.

Input: C – Dial mode:  
           0 – DTMF (tone dialing)  
           1 – Reserved for future expansions.  
           2 – Pulses (20 pps)  
           3 – Pulses (10 pps)  
           4 – Automatic  
           5 to 254 – Reserved for future expansions.  
           255 – System default.  
 B – ID of the slot from the parameter table.  
 HL – Starting address of the dial data to be sent. Valid characters for “dial” are: “0”~“9”, “A”~“D”, “#”, “\*”, “H”, “<”, “:.” and “T”. “H” means 1 second on-hook, “<” means three, “T” selects tone dialing and “:.” waits for second dial tone. The data list must end with a 00H byte.

Output: CY = 1 if there is an error in the parameters.

Registers: None.

**DIALCH** (HL+36H/ExtBIOS) – HL value obtained via EXTBIO  
 Function: Sends a single character at a time to “dial”.  
 Input: A – character to be sent.  
       C – dial mode (same as DIALST(HL+33H)).  
 Output: CY = 1 if there is an error in the parameters.  
 Registers: None.

**DTMFST** (HL+39H/ExtBIOS) – HL value obtained via EXTBIO  
 Function: Checks the status of the DTMF decoder.  
 Input: None.  
 Output: Z = 1 if DTMF code is in input mode.  
       CY = 1 if this function is not supported.  
 Registers: AF.

**RDDTMF** (HL+3CH/ExtBIOS) – HL value obtained via EXTBIO  
 Function: Read data from DTMF decoder.  
 Input: None.  
 Output: A – DTMF Code (in ASCII)  
       CY = 1 if CTRL+STOP are pressed together or if this function is not supported.  
 Registers: AF.

**HOKCNT** (HL+3FH/ExtBIOS) – HL value obtained via EXTBIO

Function: Connect or disconnect the line.

Input: A – 0 = On hook  
           1 = Off the hook

Output: CY = 1 if this function is not supported.

Registers: None.

**CONFIG** (HL+42H/ExtBIOS) – HL value obtained via EXTBIO

Function: Returns hardware specifications.

Input: A – 0 to 255.

Output: HL – Specifications.

- When A = 0:

bit 15~09: always 0

bit 8 = 1 → CCITT V 32 9600 bps full duplex

bit 7 = 1 → CCITT V 29 9600 bps half duplex

bit 6 = 1 → CCITT V 27ter 4800 bps half duplex

bit 5 = 1 → CCITT V 23 1200 bps half duplex

bit 4 = 1 → CCITT V 22a 2400 bps full duplex

bit 3 = 1 → CCITT V 22 1200 bps full duplex

bit 2 = 1 → CCITT V 21 300 bps full duplex

bit 1 = 1 → BELL 212 At 1200 bps full duplex

bit 0 = 1 → BELL 103 300 bps full duplex

- When A = 1:

bit 15~08: always 0

bit 7 = 1 → support 10pps↔20pps change by software

bit 6 = 1 → DTMF – Soft pulse switching

bit 5 = 1 → supports "H"

bit 4 = 1 → support for "A" to "D"

bit 3 = 1 → automatic

bit 2 = 1 → pulse (20 pps)

bit 1 = 1 → pulse (10 pps)

bit 0 = 1 → DTMF

- When A = 2:

bit 15~8: always 0

bit 7 = 1 → support 10pps↔20pps change by software

bit 6~4: always 0

bit 3 = 1 → integrated handsfree phone

bit 2 = 1 → built-in standard telephone

bit 1 = 1 → internal modem

bit 0 = 1 → external telephone

- When A = 3:

bit 15~bit 13: always 0

bit 12 = 1 → long loop detection function

bit 11 = 1 → carrier control function

bit 10 = 1 → transmission power switching function

bit 9 = 1 → RS-232C

bit 8 = 1 → standard MSX cartridge

bit 7 = 1 → external telephone hook detection  
(on-hook or off-hook).

bit 6 = 1 → “on hook” / “off hook” function

bit 5 = 1 → has speaker

bit 4 = 1 → has DTMF decoder

bit 3 = 1 → charging pulse detection

bit 2 = 1 → line polarity detection

bit 1 = 1 → call progress detection

bit 0 = 1 → touch signal detection

- When A is 4 to 255:

HL = 0000H

Registers: HL.

**SPECIAL** (HL+45H/ExtBIOS) – HL value obtained via EXTBIO

Function: Implements special functions for each modem model.

Input: A = 0 → Send modem power switching function.

C – transmission power value (dBm). If it is 255, it defaults to value.

A = 1 → Carrier wave control.

C = 0 – Turns off the carrier.

1 – Turns on the carrier.

H – delay time up to RS ON ( $n * 10$  mS)

L – delay time from CS ON to RETURN ( $n * 10$  mS)

A = 2 → Equalizer setting.

C = 0 – Do not use equalizer.

1 – Use the equalizer.

2 – Automatic equalizer adjustment

255 – Defaults.

Output: CY = 1 if selected function is not supported.

Registers: Depends on the called function.

### 8.5.5 – MSX-AUDIO

#### **EXTBIO** (FFCAH/Work Area)

Function: Access extended BIOS functions

Input: A – 00H.

D – 0AH – MSX-Audio manipulation device.

E – 00H – Returns the pointer to the MSX-Audio information table.

B – Address table slot ID.

HL – Address of a 64-byte buffer for the table (should be on page 3).

Output: B – ID of the information table slot.

HL – HL is incremented by 4 and will point to the end of a table that reserves 4 bytes for MSX-Audio. The original HL value points to the beginning of the table, which has the following structure:

+00H – Slot ID

+01H – Lowest address

+02H – Highest address

+03H – Reserved for expansion

The slot ID (+00H) and the address (+01H,+02H) will point to a table with the following structure:

+00H VERSION Software version

+03H MBIOS Music BIOS

+06H AUDIO Initialization of MSX-Audio

+09H SYNTHE Calls the SYNTHE app

+0CH PLAYF State instruction PLAY

+0FH BGM Enable/cancel BGM mode

+12H MKTEMP Set recording time / musical keyboard playback

+15H PLAYMK Plays via musical keyboard

+18H RECMK Records the notes played on the musical keyboard

+1BH STOPM Keyboard playback / recording / ADPCM; stops command PLAY

+1EH CONTMK Continue recording by musical keyboard

+21H RECMOD Sets recording mode of the musical keyboard

|      |           |   |
|------|-----------|---|
| +24H | STPPPLY   | Stops the PLAY instruction                      |
| +27H | SETPCM    | Protected Area ADPCM/PCM                        |
| +2AH | RECPCM    | ADPCM/PCM Recording                             |
| +2DH | PLAYPCM   | ADPCM/PCM Playback                              |
| +30H | PCMFREQ   | Changing the frequency of<br>ADPCM/PCM playback |
| +33H | MKPCM     | Set/cancel data ADPCM for<br>musical keyboard   |
| +36H | PCMVOL    | Sets the volume of ADPCM/PCM<br>playback        |
| +39H | SAVEPCM   | Save ADPCM/PCM data                             |
| +3CH | LOADPCM   | Load ADPCM/PCM data                             |
| +3FH | COPYPCM   | Transfers ADPCM/PCM data                        |
| +42H | CONVP     | Converts ADPCM to PCM data                      |
| +45H | CONVA     | Converts PCM to ADPCM data                      |
| +48H | VOICE     | Sets FM data                                    |
| +4BH | VOICECOPY | Moves FM data                                   |

Registers: F.

#### **EXTBIO** (FFCAH/Work Area)

Function: Access extended BIOS functions

Input: A – 00H.

D – 0AH – MSX-Audio manipulation device.

E – 01H – Returns how many MSX-Audio cartridges are  
connected to the MSX (maximum 2).

Output: A – 0 → There is no MSX-Audio connected.

1 → There is an MSX-Audio cartridge connected.

2 → There are two MSX-Audio cartridges connected.

Registers: BC, DE, HL.

#### **8.5.5.1 – Startup routines**

##### **VERSION** (HL+00H) – HL value obtained via EXTBIO

Function: BIOS version. Usually 00H-00H-00H.

##### **MBIOS** (HL+03H) – HL value obtained via EXTBIO

Function: Call the MBIOS routines (Music BIOS).

Input: HL – Address of the MBIOS routine.  
 IX and IY are used for interslot calling and must be defined  
 in BUF (F55EH) as follows:  
 BUF +00H/+01H – IX  
 BUF +02H/+03H – IY

Output: Depends on MBIOS routine.

Registers: It depends on the MBIOS routine.

**AUDIO** (HL+06H) – HL value obtained via EXTBIO

Function: Initialize MSX-Audio.

Input: Set the following values in BUF (F55EH):

- +01H – Switch mode
- +02H – Number of FM instruments used to configure  
MSX-Audio (0 to 9)
- +03H – Number of FM sound sources for the first string  
(0 to 9)
- +04H – Number of FM sound sources for the second string  
(0 to 8)
- +05H – Number of FM sound sources for the third string  
(0 to 7)
- +06H – Number of FM sound sources for the fourth string  
(0 to 6)
- +07H – Number of FM sound sources for the fifth string  
(0 to 5)
- +08H – Number of FM sound sources for the sixth string  
(0 to 4)
- +09H – Number of FM sound sources for the seventh  
string (0 to 3)
- +0AH – Number of FM sound sources for the eighth string  
(0 to 2)
- +0BH – Number of FM sound sources for the ninth string  
(0 to 1)

Output: CY = 1 → Initialization failed.

Registers: All.

**SYNTHE** (HL+09H) – HL value obtained via EXTBIO

Function: Calls the built-in SYNTHE application.

Input: None.

Output: None.

Registers: All.



### 8.5.5.2 – PCM/ADPCM Routines

**SETPCM** (HL+27H) – HL value obtained via EXTBIO

Function: Initializes the audio file for PCM/ADPCM.

Input: Set the parameters in BUF (F55EH):

+00H – Audio file number (0 to 15).

+01H – Device number (0 to 5, except 4).

0 or 2 → external RAM

1 or 3 → external ROM

4 → CPU (cannot be used)

5 → VRAM

+02H – Mode (0 or 1).

+03H/+04H – Depends on the device number:

RAM: No need to define

ROM: +3H – File number. audio in ROM

+4H – Always 0.

VRAM: +3H – VRAM Address (LSB)

+4H – VRAM Address (MSB)

+05H/+06H – Length (LSB-MSB).

+07H/+08H – Sampling frequency (LSB-MSB).

+09H – Channel number (0 or 1).

Output: CY = 1 → Parameter error, not configured.

Registers: All.

**RECPCM** (HL+2AH) – HL value obtained via EXTBIO

Function: Record audio file.

Input: Set the following parameters in BUF (F55EH):

+00H – Audio file number (0 to 15).

+01H – Synchronization (0 or 1).

+02H/+03H – Displacement (LSB-MSB).

+04H/+05H – Length (LSB-MSB). FFFFH to use values defined by SETPCM (HL+27H).

+06H/+07H – Sampling frequency (LSB-MSB). FFFFH to use values defined by SETPCM (HL+27H).

+08H – Channel number (0 or 1). FFH to use channel defined by SETPCM (HL+27H).

Output: CY = 1 → Parameter error, write cancelled.

Registers: All.

**PLAYPCM** (HL+2DH) – HL value obtained via EXTBIO

Function: Play audio file.

Input: Set the parameters in BUF (F55EH):

+00H – Audio file number (0 to 15).

+01H – Repeat flag (0 or 1).

+02H/+03H – Displacement (LSB-MSB).

+04H/+05H – Length (LSB-MSB). FFFFH to use values defined by SETPCM (HL+27H).

+06H/+07H – Sampling frequency (LSB-MSB). FFFFH to use values defined by SETPCM (HL+27H).

+08H – Channel number (0 or 1). FFH to use channel defined by SETPCM (HL+27H).

Output: CY = 1 → Parameter error, operation cancelled.

Registers: All.

**PCMFREQ** (HL+30H) – HL value obtained via EXTBIO

Function: Change the playback frequency.

Input: BC – First channel sampling frequency

DE – First channel sampling frequency

The frequency can vary from 1800 to 49,716 Hz. If there is no second channel, set DE value equal to BC.

Output: CY = 1 → Parameter error. The frequency is not changed.

Registers: All.

**PCMVOL** (HL+36H) – HL value obtained via EXTBIO

Function: Sets the PCM/ADPCM playback volume.

Input: BC – Volume of the first channel (0 to 63), where 63 is max.

DE – First channel volume (0 to 63), where 63 is max.

The initial value is 63 for ADPCM and 32 for PCM. If there is no second channel, set DE value equal to BC.

Output: CY = 1 → Parameter error. Volume is not set.

Registers: All.

**SAVEPCM** (HL+39H) – HL value obtained via EXTBIO

Function: Save PCM/ADPCM audio file to disk.

Input: A – Audio file number.

HL – Pointer to the filename. It must be enclosed in double quotes (22H) and end with byte 00H (Ex. "FILENAME.PCM",00H), as in MSX-BASIC.

Output: CY = 1 → Wrong audio file number. The file will not be saved.

Registers: All.

Note: If there are any errors during the save, control will be returned to the BASIC interpreter.

### **LOADPCM** (HL+3CH) – HL value obtained via EXTBIO

Function: Load PCM/ADPCM audio file from disk.

Input: A – Audio file number.

HL – Pointer to the filename. It must be enclosed in double quotes (22H) and end with byte 00H (Ex. "FILENAME.PCM",00H), as in MSX-BASIC.

Output: CY = 1 → Wrong audio file number. The file will not be loaded.

Registers: All.

Note: If there are any errors during loading, control will be returned to the BASIC interpreter.

### **COPYPCM** (HL+3FH) – HL value obtained via EXTBIO

Function: Transfer PCM/ADPCM data between audio files.

Input: Set the parameters in BUF (F55EH):

+00H – Source file number (0 to 15).

+01H – Destination file number (0 to 15).

+02H/+03H – Offset of source file (LSB-MSB).

+04H/+05H – Length (LSB-MSB).

+06H/+07H – Offset destination file (LSB-MSB).

+08H – Font specification (0 or 1).

Output: CY = 1 → Parameter error, transfer cancelled.

Registers: All.

### **CONVP** (HL+42H) – HL value obtained via EXTBIO

Function: Convert data from PCM format to ADPCM.

Input: Set the parameters in BUF (F55EH):

+00H – Source file number (0 to 15).

+01H – Destination file number (0 to 15).

Output: CY = 1 → Parameter error, conversion cancelled.

Registers: All.

**CONVA** (HL+45H) – HL value obtained via EXTBIO

Function: Convert data from ADPCM format to PCM.

Input: Set the parameters in BUF (F55EH):  
       +00H – Source file number (0 to 15).  
       +01H – Destination file number (0 to 15).

Output: CY = 1 → Parameter error, conversion cancelled.

Registers: All.

**MKTEMPO** (HL+18H) – HL value obtained via EXTBIO

Function: Sets the time for recording and playback through the musical keyboard, with metronome function.

Input: DE – Time in quarter notes per minute (25 to 360).

Output: CY = 1 → Parameter error, configuration cancelled.

Registers: All.

**MKPCM** (HL+33H) – HL value obtained via EXTBIO

Function: Specify the ADPCM sound file to play with the musical keyboard.

Input: A – Audio file number (0 to 15). To cancel, use FFH.

Output: CY = 1 → Parameter error, playback cancelled.

Registers: All.

### 8.5.5.3 – Musical keyboard routines

**PLAYMK** (HL+15H) – HL value obtained via EXTBIO

Function: Plays recorded audio via musical keyboard.

Input: DE – Starting address of reproduction.  
       BC – Final address of reproduction.

Output: None.

Registers: All.

**RECMK** (HL+18H) – HL value obtained via EXTBIO

Function: Records audio through the musical keyboard.

Input: DE – Starting address for recording.  
       BC – Final address for recording.

Output: None.

Registers: All.

**CONTMK** (HL+1EH) – HL value obtained via EXTBIO

Function: Continue recording or playing musical keyboard audio that was interrupted by STOPM.

Input: None.

Output: None.

Registers: All.

**RECMOD** (HL+21H) – HL value obtained via EXTBIO

Function: Sets the recording mode for the musical keyboard.

Input: A = 0 → Muting (do not record)

1 → Record

2 → Play

3 → Record and play simultaneously

Output: CY = 1 → Parameter error, configuration cancelled.

Registers: All.

**8.5.5.4 – FM synthesizer routines****PLAYF** (HL+0CH) – HL value obtained via EXTBIO

Function: Checks the status of the PLAY instruction.

Input: A – PLAY instruction channel number (0 = All channels).

Output: HL – 0000H → the specified channel is NOT playing.

FFFFH → the specified channel is playing

(when specified for all channels, HL will return FFFFH if any are active).

Registers: All.

**BGM** (HL+0FH) – HL value obtained via EXTBIO

Function: Specifies background execution.

Input: 0 – Does NOT perform background processing.

1 – Runs background processing (default). The functions available for the background are: playback via the PLAY command, ADPCM recording and playback via microphone, and recording and playback via the musical keyboard.

Output: None.

Registers: All.

**STOPM** (HL+1BH) – HL value obtained via EXTBIO

Function: Stop playback and recording.

Input: None.

Output: None.

Registers: All.

**STPLY** (HL+1BH) – HL value obtained via EXTBIO

Function: Stop playback of PLAY command only.

Input: None.

Output: None.

Registers: All.

**VOICE** (HL+48H) – HL value obtained via EXTBIO

Function: Sets the instrument for each FM channel.

Input: Define the following parameters in BUF (F55EH):

+0 → Voice 1 parameter block

+4 → Voice 2 Parameter Block

⋮

(n-1)\*4 → Voice n parameter block

n\*4 → End mark (FFH).

• Specifying instruments provided in ROM:

+0 → Channel number (0 to 8).

+1 → 00H.

+2 → Instrument number in ROM (0 to 63).

+3 → 00H.

• Specifying user instrument:

+0 → Channel number (0 to 8).

+1 → FFH.

+2/+3 → Instrument data address (LSB-MSB).

Output: CY = 1 → Parameter error, configuration cancelled.

Registers: All.

**VOICECOPY** (HL+4BH) – HL value obtained via EXTBIO

Function: Transfers data from FM instruments.

Input: Define the following parameters in BUF (F55EH):

• Transfer 0~63 instruments from ROM to 32~63 system instruments:

+0 → 00H

+1 → Source instrument number (0~63).

+2~+5 → 00H

- +6 → Target instrument number (32~63).
- +7~+9 → 00H
- Transfer 0~63 instruments from ROM to user data area:
  - +0 → 00H
  - +1 → Source instrument number (0~63).
  - +2~+4 → 00H
  - +5 → FFH
  - +6~+7 → Destination address in the data area.
  - +8~+9 → 00H
- Transfer instruments from user data area to 32~63 system instruments:
  - +0 → FFH
  - +1~+2 → Source address in the data area.
  - +3~+5 → 00H
  - +6 → Target instrument number (32~63).
  - +7~+9 → 00H
- Transfer all 32~63 instruments from the system to the user data area:
  - +0 → 00H
  - +1 → FFH
  - +2 ~ +4 → 00H
  - +5 → FFH
  - +6 ~ +7 → Destination address in the data area.
  - +8 ~ +9 → Length of data in bytes.
- Transfers all instruments from the user data area to 32~63 system instruments:
  - +0 → FFH
  - +1 ~ +2 → Destination address in the data area.
  - +3 ~ +4 → Length of data in bytes.
  - +5 → 00H
  - +6 → FFH
  - +7 ~ +9 → 00H.

#### 8.5.5.5 – MBIOS routines (Music BIOS)

The Music BIOS routines must be called through the MBIOS entry of the jump table, setting in HL the call address of the desired Music BIOS routine. The MBIOS format is as follows:

**MBIOS** (JumpTable+03H) – JumpTable value obtained via EXTPIO

Function: Call the MBIOS routines (Music BIOS).

Input: HL – Address of the MBIOS routine.

IX and IY are used for interslot calling and must be defined in BUF (F55EH) as follows:

BUF+00H/+01H – IX

BUF+02H/+03H – IY

Output: Depends on Music BIOS routine.

Registers: It depends on the Music BIOS routine.

The data tables used by Music BIOS are as follows:

#### CHDB (32 bytes)

|         |             |
|---------|-------------|
| +00     | YCAO0_MULTI |
| +01     | YCAO0_LS    |
| +02     | YCAO0_AR    |
| +03     | YCAO0_RR    |
| +04     | YCAO0_VELS  |
| +05     | YCAO0_VTL   |
| +06~+07 | Unused      |
| +08     | YCAO1_MULTI |
| +09     | YCAO1_LS    |
| +10     | YCAO1_AR    |
| +11     | YCAO1_RR    |
| +12     | YCAO1_VELS  |
| +13     | YCAO1_VTL   |
| +14~+15 | Unused      |
| +16~+17 | YCA_VTRANS  |
| +18~+19 | YCA_TRANS   |
| +20     | YCA_TRIG    |
| +21     | YCA_VOL     |
| +22     | YCA_FB      |
| +23     | YCA_VEL     |
| +24~+25 | YCA_PITCH   |
| +26     | YCA_VOICE   |
| +27     | ZCA_FLAG    |
| +28     | ZC_CH       |
| +29     | ZC_OP       |
| +30~+31 | ZC_COUNT    |

#### MIDB (64 bytes)

|         |               |
|---------|---------------|
| +00~+01 | Unused        |
| +02     | YM_TIM 1      |
| +03     | YM_TIM 2      |
| +04~+17 | Unused        |
| +18     | YMA_BIAS      |
| +19~+24 | Unused        |
| +25     | YMA_AUDIO     |
| +26~+31 | Unused        |
| +32~+33 | YMA_TRANS     |
| +34     | YMA_LFO       |
| +35     | YMA_RAM       |
| +36     | ZMA_FLAG      |
| +37~+38 | YMA_PDB       |
| +39     | ZMA_PH_FILTER |
| +40     | ZMA_PH_TL     |
| +41~+42 | ZMA_PH_AR     |
| +43~+44 | ZMA_PH_DIR    |
| +45     | ZMA_PH_SL     |
| +46~+47 | ZMA_PH_D 2 R  |
| +48~+49 | ZMA_PH_RR     |
| +50~+51 | ZMA_PH_EG     |
| +52     | ZMA_PH_STAT   |
| +53~+63 | Unused        |



**PDB (PCM Data Block)**

|         |            |   |
|---------|------------|---|
| +00     | PDB_DEV    | Defines the PCM/ADPCM device  |
| +01     | Unused     |   |
| +02~+03 | PDB_ADDR   | Data start address  |
| +04~+05 | PDB_SIZE   | Data Block Size   |
| +06~+07 | PDB_SAMPLE | Sampling frequency<br>(1800 to 16000 for ADPCM<br>or 1800 to 12000 for PCM) |
| +08~+09 | PDB_PCM    | Initial value when ADPCM is<br>tracked and converted to PCM.                |
| +10~+11 | PDB_STEP   | Initial Quantize Width<br>when ADPCM is tracked and<br>converted to PCM     |
| +12~+15 | Unused     |   |

The Music BIOS routines are as follows:

**SV\_RESET** (0090H/MBIOS)

Function: Initialize the MBIOS.

Input: None.

Output: None.

Registers: None.

Note: Interruptions are disabled on return. Before re-enabling them, the MBIOS hook must be set.

**SV\_DI** (0093H/MBIOS)

Function: Disables user interrupts.

Input: None.

Output: None.

Registers: None.

**SV\_EI** (0096H/MBIOS)

Function: Allow user interrupts.

Input: None.

Output: None.

Registers: None.

**SV\_ADW** (0099H/MBIOS)

Function: Write a byte of data into a Y8950 register.

Input: IY – Master/slave specification by MIDB address.  
A – Byte of data to be written.  
C – Registrar number.

Output: CY = 1 → there was an attempt to write to a non-existent slave device.

Registers: All.

### SV\_ADW\_DI (009CH/MBIOS)

Function: Write a data byte in a Y8950 register, disabling loopback interrupts.

Input: IY – Master/slave specification by MIDB address.

A – Byte of data to be written.

C – Register number.

Output: CY = 1 → there was an attempt to write to a non-existent slave device.

IFF = 0 (Interrupts Disabled)

Registers: All.

### SV\_SETUP (00ABH/MBIOS)

Function: Initial setup of various functions.

Input: A – Function code.

0 – SM\_AUDIO → tone setting.

1 – SC\_CHDB → initialize CHDB desktop.

2 – SM\_INST → initialize the instrument function.

3 – SM\_MK → initialize musical keyboard reading.

Other parameters depend on the function.

Output: CY = 1 → Configuration failed (usually because the routine is called via interrupts).

Registers: All.

### SM\_AUDIO (00ABH/MBIOS)

Function: Set the FM synthesizer music tone mode.

Input: A – 0.

C – 

| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
|----|----|----|----|----|----|----|----|
| 0  | 0  | 0  | 0  | 0  | N  | M  | S  |

0 – FM slave mode

1 – CSM slave mode

0 – FM master mode

1 – CSM master mode

0 – 9 channels

1 – 6 chan. + 5 drum pieces

DE – FM synthesizer tone channel.

bit0 = 1 → instrument in channel 0

bit1 = 1 → instrument on channel 1

⋮

bit8 = 1 → instrument on channel 8

(In 6 channel mode + 5 drum pieces, only channels 0 to 5 can be assigned).

Output: None.

Registers: All.

Note: This routine internally calls SC\_CHDB and SM\_INST.

### **SC\_CHDB** (00ABH/MBIOS)

Function: Launches the CHDB desktop.

Input: A – 1.

IX – CHDB address to be initialized.

Output: None.

Registers: All.

### **SM\_INST** (00ABH/MBIOS)

Function: Initializes the instrument's tone with timbre #0.

Input: A – 2.

Output: None.

Registers: All.

### **SM\_MK** (00ABH/MBIOS)

Function: Initializes music keyboard reading.

Input: A – 3.

B – 1 → connects the keyboard to the instrument.

0 → do not connect the keyboard.

C – speed when keys are pressed. 0 is the slowest and 15 is the fastest. Velocity is referenced in SV\_MK (musical keyboard scan).

Output: None.

Registers: All.

### **SV\_REAL** (00AEH/MBIOS)

Function: Perform real-time operations. This call is divided into several functions designated by codes that are as follows:

- 00 RM\_MOVE\_DI – Transfer. ADPCM/PCM data
- 01 RM\_TRACE\_DI – ADPCM Data Trace
- 02 RM\_CONV\_PCM\_DI – ADPCM to PCM Conversion
- 03 RM\_CONV\_ADPCM\_DI – Conversion PCM→ADPCM
- 04 RMA\_DAC\_BIAS – Volume for PCM playback
- 05 RMA\_DAC\_DI – PCM Data Playback
- 06 RMA\_ADC\_DI – PCM Data Write
- 07 RMA\_ADPCM\_BIAS – Configure ADPCM playback
- 08 RMA\_ADPLAY\_DI – Playback of ADPCM data
- 09 RMA\_ADREC\_DI – Writing ADPCM data
- 10 RMA\_BREAK – Interrupt. playback/recording
- 11 RMA\_ADPLAY – Playback of ADPCM data
- 12 RMA\_ADREC – Writing ADPCM data
- 13 RMA\_PHASE\_SET\_DI – Converts 256 bytes PCM
- 14 RMA\_PHASE\_EG – Configures the envelope
- 15 RMA\_PHASE\_EVENT – Pitch Sampling
- 16 RM\_TIMER – Enables/disables interrupt timer.
- 17 RM\_TIM1 – Sets timer 1
- 18 RM\_TIM2 – Sets timer 2
- 19 RM\_TEMPO – Defines the cycle of timer 2
- 20 RM\_DAMP – FM generator stopping force
- 21 RM\_PERC – Plays rhythm sound
- 22 RMA\_MK – Returns musical keyboard status
- 23 RMA\_LFO – Sets vibrato
- 24 RMA\_TRANS – Configures sound transition
- 25 RMA\_CSM\_DI – Reproduction of CSM data
- 26 RM\_READ\_DI – Transf. 256 bytes ADPCM/PCM
- 27 RM\_WRITE\_DI – Transf. 256 bytes ADPCM/PCM
- 28 RM\_UTEMPR – Converts the temperament pitch
- 29 RM\_CTEMPR – Temper setting
- 30 RM\_PITCH – Set current/subsequent pitch
- 31 RM\_TSRAN – Configures sound transposition
- 32 RC\_NOTE – Turns FM voice on/off
- 33 RC\_LEGATO – Turns FM voice legato on/off
- 34 RC\_DAMP – Interrupts FM voice
- 35 RC\_KON – Turns on the FM generator voice
- 36 RC\_LEGATO\_ON – Turns on legato FM voice
- 37 RC\_KOFF – Turns off FM generator voice
- 38 RCA\_PARAM – FM real-time configuration

39 RCA\_VOICE – Configures voice for the FM channel  
 40 RCA\_VPARAM – Configures voice parameters  
 41 RCA\_VOICEP – Configures voice for the FM channel  
 42 RMA\_ADPLAYLP – Playback ADPCM data repeats  
 43 RMA\_ADPLY\_SAMPLE – ADPCM Playback  
 44 RM\_PVEL – Sets rhythm sound speed  
 :  
 48 RI\_DAMP FM – Generator stopping force  
 49 RI\_ALLOFF – Activates all FM channels  
 50 RI\_EVENT – Converts a note  
 51 RI\_PCHB – Setting the pitch position  
 52 RI\_PCHBR – Setting pitch pitch  
 53 RIA\_PARAM – Realtime setting for FM  
 54 RIA\_VOICE – FM Voice Configuration  
 55 RIA\_VPARAM – Voice Definition for FM  
 56 RIA\_VOICEP – Configure tone for FM

Input: A – function code.

Other parameters depend on the called function.

Output: CY = 1 → error in input parameters.

Other parameters depend on the called function.

Registers: Depends on the called function.

### **RC\_NOTE** (00AEH/MBIOS)

Function: Turns on a voice on the FM generator and turns off automatically after a specified time.

Input: A – 32.

IX – CHDB address with FM voice data.

DE – Range (0~32.767). The central value is 15,360 and a semitone corresponds to 256.

C – Speed (0~15). 0 is the slowest and 15 is the fastest.

B – Timer. Turns off when SV\_TEMPO is called this number of times.

Output: None.

Registers: All.

### **RC\_LEGATO** (00AEH/MBIOS)

Function: Turns on a voice on the FM generator and turns off automatically after a specified time. Unlike RC\_NOTE, this function does not start wrapping.

Input: A – 33.  
 IX – CHDB address with FM voice data.  
 DE – Range (0~32.767). The central value is 15,360 and a semitone corresponds to 256.  
 C – Speed (0~15). 0 is the slowest and 15 is the fastest.  
 B – Timer. Turns off when SV\_TEMPO is called this number of times.

Output: None.

Registers: All.

### **RC\_DAMP** (00AEH/MBIOS)

Function: Forces the FM voice that is playing to stop.

Input: A – 34.  
 IX – CHDB address with FM voice data.

Output: None.

Registers: All.

### **RC\_KON** (00AEH/MBIOS)

Function: Connects an FM voice.

Input: A – 35.  
 IX – CHDB address with FM voice data.  
 DE – Range (0~32.767). The central value is 15,360 and a semitone corresponds to 256.  
 C – Speed (0~15). 0 is the slowest and 15 is the fastest.

Output: None.

Registers: All.

### **RC\_LEGATO\_ON** (00AEH/MBIOS)

Function: Connects an FM voice. Unlike RC\_KON, this function does not start wrapping.

Input: A – 36.  
 IX – CHDB address with FM voice data.  
 DE – Range (0~32.767). The central value is 15,360 and a semitone corresponds to 256.  
 C – Speed (0~15). 0 is the slowest and 15 is the fastest.

Output: None.

Registers: All.

**RC\_KOFF** (00AEH/MBIOS)

Function: Turns off an FM voice.

Input: A – 37.

IX – CHDB address with FM voice data.

Output: None.

Registers: All.

**RCA\_PARAM** (00AEH/MBIOS)

Function: Adjust real-time parameters for an FM voice.

Input: A – 38.

IX – CHDB address with FM voice data.

C – Offset of the parameter to be adjusted in the CHDB list.

DE – Configuration data.

The data that can be configured with this function are as follows:

|           |          |           |
|-----------|----------|-----------|
| YCA_TRANS | YCA_TRIG | YCA_PITCH |
| YCA_VOL   | YCA_VEL  |           |

Output: None.

Registers: All.

**RCA\_VOICE** (00AEH/MBIOS)

Function: Associate an instrument with an FM voice.

Input: A – 39.

IX – CHDB address with FM voice data.

C – Instrument pattern number in ROM (0~63). The available instruments are as follows:

|    |              |    |                  |
|----|--------------|----|------------------|
| 0  | Piano 1      | 32 | Piano 3          |
| 1  | Piano 2      | 33 | Electric Piano 2 |
| 2  | Violin       | 34 | Santool 2        |
| 3  | Flute 1      | 35 | Brass            |
| 4  | Clarinet     | 36 | Flute 2          |
| 5  | Oboe         | 37 | Clavicode 2      |
| 6  | Trumpet      | 38 | Clavicode 3      |
| 7  | Pipe Organ 1 | 39 | Koto 2           |
| 8  | Xylophone    | 40 | Pipe Organ 2     |
| 9  | Organ        | 41 | PohdsPLA         |
| 10 | Guitar       | 42 | RohdsPRA         |
| 11 | Santool 1    | 43 | Orch L           |

|    |                  |    |              |
|----|------------------|----|--------------|
| 12 | Electric Piano 1 | 44 | Orch R       |
| 13 | Clavicode 1      | 45 | Synth Violin |
| 14 | Harpsicode 1     | 46 | Synth Organ  |
| 15 | Harpsicode 2     | 47 | Synth Brass  |
| 16 | Vibraphone       | 48 | Tube         |
| 17 | Koto 1           | 49 | Shamisen     |
| 18 | Taiko            | 50 | Magical      |
| 19 | Engine 1         | 51 | Huwawa       |
| 20 | UFO              | 52 | Wander Flat  |
| 21 | Synthesizer bell | 53 | Hardrock     |
| 22 | Chime            | 54 | Machine      |
| 23 | Synthesizer bass | 55 | Machine V    |
| 24 | Synthesizer      | 56 | Comic        |
| 25 | Synth Percussion | 57 | SE-Comic     |
| 26 | Synth Rhythm     | 58 | SE-Laser     |
| 27 | Harm Drum        | 59 | SE-Noise     |
| 28 | Cowbell          | 60 | SE-Star 1    |
| 29 | Close Hi-hat     | 61 | SE-Star 2    |
| 30 | Snare Drum       | 62 | Engine 2     |
| 31 | Bass Drum        | 63 | Silence      |

Output: None.

Registers: All.

### **RCA\_VPARAM (00AEH/MBIOS)**

Function: Adjust parameters of an FM voice.

Input: A - 40.

IX - CHDB address with FM voice data.

C - Offset of the parameter to be adjusted in the CHDB list.

DE - Configuration data.

The data that can be configured with this function are as follows:

|           |            |             |
|-----------|------------|-------------|
| CA00_LS   | CA01_LS    | YCA00_MULTI |
| CA00_AR   | YCA01_AR   | CA01_MULTI  |
| CA00_RR   | YCA01_RR   | YCA_VTRANS  |
| CA00_VELS | YCA01_VELS | YCA_FB      |
| CA00_VTL  | YCA01_VTL  |             |

Output: None.

Registers: All.



**RCA\_VOICEP (00AEH/MBIOS)**

Function: Set an FM voice with tone data.

Input: A - 41.

IX - CHDB address with FM voice data.

BC - Pointer to the data, which occupy 32 bytes with the following structure:

```

0~7 V_NAME   Sound name
8~9 V_TRANS  Transposition value
10 V_ARG     Several configurations:
    bit7 - Tremolo level:
            0- 1dB; 1- 4,8 dB
    bit6 - Vibrato level:
            0- 7%; 1- 14%
    bit5 - Defines tremolo/vibrato:
            0- no; 1- configure
    bit4 - Defines fixed tone:
            0- normal tone; 1- fixed tone
    bit3~bit1 - Feedback level:
            000 - 0          100 -  $\pi/2$ 
            001 -  $\pi/16$        101 -  $\pi$ 
            010 -  $\pi/8$         110 -  $2\pi$ 
            011 -  $\pi/4$         111 -  $4\pi$ 
    bit0 - Type of operators connection:
            0- serial; 1- parallel
11~15 - Unused
16 VO0_MULTI - Data to be configured
               for registers 20H (voice 0) to
               35H (voice 1):
    bit7 - Amplitude modulation:
            0- no; 1- yes
    bit6 - Vibrato:
            0- no; 1- yes
    bit5 - EG-TYP (type of envelope):
            0- decaying; 1- sustained
    bit4 - KSR (Key Scale Rate):
            0- no; 1- yes
    bit3~bit0 - Multiple:
            00-1/2  04-4   08-8   12-12
            01-1   05-5   09-9   13-12
            02-2   06-6   10-10  14-15
            03-3   07-7   11-10  15-15

```

- 17 VO0\_TL - Data to be configured for registers 40H (voice 0) to 55H (voice 1):
- bit7~bit6 - KSL (Key Scale Level):
    - 00 - 0 dB/octave
    - 01 - 1,5 dB/octave
    - 10 - 3 dB/octave
    - 11 - 6 dB/octave
  - bit7~bit6 - Total level:
    - bit0 - 0,75 dB
    - bit1 - 1,5 dB
    - bit2 - 3 dB
    - bit3 - 6 dB
    - bit4 - 12 dB
    - bit5 - 24 dB
- 18 VO0\_AR - Data to be configured for registers 60H (voice 0) to 75H (voice 1):
- bit7~bit4 - Attack Rate:
    - 0dB to 96dB: 1111 - 0 mS
    - 1110 - 0,2 mS
    - 0000 - 2826 mS
    - 10% to 90%: 1111 - 0 mS
    - 1110 - 0,11 mS
    - 0000 - 1482 mS
  - bit7~bit4 - Decay Rate:
    - 0dB to 96dB: 1111 - 0 mS
    - 1110 - 2,4 mS
    - 0000 - 39280 mS
    - 10% to 90%: 1111 - 0 mS
    - 1110 - 0,51 mS
    - 0000 - 8212 mS
- 19 VO0\_RR - Data to be configured for registers 80H (voice 0) to 95H (voice 1):
- bit7~bit4 - Sustain Level:
    - bit7 - 24 dB
    - bit6 - 12 dB
    - bit5 - 6 dB
    - bit4 - 3 dB
  - bit3~bit0 - Release Rate:
    - bit0 - 24 dB
    - bit1 - 12 dB
    - bit2 - 6 dB
    - bit3 - 3 dB

- 20 V00\_VELS - Speed sensitivity  
performed by software via MBIOS.  
bit7~bit4 - Unused.  
bit3~bit0 - Sensitivity:  
0000 - Invalid  
0001 - Minimum  
1111 - Maximum
- 21~23 - Unused.
- 24 V01\_MULTI (same as V00\_MULTI but  
acts on operator 1)
- 25 V01\_TL (same as V00\_TL but acts  
on operator 1)
- 26 V01\_AR (same as V00\_AR but acts  
on operator 1)
- 27 V01\_RR (same as V00\_RR but acts  
on operator 1)
- 28 V01\_VELS (same as V00\_VELS but acts  
on operator 1)
- 29~31 - Unused.

### **RM\_TIMER (00AEH/MBIOS)**

Function: Enable/disable timer functions.

Input: A - 16.

C - bit7~bit2 - Unused.

bit1 - Timer 2: 0- Disable; 1- Activate.

bit0 - Timer 1: 0- Disable; 1- Activate.

Output: None.

Registers: All.

### **RM\_TIM1 (00AEH/MBIOS)**

Function: Set the value of timer #1.

Input: A - 17.

C - Period with 80 uS step. Corresponds to 20.48 mS  
when C=0 and 80 uS when C=255.

Output: None.

Registers: All.

### **RM\_TIM2 (00AEH/MBIOS)**

Function: Set the value of timer #2.

Input: A - 18.

C - Period with 80 uS step. Corresponds to 20.48 mS  
when C=0 and 80 uS when C=255.

Output: None.

Registers: All.

### RM\_TEMPO (00AEH/MBIOS)

Function: Set timer cycle #2.

Input: A - 19.

C - Number of quarter notes per minute.

Output: None.

Registers: All.

### RM\_DAMP (00AEH/MBIOS)

Function: Force stop of all active FM generator channels.

Input: A - 20.

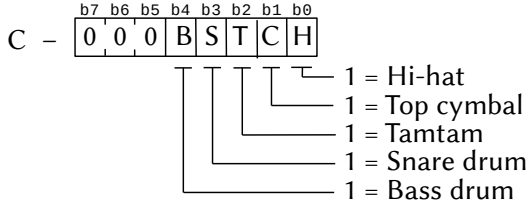
Output: None.

Registers: All.

### RM\_VEL (00AEH/MBIOS)

Function: Sets the speed of the five drum pieces (rhythm). This function can define more than one part at a time.

Input: A - 44.



E - Speed. 0 is the fastest and 31 is the slowest.

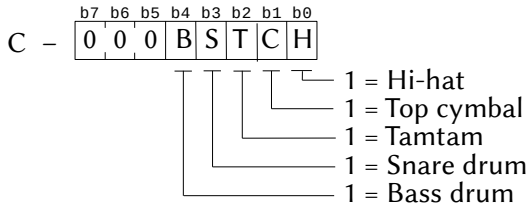
Output: None.

Registers: All.

### RM\_PERC (00AEH/MBIOS)

Function: Activates drum parts sound (rhythm). Several pieces can be played simultaneously.

Input: A - 21.



E – Speed. 0 is the fastest and 31 is the slowest.

Output: None.

Registers: All.

### RMA\_MK (00AEH/MBIOS)

Function: Returns the state of the musical keyboard.

Input: A – 22.

DE – Pointer to a 9-byte buffer.

IY – MIDB pointer indicating master/slave.

Output: The buffer pointed to by DE contains the following structure, where a pressed key corresponds to a set bit:

|     | bit7 | bit6 | bit5 | bit4 | bit3 | bit2 | bit1 | bit0 |
|-----|------|------|------|------|------|------|------|------|
| 0 → | 0    | C    | B    | A#   | 0    | A    | G#   | G    |
| 1 → | 0    | F#   | F    | E    | 0    | D#   | D    | C#   |
| 2 → | 0    | C    | B    | A#   | 0    | A    | G#   | G    |
| 3 → | 0    | F#   | F    | E    | 0    | D#   | D    | C#   |
| 4 → | 0    | C    | B    | A#   | 0    | A    | G#   | G    |
| 5 → | 0    | F#   | F    | E    | 0    | D#   | D    | C#   |
| 6 → | 0    | C    | B    | A#   | 0    | A    | G#   | G    |
| 7 → | 0    | F#   | F    | E    | 0    | D#   | D    | C#   |
| 8 → | 0    | C    | 0    | 0    | 0    | 0    | 0    | 0    |

Note: The note “C” (do) of the byte 8 corresponds to the second octave and “C” of the byte 0 corresponds to the sixth octave.


Registers: All.

### RMA\_LFO (00AEH/MBIOS)

Function: Sets vibrato and tremolo levels.

Input: A – 23.

|     | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
|-----|----|----|----|----|----|----|----|----|
| C – | 0  | 0  | 0  | 0  | 0  | 0  | V  | T  |


 Tremolo: 0=1dB, 1=4,8 dB  
 Vibrato: 0=7%, 1=14%

IY – MIDB pointer indicating master/slave.

Output: None.

Registers: All.

### RMA\_TRANS (00AEH/MBIOS)

Function: Sets the transition from the current tone to the subsequent tone.

Input: A – 24.

DE – Transposition value, in units corresponding to 1% of 100/256 (~0.0039).

IY – MIDB pointer indicating master/slave.

Output: None.

Registers: All.

### **RM\_UTEMPR** (00AEH/MBIOS)

Function: Converts the temper tone to the set temper tone.

Input: A – 28.

D – Interval (The middle do (C) note is 60).

Output: DE – Converted tone.

Registers: All.

### **RM\_CTEMPR** (00AEH/MBIOS)

Function: Selects the temper.

Input: A – 29.

C – Temper:

00 – Pythagoras

01 – Meanone

02 – Werk Meister

03 – Werk Meister (modified)

04 – Werk Meister (another)

05 – Kirunker

06 – Kirunberger (modified)

07 – Valory Young

08 – Lamoo

09 – Balanced temperament (initial value)

10 – C (C minor)

11 – C# (C major)

12 – D (D minor)

13 – D# (D major)

14 – E (mi)

15 – F (F minor)

16 – F# (F major)

17 – G (minor G)

18 – G# (greater sun)

19 – A (minor A)

20 – A# (major)

21 – B (sol)

Output: None.

Registers: All.

**RM\_PITCH** (00AEH/MBIOS)

Function: Adjusts the pitch of the current and subsequent notes. The pitch must be in the range 410~459, the initial value is 440.

Input: A – 30.  
BC – Master channel.  
DE – Slave channel.

Output: None.

Registers: All.

**RM\_TRANS** (00AEH/MBIOS)

Function: Sets the transition between the current tone and the subsequent tone. The transition value must be in the range –12,799 to +12,799 where the initial value is 0 and the values are in hundredths.

Input: A – 31.  
BC – Master channel.  
DE – Slave channel.

Output: None.

Registers: All.

**RI\_DAMP** (00AEH/MBIOS)

Function: Force all FM channels to stop.

Input: A – 48.

Output: None.

Registers: All.

**RI\_ALLOFF** (00AEH/MBIOS)

Function: Turns off all assigned FM channels.

Input: A – 49.

Output: None.

Registers: All.

**RI\_EVENT** (00AEH/MBIOS)

Function: Convert pitch and turn FM voice on or off.

Input: A – 50.  
D – On: interval + 80H (central value: 60).  
Off: interval (central value: 60).

Output: None.

Registers: All.

**RI\_PCHB** (00AEH/MBIOS)

Function: Sets the pitch bender position.

Input: A – 51.

DE – Pitch bender position (the 16 bits are valid in 2's complement, where 7FFFH defines the highest position, 0 the center and 8000H the lowest position)

Output: None.

Registers: All.

Note: This function calls RCA\_PARAM (38) internally.

**RI\_PCHBR** (00AEH/MBIOS)

Function: Sets the degree to which the pitch bender will give the pitch.

Input: A – 52.

C – Degree (0 to 12 times).

Output: None.

Registers: All.

**RIA\_PARAM** (00AEH/MBIOS)

Function: Adjusts real-time parameters for active FM voice. The parameters that can be adjusted by this function are YCA\_TRANS and YCA\_VOL.

Input: A – 53.

IY – MIDB pointer indicating master/slave.

C – Parameter offset in CHDB.

DE – Configuration parameters.

Output: None.

Registers: All.

**RIA\_VOICE** (00AEH/MBIOS)

Function: Assigns an instrument number to an FM channel.

Input: A – 54.

IY – MIDB pointer indicating master/slave and for the FM voice to be assigned.

C – Instrument number (0 to 63).

Output: None.

Registers: All.



**RIA\_VPARAM** (00AEH/MBIOS)

Function: Sets the parameters of an FM channel.

Input: A – 55.

IY – MIDB pointer indicating master/slave and for the FM voice to be assigned.

C – Parameter offset in CHDB.

DE – Configuration parameters.

The following parameters can be set by this function:

|           |            |             |
|-----------|------------|-------------|
| CAO0_LS   | CAO1_LS    | YCAO0_MULT1 |
| CAO0_AR   | YCAO1_AR   | CAO1_MULT1  |
| CAO0_RR   | YCAO1_RR   | YCA_VTRANS  |
| CAO0_VELS | YCAO1_VELS | YCA_FB      |
| CAO0_VTL  | YCAO1_VTL  |             |

Output: None.

Registers: All.

**RIA\_VOICEP** (00AEH/MBIOS)

Function: Sets an instrument to an FM channel.

Input: A – 56.

IY – MIDB pointer indicating master/slave and for the FM voice to be defined.

BC – Instrument data address.

Output: None.

Registers: All.

**RM\_MOVE\_DI** (00AEH/MBIOS)

Function: Transfer PCM/ADPCM data between devices.

Input: A – 0.

IX – PDB address indicating the origin. The following fields are relevant:

PDB\_DEV (Device Number)

PDB\_ADDR (Starting address)

PDB\_SIZE (Transfer Data Size)

IY – PDB address indicating the destination. The following fields are relevant:

PDB\_DEV (Device Number)

PDB\_ADDR (Starting address)

Output: CY = 1 → transfer error.

Registers: All.

**RM\_READ\_DI** (00AEH/MBIOS)

Function: Transfers 256 bytes of PCM/ADPCM data to RAM.

Input: A – 26

DE – Destination address in RAM.

IX – PDB address indicating the origin. The following fields are relevant:

PDB\_DEV (Device Number)

PDB\_ADDR (Starting address)

Output: CY = 1 → transfer error.

Registers: All.

**RM\_WRITE\_DI** (00AEH/MBIOS)

Function: Transfers 256 bytes of data from RAM to PCM/ADPCM.

Input: A – 27.

DE – Source address in RAM.

IX – PDB address indicating the destination. The following fields are relevant:

PDB\_DEV (Device Number)

PDB\_ADDR (Starting address)

Output: CY = 1 → transfer error.

Registers: All.

**RM\_TRACE\_DI** (00AEH/MBIOS)

Function: Track ADPCM data based on initial prediction value and quantize width to find the next predicted value and next quantize width.

Input: A – 1.

C – Mode to start tracking:

0 – initial forecast at 8000H and quantization width at 007FH. The following data must be specified in the PDB:

PDB\_DEV (device number)

PDB\_ADDR (initial address)

PDB\_SIZE (transfer size)

1 – initial prediction and quantization width specified in the PDB. In addition to the data for C=0, the following must also be specified:

PDB\_PCM (initial predicted value)

PDB\_STEP (initial quantization width)

Output: The following fields return valid in the PDB:

PDB\_ADDR (next start address)  
 PDB\_PCM (next expected value)  
 PDB\_STEP (next quantization width)  
 If CY = 1, there was an error in the trace.

Registers: All.

### **RM\_CONV\_PCM\_DI (00AEH/MBIOS)**

Function: Convert ADPCM data to PCM data based on initial prediction value and quantize width.

Input: A - 2.

C - Mode to start tracking:

0 → initial forecast at 8000H and quantization width at 007FH.  
 1 → initial prediction and quantization width specified in the PDB.

IX - Source PDB address with ADPCM data. The following fields must be completed:

PDB\_DEV (device number)  
 PDB\_ADDR (initial address)  
 PDB\_SIZE (conversion value)  
 PDB\_SAMPLE (sampling frequency)

• If C=1, also fill in:

PDB\_PCM (initial predicted value)  
 PDB\_STEP (initial quantization width)

IY - Destination PDB address with PCM data. The following fields must be completed:

PDB\_DEV (device number)  
 PDB\_ADDR (initial address)

Output: The following fields return valid in the source PDB:

PDB\_PCM (next expected value)  
 PDB\_STEP (next quantization width)  
 • If CY = 1, there was an error in the conversion.

Registers: All.

### **RM\_CONV\_ADPCM\_DI (00AEH/MBIOS)**

Function: Convert PCM data to ADPCM data based on initial prediction value and quantize width for ADPCM data.

- Input:
- A – 3.
  - C – Mode to start tracking:
    - 0 → initial forecast at 8000H and quantization width at 007FH.
    - 1 → initial prediction and quantization width specified in the PDB.
  - IX – Address of the source PDB with PCM data. The following fields must be completed:
    - PDB\_DEV (device number)
    - PDB\_ADDR (initial address)
    - PDB\_SIZE (conversion value)
    - PDB\_SAMPLE (sampling frequency)
      - If C=1, also fill in:
        - PDB\_PCM (initial predicted value)
        - PDB\_STEP (initial quantization width)
  - IY – Destination PDB address with ADPCM data. The following fields must be completed:
    - PDB\_DEV (device number)
    - PDB\_ADDR (initial address)
- Output: The following fields return valid in the source PDB:
- PDB\_SIZE (size after conversion)
  - PDB\_SAMPLE (copy of sampling freq. from PCM source)
  - PDB\_PCM (next expected value)
  - PDB\_STEP (next quantization width)
    - If CY = 1, there was an error in the conversion.
- Registers: All.

### **RM\_DAC\_BIAS** (00AEH/MBIOS)

Function: Sets the volume for PCM playback (sets the 17H register of the Y8950).

- Input:
- A – 4.
  - IY – MIDB pointer indicating master/slave and PCM channel (device) 0 or 1.
  - C – Volume (1 to 7). Volume 7 is the maximum.

Output: None.

Registers: All.

### **RMA\_DAC\_DI** (00AEH/MBIOS)

Function: Play PCM data.

Input: A – 5.  
 IY – Pointer to MIDB indicating master/slave.  
 C – Filter specification (see ZMA\_PH\_FILTER).  
 IX – PDB address with reproduction data. The following fields must be defined:  
     PDB\_DEV (device number)  
     PDB\_ADDR (initial address)  
     PDB\_SIZE (size)  
     PDB\_SAMPLE (sampling frequency)

Output: CY = 1 → playback error.

Registers: All.

### **RMA\_ADC\_DI (00AEH/MBIOS)**

Function: Write PCM data.

Input: A – 6.  
 IY – Pointer to MIDB indicating master/slave.  
 C – Filter specification (see ZMA\_PH\_FILTER).  
 IX – PDB address with recording data. The following fields must be defined:  
     PDB\_DEV (device number)  
     PDB\_ADDR (initial address)  
     PDB\_SIZE (size)  
     PDB\_SAMPLE (sampling frequency)

Output: CY = 1 → write error.

Registers: All.

### **RMA\_ADPCM\_BIAS (00AEH/MBIOS)**

Function: Sets the volume for ADPCM playback.

Input: A – 7.  
 IY – MIDB pointer indicating master/slave and PCM channel (device) 0 or 1.  
 C – Volume (0 to 63). Volume 63 is the maximum.

Output: None.

Registers: All.

### **RMA\_ADPLAY\_DI (00AEH/MBIOS)**

Function: Play ADPCM data in non-local mode.

Input: A – 8.  
 IY – Pointer to MIDB indicating master/slave.  
 C – Filter specification (see ZMA\_PH\_FILTER).

IX – PDB address with reproduction data. The following fields must be defined:

PDB\_DEV (device number)

PDB\_ADDR (initial address)

PDB\_SIZE (size)

PDB\_SAMPLE (sampling frequency)

Output: CY = 1 → playback error.

Registers: All.

### **RMA\_ADPLAY\_DI** (00AEH/MBIOS)

Function: Record ADPCM audio in non-local mode.

Input: A – 9.

IY – Pointer to MIDB indicating master/slave.

C – Filter specification (see ZMA\_PH\_FILTER).

IX – PDB address with recording data. The following fields must be defined:

PDB\_DEV (device number)

PDB\_ADDR (initial address)

PDB\_SIZE (size)

PDB\_SAMPLE (sampling frequency)

Output: CY = 1 → write error.

Registers: All.

### **RMA\_ADPAY\_SAMPLE** (00AEH/MBIOS)

Function: Changes the sampling frequency during playback in local mode.

Input: A – 43.

IY – Pointer to MIDB indicating master/slave.

DE – Frequency of sampling.

Output: None.

Registers: All.

### **RMA\_BREAK** (00AEH/MBIOS)

Function: Stop recording or playback in local mode.

Input: A – 10.

IY – Pointer to MIDB indicating master/slave.

Output: None.

Registers: All.

### **RMA\_ADPLAY** (00AEH/MBIOS)

Function: Play ADPCM data in local mode.

Input: A – 11.  
 IY – Pointer to MIDB indicating master/slave.  
 C – Filter specification (see ZMA\_PH\_FILTER).  
 IX – PDB address with reproduction data. The following fields must be defined:  
     PDB\_DEV (device number)  
     PDB\_ADDR (initial address)  
     PDB\_SIZE (size)  
     PDB\_SAMPLE (sampling frequency)

Output: CY = 1 → playback error.

Registers: All.

### **RMA\_ADREC (00AEH/MBIOS)**

Function: Play ADPCM data in local mode.

Input: A – 12.  
 IY – Pointer to MIDB indicating master/slave.  
 C – Filter specification (see ZMA\_PH\_FILTER).  
 IX – PDB address with recording data. The following fields must be defined:  
     PDB\_DEV (device number)  
     PDB\_ADDR (initial address)  
     PDB\_SIZE (size)  
     PDB\_SAMPLE (sampling frequency)

Output: CY = 1 → write error.

Registers: All.

### **RMA\_ADPLAYLP (00AEH/MBIOS)**

Function: Play ADPCM data in local loop mode. At the end, playback resumes indefinitely. To break, execute RMA\_BREAK (Function 10).

Input: A – 42.  
 IY – Pointer to MIDB indicating master/slave.  
 C – Filter specification (see ZMA\_PH\_FILTER).  
 IX – PDB address with reproduction data. The following fields must be defined:  
     PDB\_DEV (device number)  
     PDB\_ADDR (initial address)  
     PDB\_SIZE (size)  
     PDB\_SAMPLE (sampling frequency)

Output: CY = 1 → playback error.

Registers: All.

**RMA\_PHASE\_SET\_DI (00AEH/MBIOS)**

Function: Take 256 bytes of PCM data in main RAM as waveform data, convert to ADPCM data and store in external RAM.

| Ext RAM adr | Pitch   | No. waveforms |
|-------------|---------|---------------|
| 0000H~07FFH | 24H~36H | 16            |
| 0800H~0FFFH | 37H~42H | 32            |
| 1000H~17FFH | 43H~4EH | 64            |
| 1800H~1FFFH | 4FH~5AH | 128           |

Input: A - 13.  
 IY - Pointer to MIDB indicating master/slave.  
 C - Filter specification (see ZMA\_PH\_FILTER).  
 DE - PCM data address.

Output: None.

Registers: All.

Note: Before conversion, RMA\_BREAK (Func. 10) is executed.

**RMA\_PHASE\_EG (00AEH/MBIOS)**

Function: Define the wrap data.

Input: A - 14.  
 IY - Pointer to MIDB indicating master/slave.  
 DE - Envelope data address (7 bytes):  
   +0 Timer#1 value  
   +1 Total level  
   +2 Attack rate  
   +3 Decay rate#1  
   +4 Sustain Level  
   +5 Decay rate#2  
   +6 Release rate

Output: None.

Registers: All.

**RMA\_PHASE\_EVENT (00AEH/MBIOS)**

Function: Turn on sampling of the specified tone or turn off keyboard sampling simulation.

Input: A - 115.  
 IY - Pointer to MIDB indicating master/slave.  
 D - On: interval + 80H (central value: 60).  
   Off: interval (central value: 60).  
 Valid range is 24H~5AH.

Output: None.

Registers: All.



**RMA\_CSM\_DI** (00AEH/MBIOS)

Function: CSM data playback.

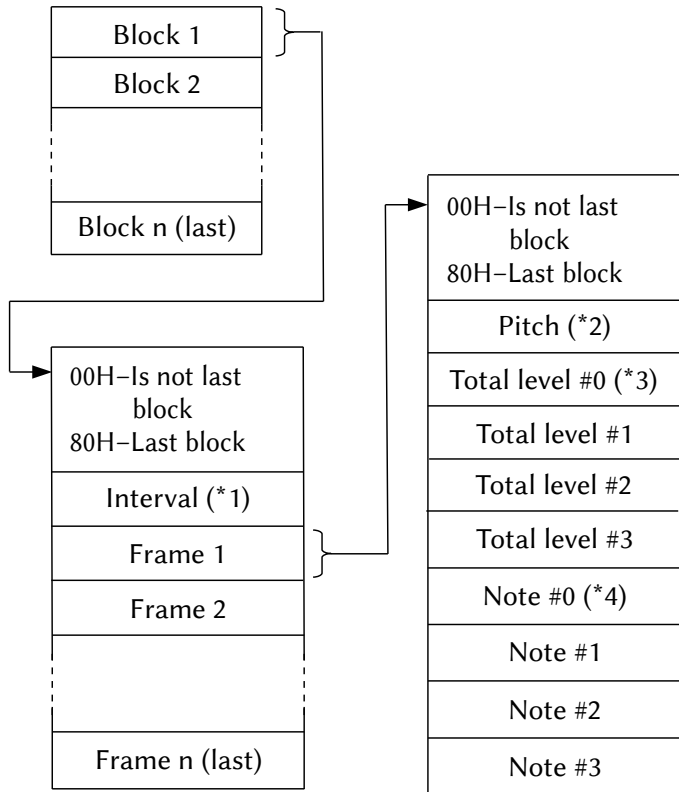
Input: A – 25.

IY – Pointer to MIDB indicating master/slave.

B – Volume (0 to 127, where 0 is the maximum volume).

C – Filter specification (see ZMA\_PH\_FILTER).

DE – Address of the CSM data, which have the following structure:

(\*1) Interval: Timer 2  $\rightarrow 0 = 81.9 \text{ mS} / 255 = 0.32 \text{ mS}$ (\*2) Pitch: Timer 1  $\rightarrow 0 = 20.9 \text{ mS} / 255 = 0.08 \text{ mS}$ 

(\*3) Total level (volume data for each channel):

0 = maximum / 127 = minimum

(\*4) Note (pitch data for each channel).

The upper 4 bits specify the octave from 0 to 7 and the lower 4 bits specify the scale. Values are:

|           |           |
|-----------|-----------|
| 00 – C#   | 08 – G    |
| 01 – D    | 09 – G#   |
| 02 – D#   | 10 – A    |
| 03 – None | 11 – None |
| 04 – E    | 12 – A#   |
| 05 – F    | 13 – B    |
| 06 – F#   | 14 – C    |
| 07 – None | 15 – None |

Output: None.

Registers: All.

#### **SV\_IRQ** (00B4H/MBIOS)

Function: MSX-Audio interrupt handling (must configure HKEYI hook (FD9AH) ).

Input: None.

Output: None.

Registers: None.

### **8.5.6 – MSX-JE**

#### **EXTBIO** (FFCAH/Work Area)

Function: Access extended BIOS functions

Input: A – 00H.

D – 16H – MSX-JE manipulation device.

E – 00H – Returns the pointer to the table of input addresses of the MSX-JE routines.

B – Address table slot ID.

HL – Address of a 64-byte buffer for the table (should be on page 3).

Output: CY = 1 → there is no MSX-JE.

CY = 0 → HL is incremented by 4 for each MSX-JE found and will point to the end of a table that reserves 4 bytes for each MSX-JE. The original value of HL points to the beginning of the table, which has the following structure:

+00H – Capacity vector

+01H – Slot ID

+02H – Lowest address

+03H – Highest address

The capacity vector byte has the following structure:

Bit 0 – 0 → MSX-JE compatible

1 → incompatible

Bit 1 – 0 → there is virtual terminal interface

1 → there is no interface

Bit 2 – 0 → dictionary interface exists

1 → there is no dictionary

Bit 3 – 0 → there is register and deletion function.

1 → there is no register and deletion function.

bit 4~bit 7 → always 0.

Slot ID (+01H) and address (+02H,+03H) specify the entry point for MSX-JE functions. The call must be made through the CALSLT (0030H) routine of the Main-ROM, placing the function number in register A.

Registers: All.

### 8.5.6.1 – Calling MSX-JE functions

#### **INQUIRY** (Function 01H / MSX-JE)

Function: Returns the size of the desktop.

Input: A – 01H.

Output: HL – Maximum desktop size limit used by MSX-JE.  
DE – Lower limit on the size of the desktop used by MSX-JE.  
BC – Minimum size required for MSX-JE to use learning function

#### **INVOKE** (Function 02H / MSX-JE)

Function: Initializes the desktop.

Input: A – 02H.

HL – Address of the desktop protected by the AP.

DE – Size of the working area protected by the AP.

Output: None.

**RELEASE** (Function 03H / MSX-JE)

Function: Frees the memory protected by the AP.

Input: A – 03H.

HL – Address of the desktop protected by the AP.

Output: None.

**CLEAR** (Function 04H / MSX-JE)

Function: Clears the buffer for Kana – Kanji conversion.

Input: A – 04H.

HL – Address of the desktop protected by the AP.

Output: None.

**SET\_TTB** (Function 05H / MSX-JE – Optional)

Function: Pass the text and read the data to be converted from the AP to the MSX-JE, configuring them in the MSX-JE's internal buffer. This function causes MSX-JE to reconvert the given text.

Input: A – 05H.

HL – Desktop address.

DE – Address of the text to be converted again.

BC – TTB (Transferable Text Block) address.

Output: A = 255 → function not supported.

**DISPATCH** (Function 06H / MSX-JE – Optional)

Function: Pass CPU control from AP to MSX-JE.

Input: A – 06H.

HL – Desktop address.

Output: HL – STB (Screen image Text Block) address.

A – Return status:

bit 0 = 1 → the AP displays the STB.

bit 1 = 1 → AP can get conversion result.

bit 2 = 1 → MSX-JE conversion finished.

Possible states with the combined bits:

000 – MSX-JE ignores the key (no key entry).

001 – Entry or conversion in progress.

01x – Partial conversion made.

10x – Conversion stopped and finished.

11x – Fully converted.

**GET\_RESULT** (Function 07H / MSX-JE)

Function: Returns the result of the conversion.

Input: A – 07.

HL – Desktop address.

Output: HL – Start address of the conversion result, ending with a 00H byte.

**GET\_TTB** (Function 08H / MSX-JE – Optional)

Function: Acquires text data obtained by GET\_RESULT.

Check-in: A – 08.

HL – Desktop address.

Output: HL – TTB (Transferable Text Block) address. If this function is not supported, (HL) will point to a 00H byte.

**INQUIRY\_WINDOW\_SIZE** (Function 09H / MSX-JE – Optional)

Function: Defines the window format.

Input: A – 09:00.

HL – desktop address.

E – maximum length of the “tail”.

B – maximum height of the window.

C – maximum width of the window.

Output: HL – address of window specification data.

+00H – Type of window:

1 – Independent.

2 – “tail”.

+01H – Window width (1~255).

+02H – Window height (1~255).

**CONFLICT\_DETECT** (Function 0AH / MSX-JE – Optional)

Function: Avoid key collision conflicts.

Input: A – 0AH.

HL – Desktop address.

Output: A – 00H → conflict not detected.

FFH → conflict detected.

**8.5.6.2 – MSX-JE dictionary interface****HAN\_ZEN** (Function 40H)

Function: Converts a one-byte character string to a two-byte character string.

Input: A – 40H.  
 HL – Desktop address.  
 DE – Source string address (one byte).  
 BC – Address of the two-byte character string.

Output: A = 0 → successful conversion.  
 A ≠ 0 → conversion error.

### **ZEN\_HAN** (Function 41H / MSX-JE)

Function: Converts a two-byte character string to a one-byte character string.

Input: A – 41H.  
 HL – Desktop address.  
 DE – Source string address (two byte).  
 BC – Address of one-byte character string.

Output: A = 0 → successful conversion.  
 A ≠ 0 → conversion error.

### **HAN\_KATA** (Function 42H / MSX-JE)

Function: Converts a one-byte character string from the roman alphabet, katakana, or a combination thereof to a two-byte katakana character string.

Input: A – 42H.  
 HL – Desktop address.  
 DE – Source string address (one byte).  
 BC – Address of the katakana character string.

Output: A = 0 → successful conversion.  
 A ≠ 0 → conversion error.

### **HAN\_HIRA** (Function 43H / MSX-JE)

Function: Converts a one-byte character string from the roman alphabet, katakana, or a combination thereof to a two-byte hiragana character string.

Input: A – 43H.  
 HL – Desktop address.  
 DE – Source string address (one byte).  
 BC – Address of the hiragana character string.

Output: A = 0 → successful conversion.  
 A ≠ 0 → conversion error.

**KATA\_HIRA** (Function 44H / MSX-JE)

Function: Converts a two-byte katakana character string to a two-byte hiragana character string.

Input: A - 44H.

HL - Desktop address.

DE - Address of the two-byte katakana string.

BC - Address of the hiragana character string.

Output: A = 0 → successful conversion.

A ≠ 0 → conversion error.

**HIRA\_KATA** (Function 45H / MSX-JE)

Function: Converts a two-byte hiragana character string to a two-byte katakana character string.

Input: A - 45H.

HL - Desktop address.

DE - Address of the two-byte katakana string.

BC - Address of the hiragana character string.

Output: None.

**OPEN\_DIC** (Function 46H / MSX-JE)

Function: Reserved for future expansions.

Input: A - 46H.

HL - Desktop address.

DE - 0000H

Output: A - Always returns 5.

**HENKAN** (Function 47H / MSX-JE)

Function: Converts a 2-byte katakana and hiragana character string to a mixed Kanji-Kana string.

Input: A - 47H.

HL - Desktop address.

DE - Address of the string katakana/hiragana.

Output: A - Number of possible conversions. If there is none, it returns 0. The converted strings must be obtained by the JI\_KOHO (48H) function.

**JI\_KOHO** (Function 48H / MSX-JE)

Function: Acquires the next conversion obtained by HENKAN (47H).

Input: A – 48H.  
 HL – Desktop address.  
 DE – Address of the next converted Kanji-Kana string.  
 BC – Secondary Kanji-Kana string address.

Output: A = 0 → No Kanji-Kana conversion acquired.  
 A > 0 → Acquired Kanji-Kana conversion number.

### **ZEN\_KOHO** (Function 49H / MSX-JE)

Function: Acquires the previous conversion obtained by HENKAN (47H).

Input: A – 49H.  
 HL – Desktop address.  
 DE – Address of the former Kanji-Kana string converted.  
 BC – Secondary Kanji-Kana string address.

Output: A = 0 → No Kanji-Kana conversion acquired.  
 A > 0 → Acquired Kanji-Kana conversion number.

### **JI\_BLOCK** (4AH Function / MSX-JE)

Function: Creates a lower priority Kanji-Kana conversion group next to the main group.

Input: A – 4AH.  
 HL – desktop address.

Output: A = 0 → group not created.  
 A > 0 → number of lowest priority Kanji-Kana conversions.

### **ZEN\_BLOCK** (4BH Function / MSX-JE)

Function: Creates a higher priority Kanji-Kana conversion group next to the main group.

Input: A – 4BH.  
 HL – Desktop address.

Output: A = 0 → Group not created.  
 A > 0 → Highest priority number of Kanji-Kana conversions.

### **KAKUTEI1** (4CH Function)

Function: Confirms the result of Kanji-Kana conversion.

Input: A – 4CH.  
 HL – Desktop address.  
 E – Kanji-Kana conversion number within the group.  
 BC – Kanji-Kana translation buffer address.

Output: BC – 0AH + "natto curry".



**KAKUTEI2** (4DH Function)

Function: Confirms the result of Kanji-Kana conversion.

Input: A - 4DH.

HL - Desktop address.

E - Kanji-Kana conversion number within the group

BC - Kanji-Kana translation buffer address.

Output: A - Size in bytes of the Kanki-Kana string.

BC - 04H + "natto".

**CLOSE\_DIC** (4EH Function)

Function: Function not implemented.

Input: A - 4EH.

Output: A - Always 0.

**TOUROKU** (4FH Function)

Function: Provides reading data, word data and part of text, and includes the specified word in the dictionary.

Input: A - 4FH.

HL - Desktop address.

DE - Read buffer address.

BC - Address of word inclusion buffer.

Output: A - 00H → word successfully added.

01H → insufficient free space.

02H → word parity overflow.

04H → incorrect reading data.

08H → incorrect word data.

10H → part of text is incorrect.

FFH → not supported.

**SAKUJO** (50H Function)

Function: Provides reading data, word data and part of text excluding the specified word from the dictionary.

Input: A - 50H.

HL - Desktop address.

DE - Read buffer address.

BC - Address of the word exclusion buffer.

Output: A - 00H → word successfully deleted.  
           01H → word to be deleted was not found.  
           04H → incorrect reading data.  
           08H → incorrect word data.  
           10H → part of text is incorrect.  
           FFH → not supported.

### 8.5.7 – MSX UNAPI

#### **EXTBIO** (FFCAH/Work Area)

Function: Accesses extended BIOS functions.

Input: A = 00H – Gets the number of implementations of the specified API.  
       A > 00H → Returns the parameters of the specified API.  
       D = 22H → MSX UNAPI manipulation device.  
       E = 22H → Returns data from the specified API.  
       (F487H) – API specification identifier, which must be an alphanumeric string of up to 15 characters ending in 00H, not case sensitive.

Output: A = 00H → B – Number of implementations of the specified API.  
       A > 00H → A – Implementation routine slot ID.  
       B – Implementation mapper segment  
           (FFH = not in the mapper).  
       HL – Entry point address of the implementation routines  
           (if it is on physical page 3, the values of A and B are disregarded).

Registers: AF, BC, HL.

#### 8.5.7.1 – RAM Helper

#### **EXTBIO** (FFCAH/Work Area)

Function: Accesses extended BIOS functions.

Input: A = FFH → API: RAM helper  
       D = 22H → MSX UNAPI manipulation device.  
       E = 22H → Returns API parameters.  
       HL = 0000H

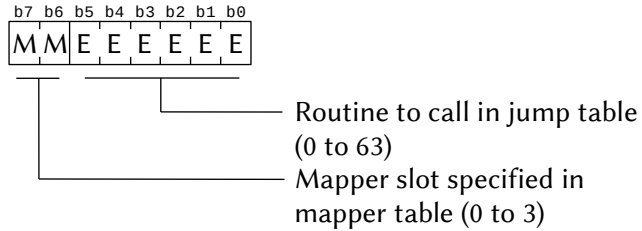
Output: HL = 0000H → RAM helper not installed.  
 HL > 0000H → HL = Jump table address on page 3.  
 BC – Address of the mapping table.  
 A – Number of entries in the jump table, which has the following structure:  
     +00H CALMAP calls routine mapper  
     +03H RDBYTE reads byte from RAM  
     +06H CALSEG calls routine in RAM

Registers: AF, BC, HL.

**CALMAP** (HL+00H) – HL value obtained via EXTBIO  
 Function: Calls a routine on a mapped RAM segment.  
 Input: IYh – Slot ID.  
       IYl – Mapper segment number.  
       IX – Routine address (must be on physical page 1).  
       AF, BC, DE, HL – Parameters for the called routine.  
 Output: AF, BC, DE, HL, IX, IY – Routine return parameters.  
 Registers: Depends on the called routine.

**RDBYTE** (HL+03H) – HL value obtained via EXTBIO  
 Function: Reads a byte from a segment of the mapped RAM.  
 Input: A – Slot ID.  
       B – Segment number.  
       HL – Address to be read (highest two bits are ignored).  
 Output: A – byte read at specified address.  
 Registers: A.

**CALSEG** (HL+06H) – HL value obtained via EXTBIO  
 Function: Calls a routine a segment of the mapped RAM using inline parameters.  
 Input: AF, BC, DE and HL can contain parameters for the called routine (do not use IX and IY).  
       Inline call parameters, in the following format:  
         CALL <routine address>  
         DB <routine ID>  
         DB <segment number>  
       Routine ID:



- The jump table starts at address 4000H, where index 0 means 4000H, index 1 means 4003H, and so on up to the value 63, every three bytes.
- The mapper table occupies 8 bytes, reserving two bytes for each mapper, being able to manage up to 4 mappers (0 to 3), and has the following structure:
  - +0 – Slot ID of first mapper
  - +1 – Number segments available in the 1st mapper
  - +2 – Second mapper slot ID
  - +3 – Number segments available in the 2nd mapper
  - +4 – Third mapper slot ID
  - +5 – Number segments available on the 3rd mapper
  - +6 – Fourth mapper slot ID
  - +7 – Number segments available on the 4th mapper

Note: if the mapper has 4 Mbytes, the number of segments will be FEH, since the FFH value has been reserved for the system.

Output: AF, BC, DE, HL, IX and IY can contain valid values.

Registers: Depends on the called routine.

### 8.5.7.2 – API for Ethernet cartridges

#### EXTBIO (FFCAH/Work Area)

Function: Accesses extended BIOS functions.

Input: A = 00H → Gets the number of implementations of the specified API.

A > 00H → Returns the parameters of the specified API.

D = 22H → MSX UNAPI manipulation device.

E = 22H → Returns data from the specified API.

(F487H) – "ETHERNET"

Output: A = 00H → B – Number of API implementations.  
 A > 00H → A – Implementation routine slot ID.  
 B – Implementation mapper segment  
 (FFH = not in the mapper).  
 HL – Entry point address of the implementation routines  
 (if it is on physical page 3, the values of A and B are  
 disregarded).  
 Registers: AF, BC, HL.

**ETH\_GETINFO** (HL/ExtBIOS) – HL value obtained via EXTBIOS

Function: Returns the version and name of the implementation.

Input: A = 0.

Output: HL – Implementation name string address.  
 B – Version of the API implementation (primary).  
 C – Version of the API implementation (secondary).  
 D – API version specification (primary).  
 E – API version specification (secondary).

Registers: All.

**ETH\_RESET** (HL/ExtBIOS) – HL value obtained via EXTBIOS

Function: Returns the hardware and state variables to their initial  
 condition (condition right after computer reset).

Input: A = 1.

Output: None.

Registers: All.

**ETH\_GET\_HWADD** (HL/ExtBIOS) – HL value obtained via EXTBIOS

Function: Returns Ethernet address.

Input: A = 2.

Output: L-H-E-D-C-B – Address.

Registers: All.

**ETH\_GET\_NETSTAT** (HL/ExtBIOS) – HL value obtained via EXTBIOS

Function: Checks the network connection status.

Input: A = 3.

Output: A = 0 → no connection to an active network.  
 1 → there is an active network connection.

Registers: All.

**ETH\_NET\_ONOFF** (HL/ExtBIOS) – HL value obtained via EXTBIOS

Function: Enables or disables the network.

Input: A = 4.

B = 0 → returns the current networking state.

1 → enable networking.

2 → disable networking.

Output: A = 1 → network enabled.

2 → network disabled.

Registers: All.

**ETH\_DUPLEX** (HL/ExtBIOS) – HL value obtained via EXTBIOS

Function: Sets duplex mode.

Input: A = 5.

B = 0 → returns to current mode.

1 → select half-duplex mode.

2 → selects full-duplex mode.

Output: A = 1 → half-duplex mode selected.

2 → half-duplex mode selected.

3 → unknown mode or duplex mode is not applicable.

**ETH\_FILTERS** (HL/ExtBIOS) – HL value obtained via EXTBIOS

Function: Configures frame reception filters.

Input: A = 6.

B – bit 7 – 0 → no action.

1 → returns current setting.

bit 6 – reserved.

bit 5 – reserved.

bit 4 – 0 → disable promiscuous mode.

1 → enable promiscuous mode.

bit 3 – reserved.

bit 2 – 0 → reject “broadcast” frames.

1 → accepts “broadcast” frames.

bit 1 – 0 → reject frames smaller than 64 bytes.

1 → accept frames smaller than 64 bytes.

bit 0 – Reserved.

Output: A – filter configuration after execution (same format as register B on input).

Registers: All.

**ETH\_IN\_STATUS** (HL/ExtBIOS) – HL value obtained via EXTBIOS

Function: Checks the availability of received frames.

Input: A = 7.

Output: A – 0 → No incoming frames available.

1 → At least one received frame is available.

BC – Oldest frame size available.

HL – Bytes 12 and 13 of the oldest frame available.

Registers: All.

**ETH\_GET\_FRAME** (HL/ExtBIOS) – HL value obtained via EXTBIOS

Function: Recovers the oldest frame.

Input: A = 8.

HL = 0 → Discard the frame.

Other value → frame destination address.

Output: A – 0 → Frame retrieved or discarded.

1 → There are no received frames available.

BC – Retrieved frame size.

**ETH\_SEND\_FRAME** (HL/ExtBIOS) – HL value obtained via EXTBIOS

Function: Sends a frame.

Input: A = 9.

HL – Destination address of the frame in memory.

BC – Frame size.

D – Execution mode: 0 – Synchronous.

1 – Asynchronous.

Output: A – 0 → Frame sent or transmission started.

1 → Invalid frame size.

2 → Ignored.

3 → Lost carrier.

4 → Excessive number of collisions.

5 → Asynchronous mode not supported.

Registers: All.

**ETH\_OUT\_STATUS** (HL/ExtBIOS) – HL value obtained via EXTBIOS

Function: Recovers the oldest frame.

Input: A = 10.

Output: A – 0 → no frame sent since last reset.

1 → transmitting at this time.

- 2 → transmission completed successfully.
- 3 → lost carrier.
- 4 → excessive number of collisions.

Registers: All.

### **ETH\_SET\_HWADD** (HL/ExtBIOS) – HL value obtained via EXTBIO

Function: Selects Ethernet address.

Input: A = 11.

L-H-E-D-C-B – Ethernet address to be set.

Output: L-H-E-D-C-B – Ethernet address after execution.

Registers: All.

## **8.5.8 – MemMan**

### **EXTBIO** (FFCAH/Work Area)

Function: Accesses extended BIOS functions.

Input: A = 00H

D = 4DH – MEMMAN manipulation device.

E = 32H – Returns information about alternative inputs for MemMan functions.

B = 0 → Input address for FastUse0 (func. 0)

1 → Input address for FastUse1 (func. 1)

2 → Input address for FastUse2 (func. 2)

3 → Input address for FastTsrCall (fn. 63)

4 → Input address for BasicCall

5 → Input address for FastCurSeg (fn. 32)

6 → Input address for handler of MemMan functions

7 → Returns MemMan version (VerMM:#H.L)

8 → Input address for FastXTsrCall (f. 61)

Output: HL – Address or version.

Registers: All.

#### **8.5.8.1 – Fast Calls (Preferred alternative entries)**

### **FastUse0** (HL/ExtBIOS) – HL value obtained via EXTBIO

Function: Enables a segment on physical page 0 (0000H~3FFFH).

Enabling is only possible if the segment contains the entry points to the standard slot switching routines.



Input: HL – Segment number.  
 Output: A – 00H → segment enabled successfully.  
           FFH → segment enable failed.  
 Note: This function is identical to function 0 (Use0).

**FastUse1** (HL/ExtBIOS) – HL value obtained via EXTBIO  
 Function: Enables a segment on physical page 1 (4000H~7FFFH).  
 Input: HL – Segment number.  
 Output: A – 00H → segment enabled successfully.  
           FFH → segment enable failed.  
 Note: This function is identical to function 1 (Use1).

**FastUse2** (HL/ExtBIOS) – HL value obtained via EXTBIO  
 Function: Enables a segment on physical page 2(8000H~BFFFFH).  
 Input: HL – Segment number.  
 Output: A – 00H → segment enabled successfully.  
           FH → segment enable failed.  
 Note: This function is identical to function 2 (Use2).

**FastTsrCall** (HL/ExtBIOS) – HL value obtained via EXTBIO  
 Function: Calls a TSR driver routine.  
 Input: BC – TSR function ID code.  
       AF, DE, HL – Parameters for the TSR.  
 Output: AF, BC, DE, HL – TSR return parameters.  
 Note: This function is identical to function 63 (TsrCall), except here the DE register can be used without problems.

**BasicCall** (HL/ExtBIOS) – HL value obtained via EXTBIO  
 Function: Calls a routine from Main-ROM.  
 Input: IX – Routine address on physical page 0 or 1.  
       AF, BC, DE, HL – Parameters to be passed to the routine.  
 Output: AF, BC, DE, HL – Routine return parameters.  
 Note: Interrupts are disabled.

**FastCurSeg** (HL/ExtBIOS) – HL value obtained via EXTBIO  
 Function: Returns the current segment number of a page.  
 Input: B – Physical page (0, 1, 2 or 3).

Output: HL – Segment number.  
 A – Type of segment: 00H → PSEG.  
 FFH → FSEG.  
 Note: This function is identical to function 32 (CurSeg).

**MemMan** (HL/ExtBIOS) – HL value obtained via EXTBIO  
 Function: Directly call a MemMan function.  
 Input: E – Function number.  
 AF, BC, HL – Parameters to be passed to the routine.  
 Output: AF, BC, DE, HL – Routine return parameters.

**VerMM** (HL/ExtBIOS) – HL value obtained via EXTBIO  
 Function: Returns the MemMan version number.  
 Input: None.  
 Output: HL – “H.L” format version.

**FastXTsrCall** (HL/ExtBIOS) – HL value obtained via EXTBIO  
 Function: Calls driver input from a TSR.  
 Input: IX – ID code of the called TSR input.  
 AF, BC, DE, HL – Parameters to be passed to the routine.  
 Output: AF, BC, DE, HL – Routine return parameters.  
 Note: This function is identical to function 61 (XtrsCall).

### 8.5.8.2 – MemMan Functions

**Use0** (FFCAH/Work Area) – Execution via EXTBIO  
 Function: Accesses extended BIOS functions.  
 Input: A = 00H.  
 D = 4DH – MEMMAN manipulation device.  
 E = 00H – Use0 function. Enables a segment on physical page 0 (0000H~3FFFH). Enabling is only possible if the segment contains the entry points to the standard slot switching routines.  
 HL – segment number.  
 Output: A – 00H → segment enabled successfully.  
 FFH → segment enable failed.  
 Note: Preferably use the FastUse0 input of the 32H function (Info) of MemMan.

**Use1** (FFCAH/Work Area) – Execution via EXTBIOS

Function: Accesses extended BIOS functions.

Input: A = 00H.

D = 4DH – MEMMAN manipulation device.

E = 01H – Use0 function. Enables a segment on physical page 1 (4000H~7FFFH).

HL – segment number.

Output: A – 00H → segment enabled successfully.

FFH → segment enable failed.

Note: Preferably use the FastUse1 input of the 32H function (Info) of MemMan.

**Use2** (FFCAH/Work Area) – Execution via EXTBIOS

Function: Accesses extended BIOS functions.

Input: A = 00H.

D = 4DH – MEMMAN manipulation device.

E = 02H – Use2 function. Enables a segment on physical page 2 (8000H~BFFFFH).

HL – segment number.

Output: A – 00H → segment enabled successfully.

FFH → segment enable failed.

Note: Preferably use the FastUse2 input of the 32H function (Info) of MemMan.

**Alloc** (FFCAH/Work Area) – Execution via EXTBIOS

Function: Accesses extended BIOS functions.

Input: A = 00H.

D = 4DH – MEMMAN manipulation device.

E = 0AH – Alloc function. Allocates a segment.

B – Segment preference code:

| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
|----|----|----|----|----|----|----|----|
| P  | T  | 0  | 0  | 0  | 0  | S  | S  |

Segment type:

00–PSEG0000 10–PSEG8000

01–PSEG4000 11–FSEG

1 – Prefer TPA (Standard MSXDOS RAM).

1 – Prefer not expanded slot.

Output: HL – Segment number (0000H → no segments free).

- SetRes** (FFCAH/Work Area) – Execution via EXTBIO  
 Function: Accesses extended BIOS functions.  
 Input: A = 00H.  
       D = 4DH – MEMMAN manipulation device.  
       E = 0BH – SetRes function. Assigns to a segment the  
           “reserved” status.  
       HL – Segment number.  
 Output: None.
- DeAlloc** (FFCAH/Work Area) – Execution via EXTBIO  
 Function: Accesses extended BIOS functions.  
 Input: A = 00H  
       D = 4DH – MEMMAN manipulation device.  
       E = 14H – DeAlloc function. Frees a segment.  
       HL – Segment number.  
 Output: None.
- IniChk** (FFCAH/Work Area) – Execution via EXTBIO  
 Function: Accesses extended BIOS functions.  
 Input: A = Control code.  
       D = 4DH – MEMMAN manipulation device.  
       E = 1EH – IniChk function. Start MemMan before  
           a program.  
 Output: A – Control code + “M”.  
       DE – Version number in “D:E” format.
- Status** (FFCAH/Work Area) – Execution via EXTBIO  
 Function: Accesses extended BIOS functions.  
 Input: A = 00H.  
       D = 4DH – MEMMAN manipulation device.  
       E = 1FH – Status Function. Return status information  
           of MemMan.  
 Output: HL – Number if segments available.  
       BC – Number of free segments.  
       DE – Number of segments controlled simultaneously  
           through MemMan and DOS2.  
       A – Hardware connection status:  
           bit0 – 0 → DOS2 mapper support not available.  
                   1 → DOS2 mapper support installed.  
           bit1~bit 7 → always 0.

**CurSeg** (FFCAH/Work Area) – Execution via EXTBIOS

Function: Accesses extended BIOS functions.

Input: A = 00H.

D = 4DH – MEMMAN manipulation device.

E = 20H – CurSeg function. Returns the segment number on a page.

B – Physical page number (0, 1, 2 or 3).

Output: HL – Segment number.

A – Type of segment: 00H → PSEG.

FFH → FSEG.

Note: Preferably use the FastCurSeg entry of the 32H (Info) function of MemMan.

**StoSeg** (FFCAH/Work Area) – Execution via EXTBIOS

Function: Accesses extended BIOS functions.

Input: A = 00H.

D = 4DH – MEMMAN manipulation device.

E = 28H – StoSeg function. Stores the state of the current segment.

HL – Address of a 9-byte buffer.

Output: None.

**RstSeg** (FFCAH/Work Area) – Execution via EXTBIOS

Function: Accesses extended BIOS functions.

Input: A = 00H.

D = 4DH – MEMMAN manipulation device.

E = 29H – RstSeg function. Reactivates the state of a segment that has been stored.

HL – Address of a 9-byte state buffer.

Output: None.

**XtsrCall** (FFCAH/Work Area) – Execution via EXTBIOS

Function: Accesses extended BIOS functions.

Input: A = 00H.

D = 4DH – MEMMAN manipulation device.

E = 3DH – XTsrCall function. Call an entry from the TSR driver.

IX = ID code of the called TSR input.

AF, BC, HL – Parameters to be passed to the routine.

Output: AF, BC, DE, HL – Routine return parameters.  
 Note: Preferably use the FastXtrsCall entry of the 32H (Info) function of MemMan.

**GetTsrID** (FFCAH/Work Area) – Execution via EXTPIO

Function: Accesses extended BIOS functions.

Input: A = 00H.  
 D = 4DH – MEMMAN manipulation device.  
 E = 3EH – GetTsrID function. Determines the TSR ID code.  
 HL – Pointer to TsrName. Unused positions must be padded with spaces.  
 Output: CY = 0 → Not found; 1 → ID found.  
 BC → TSR ID code.

**TsrCall** (FFCAH/Work Area) – Execution via EXTPIO

Function: Accesses extended BIOS functions.

Input: A = 00H.  
 D = 4DH – MEMMAN manipulation device.  
 E = 3FH – TsrCall function. Call an entry from the TSR driver.  
 BC – ID code of the called TSR input.  
 AF, HL – Parameters to be passed to the routine.  
 Output: AF, BC, DE, HL – Routine return parameters.  
 Note: Preferably use the FastTrsCall entry of the 32H (Info) function of MemMan.

**HeapAlloc** (FFCAH/Work Area) – Execution via EXTPIO

Function: Accesses extended BIOS functions.

Input: A = 00H.  
 D = 4DH – MEMMAN manipulation device.  
 E = 46H – HeapAlloc function. Allocates space in the “heap”.  
 HL – Size of the space to be allocated.  
 Output: HL = 0000H → insufficient memory for allocation.  
 Other value → start address of allocated space.

**HeapDeAlloc** (FFCAH/Work Area) – Execution via EXTPIO

Function: Accesses extended BIOS functions.

Input: A = 00H.  
 D = 4DH – MEMMAN manipulation device.  
 E = 47H – HeapDeAlloc Function. Frees up space allocated in the “heap”.  
 HL – Size of the space to be allocated.

Output: None.

**HeapMax** (FFCAH/Work Area) – Execution via EXT BIO

Function: Accesses extended BIOS functions.

Input: A = 00H.  
 D = 4DH – MEMMAN manipulation device.  
 E = 48H – HeapMax function. Returns the maximum size of space available in the “heap”.  
 Output: HL – Available space in the “heap”.

### 8.5.9 – System commands

**EXT BIO** (FFCAH/Work Area)

Function: Access extended BIOS functions

Input: A = 00H.  
 D = FFH – System Device.  
 E = 00H – Returns the starting address, slot ID and manufacturer code of the device.  
 B – ID of the slot from the parameter table.  
 HL – Address of the parameter table.

Output: CY = 1 → there are no devices.  
 0 → there are devices.  
 B – ID of the slot from the parameter table.  
 HL – Starting address of the parameter table.

Note: Each device occupies 5 bytes in the table pointed to by HL, with the following structure:

- +00H – Reserved. Always 0.
- +01H – Manufacturer code.
- +02H – MSB address of the jump table.
- +03H – LSB address of the jump table.
- +04H – Device slot ID.

Manufacturers are as follows:

- 00 – ASCII
- 01 – Microsoft
- 02 – Canon
- 03 – Casio Computer
- 04 – Fujitsu
- 05 – General Fujitsu
- 06 – Hitachi, Ltd.
- 07 – Kyocera
- 08 – Matsushita (Panasonic)
- 09 – Mitsubishi Electric Corporation
- 10 – NEC
- 11 – Yamaha (Nippon Gakki)
- 12 – Japan Victor Company (JVC)
- 13 – Philips
- 14 – Pioneer
- 15 – Sanyo Electric
- 16 – Sharp Japan
- 17 – Sony
- 18 – Spectravideo
- 19 – Toshiba
- 20 – Mitsumi Electric
- 21 – Telematica
- 22 – Gradient Brazil
- 23 – Sharp do Brasil
- 24 – GoldStar (LG)
- 25 – Daewoo
- 26 – Samsung
- 128 – Image Scanner (Matsushita)
- 170 – Darky (SuperSoniqs)
- 171 – Darky (SuperSoniqs) second setting
- 212 – 1chipMSX / Zemmix Neo (KdL firmware)
- 254 – MPS2 (ASCII)

## **8.6 – DISC INTERFACE ROUTINES**

### **8.6.1 – Interface Initialization**

The routines below are executed only once during the system initialization at power up or after a reset, in the sequence INIHRD, DRIVES, INIENV. Their calling address is different for each interface.



**INIHRD** (???H/Disk interface).

Function: Initializes the hardware as soon as control is passed to the disk interface cartridge.

Input: None.

Output: None.

Registers: All.

**DRIVES** (???H/Disk Interface).

Function: Checks the physical drives connected to the system.

Input: Flag Z = 0 → Two logical units are assigned to one physical unit.

1 → Only one logical drive is assigned to a physical drive.

Output: L – Number of connected drives.

Registers: F, HL, IX, IY.

**INIENV** (???H/Disk Interface).

Function: Initializes the disk interface desktop.

Input: None.

Output: None.

Registers: All.

**8.6.2 – Standard interface routines****MALLOC** (01CBH/Disk Interface)

Function: Allocates a buffer for a segment for MSXDOS2.

Input: Number of bytes to reserve.

Output: A > 0 → Allocation error.

A = 0 → Allocation made.

HL – Buffer start address.

(HL-2, HL-1) – Buffer size + 2.

Registers: All.

**DEALLOC** (2D0FH/Disk Interface)

Function: Reallocates a buffer to a segment for MSXDOS2.

Input: HL – Initial buffer address.

(HL-2, HL-1) → Buffer size + 2.

Output: Unknown.

Registers: All.

**DSKIO** (4010H/Disk Interface)

Function: Direct reading/writing of sectors.

Input: HL – Pointer to TPA (clipboard).

DE – Number of the first sector to read  
or write.

B – Number of sectors to read or write.

A – Drive number (0=A:, 1=B:, 2=C:, etc).

C – Disk formatting ID:

F0H – 63 sectors per track (for HD's).

F8H – 80 tracks, 9 sectors per track, single face.

F9H – 80 tracks, 9 sectors per track, double sided.

FAH – 80 tracks, 8 sectors per track, single face.

FBH – 80 tracks, 8 sectors per track, double sided.

FCH – 40 tracks, 9 sectors per track, single face.

FDH – 40 tracks, 9 sectors per track, double sided.

CY = 0 → reading.

1 → writing

Output: B – Number of sectors effectively transferred.

CY = 1 → Transfer successfully executed.

0 → Transfer error. The error code is returned in  
register A.

A – Error code:

00 – Write protected.

02 – Not ready.

04 – CRC error (sector not accessible).

06 – Seek error.

08 – Record not found.

10 – Write fault.

12 – Other errors.

MSXDOS2 or higher only:

18 – Not a DOS disk.

20 – Incorrect MSXDOS version.

22 – Unformatted disk.

24 – Disk swapped.

Remaining: disk error.

Registers: All.

**DSKCHG** (4013H/Disk Interface)

Function: Check the swap status of the disk.

Input: A – Drive number (0=A:, 1=B:, 2=C:, etc).  
 B – Always 00H.  
 C – Disk formatting ID (same as DISKIO/4010H).  
 HL – Pointer for the respective DPB.

Output: CY = 0 → successfully verified.  
 CY = 1 → execution error.  
 A – Error code (same as DISKIO/4010H).  
 B – 00H → unknown state.  
 01H → disk not exchanged.  
 FFH → disk swapped.

Registers: All.

**GETDPB** (4016H/Disk Interface)

Function: Return the disk drive DPB.

Input: A – Drive number  
 B – First byte of FAT (disk ID).  
 C – Disk formatting ID (same as DISKIO/4010H).  
 HL – Pointer to the DPB to be filled (18 bytes).

Output: HL – DPB initial address filled in:

|      |         |                                  |
|------|---------|----------------------------------|
| +00H | DRIVE   | Drive number (0=A:, etc)         |
| +01H | MEDIA   | Media Type (F8H~FFH)             |
| +02H | SECSIZ  | Sector Size (must be $2^n$ )     |
| +04H | DIRMSK  | (SECSIZ/32) – 1                  |
| +05H | DIRSHFT | Number of bits 1 in DIRMSK       |
| +06H | CLUSMSK | (Sectors per cluster) – 1        |
| +07H | CLUSHFT | (Num of 1 bits in CLUSMSK) – 1   |
| +08H | FIRFAT  | Logical sector number of 1st FAT |
| +0AH | FATCNT  | Number of FATs                   |
| +0BH | MAXENT  | Number of root directory entries |
| +0CH | FIRREC  | First sector data area           |
| +0EH | MAXCLUS | (Total of clusters) + 1          |
| +10H | FATSIZ  | Number of sectors used           |
| +11H | FIRDIR  | First directory sector           |
| +13H | FATDIR  | FAT address in RAM               |

Registers: All.

**CHOICE** (4019H/Disk Interface)

Function: Returns the address of the disk format message.

Input: None.

Output: HL – Message address, which ends with a 00H byte. If there is no choice (only one formatting type is supported), HL returns 0000H.

Registers: All.

**DSKFMT** (401CH/Disk Interface)

Function: Format a disk.

Input: A – Choice of formatting by the user (CHOICE /4019H routine). It can range from 1 to 9.

D – Drive number (00H=A:, 01H=B:, etc).

HL – Starting address of the workspace used by the formatting routine.

BC – Size of the workspace used by the formatting routine.

Output: CY – 0 → Formatting completed successfully.  
1 → Error during formatting.

A – Error code:

00 – Write protected.

02 – Not ready.

04 – Data error (CRC).

06 – Seek error.

08 – Record not found.

10 – Write fault.

12 – Bad parameter.

14 – Insufficient memory.

16 – Other errors.

Registers: All.

**MTROFF** (401FH/Disk Interface)

Function: Stop the motor of the drives.

Input: None.

Output: None.

Registers: All.

Note: This function is implemented in only some interfaces. If the interface does not have this function implemented, the value of address 401FH will be 00H. Therefore, it is necessary to verify that the function exists by reading address 401FH before calling it.

**CALBAS** (4022H/Disk Interface)

Function: Call the BASIC interpreter.

Input: None.

Output: None.

Registers: All.

**FORMAT** (4025H/Disk Interface)

Function: Format a disk displaying message.

Input: None.

Output: None.

Registers: All.

**STPDRV** (4029H/Disk Interface)

Function: Stop the motor of the drives.

Input: None.

Output: None.

Registers: All.

**SLTDOS** (402DH/Disk Interface)

Function: Returns the DOS Kernel slot ID.

Input: None.

Output: A – Slot ID (same as RDSLT (000CH/Main)).

Registers: All.

**HIGMEM** (4030H/Disk Interface)

Function: Returns the highest address available in RAM.

Input: None.

Output: HL – Address.

Registers: All.

**BLKDOS** (402DH/Disk Interface)

Function: Returns the current MSXDOS2 block.

Input: None.

Output: A – Current block number (0 to 3).

Registers: All.

Note: The 64 Kbytes of MSXDOS2 Kernel ROM are split into 4 segments that can be active only on physical page 1. Therefore, they are constantly swapped during processing.

### 8.6.3 – Routines for accessing standard IDE Hard-Disks

#### IDBYT (7F80H/IDE Interface)

Function: Interface ID in 3 bytes. ("ID#" for IDE interfaces).

#### RDLBLK (7F89H/IDE Interface)

Function: Read logical sectors from disk or device.

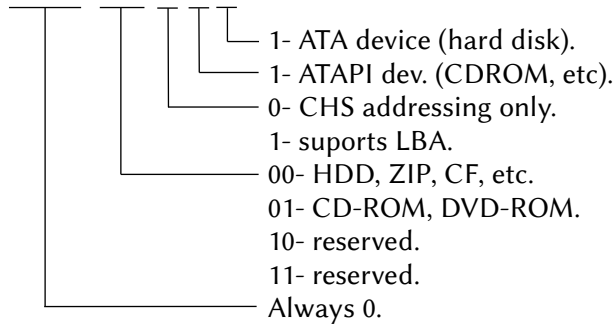
Input: CDE – Sector number.

HL – RAM address for read data.

B – Number of sectors to read.

A – Device ID:

| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
|----|----|----|----|----|----|----|----|
| 0  | 0  | 0  | T  | T  | E  | C  | H  |



Output: HL – Pointer to the read data.

CY = 1 → Read error.

A – Error code for IDE devices:

00 – Write protected.

02 – Not ready.

04 – CRC error (sector not accessible).

06 – Seek error.

08 – Record not found.

10 – Write fault.

12 – Other errors.

MSXDOS2 or higher only:

18 – Not a DOS disk.

20 – Incorrect MSXDOS version.

22 – Unformatted disk.

24 – Disk swapped.

Remaining: other errors.

Registers: All.

Note: This routine can also read sectors from the CD-ROM, which have 2048 bytes instead of 512 bytes of the HD's.

### **WRLBLK** (7F8CH/IDE Interface)

Function: Write logical sectors of the disk.

Input: CDE – Sector number.

HL – Starting address of the data to be written.

B – Number of sectors to write.

A – Device ID. Same as RDLBLK (7F89H).

Output: CY = 1 → Writing error.

A – Error code. Same as RDLBLK (7F89H).

Registers: All.

### **SELDEV** (7FB9H/IDE Interface)

Function: Select master/slave for ATAPI devices.

Input: A – bit0 = 0 → Master.

1 → Slave.

bit1~bit7: reserved. Always 0.

Output: CY = 1 → time-out error occurs.

Registers: A, BC, IX.

### **PACKET** (7FBCH/IDE Interface)

Function: Send a sequence of ATAPI commands to the selected device.

Input: HL – Pointer to 12-byte ATAPI command packet (cannot be on page 1 – 4000H~7FFFH).

DE – Address for data transfer (if any).

Output: CY = 1 → execution error.

Z = 1 → time-out error.

A = Error code. Same as RDLBLK (7F89H).

Registers: All.

Attention: This entry has different function on SCSI interfaces.

### **DRVADR** (7FBFH/IDE Interface)

Function: Returns the desktop address.

Input: A – Unit number (0 to 7).

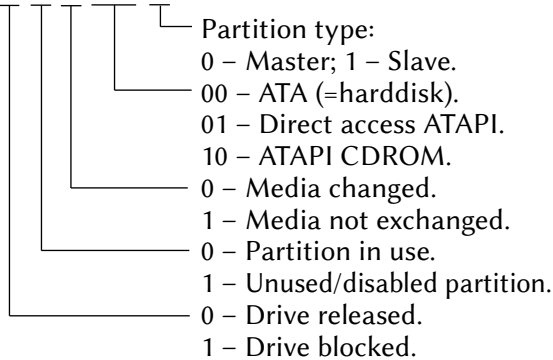
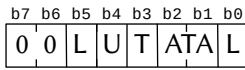
0~5 – Drive number (0=A:~5 = F:)

6 – Device Y Infobytes.

7 – 18 bytes of free space (used internally for sending ATAPI command strings).

Output: HL – Pointer to start of data:

+00H – Device Codebyte:



+01H~+03H – Partition start sector (bits 0~23).

+04H~+06H – (Size of partition in sectors) – 1 (bits 0~23).

+07H – Additional information about the partition.

For BIOS 3.0 or higher:

+08H – Partition start sector (bits 24~31).

+08H – (Size of partition in sectors) – 1 (bits 24~31).

Registers: AF, BC, DE, HL, IX.

## 8.6.4 – Routines for accessing standard SCSI Hard-Disks

**IDBYT** (7F80H/SCSI Interface)

Function: Interface ID in 3 bytes. (Ex.: “HD#”).

**INISYS** (7F83H/SCSI Interface)

Function: Starts SCSI interface.

Input: None.

Output: None.

Registers: All.

**TRMACT** (7F86H/SCSI Interface)

Function: Terminates HDD actions.

Input: None.



Output: A – SCSI interface status. Same as RDLBLK (7F89H).  
 D – Current Device Status. Same as RDLBLK (7F89H).  
 E – Messages. Same as RDLBLK (7F89H).

Registers: AF, DE.

### **RDLBLK** (7F89H/SCSI Interface)

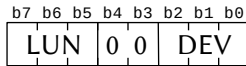
Function: Read logical sectors from disk or device.

Input: CDE – Sector number.

HL – RAM address for read data.

B – Number of sectors to read.

A – Device ID:



Número do dispositivo SCSI  
(0 a 7, ou 000 a 111).

LUN – Logical Unit Number.  
(Normalmente 0)

Output: HL – Pointer to the read data.

A – SCSI interface status.

00H – There was no error.

02H – Check condition.

04H – “MET” condition.

08H – Device busy.

0CH – Booking conflict.

10H – Intermediate condition.

14H – Intermediate condition “MET”.

18H – Reservation conflict.

22H – Command finished.

28H – Full queue.

30H – ACA active.

40H – Operation aborted.

D – Current device status.

00H – There was no error.

02H – Check condition.

04H – “MET” condition.

08H – Device busy.

0CH – Booking conflict.

10H – Intermediate condition.

14H – Intermediate condition “MET”.

18H – Reservation conflict.

22H – Command finished.

28H – Full queue.

30H – ACA active.

40H – Operation aborted.

E – Messages:

00H – Complete command.

01H, xx, 00H – Modify given pointers.

01H, xx, 01H – Request for transfer synchronous data.

01H, xx, 03H – Request for transfer total data.

02H – Save data pointers.

03H – Restore pointers.

04H – Disconnect.

05H – Initialization error.

06H – Abort.

07H – Message rejected.

08H – No operation.

09H – Message parity error.

0AH – Command attached complete.

0BH – Complete attached command (with flag).

0CH – Reset on device bus.

0DH – Abort TAG.

0EH – Clean/Empty Queue.

0FH – Start recovery.

10H – Release recovery.

11H – End I/O process.

20H – Single row tag.

21H – Queue header tag

22H – Ordered queue tag.

23H – Ignore waste.

80H ~ 0FFH – Identify.

Registers: All.

Note: This routine can also read sectors from the CD-ROM, which have 2048 bytes instead of 512 bytes from the HD's.

**WRLBLK** (7F8CH/SCSI Interface)

Function: Write logical sectors of the disk.

Input: CDE – Sector number.  
 HL – Starting address of the data to be written.  
 B – Number of sectors to write.  
 A – Device ID. Same as RDLBLK (7F89H).

Output: HL – Pointer to the read data.  
 A – SCSI Status. Same as RDLBLK (7F89H).  
 D – Device Status. Same as RDLBLK (7F89H).  
 E – Messages. Same as RDLBLK (7F89H).

Registers: All.

### RQSENS (7F8FH/SCSI Interface)

Function: Returns "sense" information about the SCSI device.

Input: A – Device ID. Same as RDLBLK (7F89H).

Output: A – DOS error code.

IX – Pointer to a buffer filled with "sense" data:

+00H – Error code:

70H – Fixed format, current "sense".

71H – Fixed format, "sense" previous.

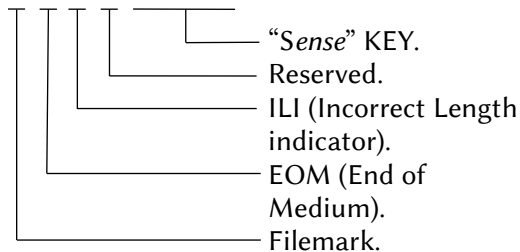
72H – Descriptor format, current "sense".

73H – Descriptor format, "sense" previous.

+01H – Segment number.

+02H –

|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
| F  | E  | I  | R  | C  | C  | C  | C  |



"Sense" key codes:

00H – No sense

01H – Error recovered

02H – Not ready

03H – Media error

04H – Hardware error

05H – Illegal request

- 06H – Unit requires attention
- 07H – Data protected
- 08H – Blank verification
- 09H – Manufacturer Specific
- 0AH – Copy aborted
- 0BH – Command aborted
- 0DH – Volume overflow
- 0EH – Agreement error
- 0FH – Completed
- +03H~+06H – Information
- +07H – Additional "sense" length (n-7)
- +08H~+11H – Command specific information
- +12H – Additional "sense" code
- +13H – Additional "sense" code qualifier
- +14H – Replaceable unit code
- +15H – Bit7 = 0 → There is no valid information  
1 → There is valid information
- +15H ( bit6~bit0)~+17H – Manufacturer specific information

Registers: AF, BC, DE.

### **INQUIRY** (7F92H/SCSI Interface)

Function: Returns SCSI device information.

Input: HL – Buffer address for read information.

A – Device ID.

Output: CY = 1 → reading error.

A – SCSI Status. Same as RDLBLK (7F89H).

D – Device Status. Same as RDLBLK (7F89H).

E – Messages. Same as RDLBLK (7F89H).

CY = 0 → HL – Points to the beginning of the buffer:

+00H – device code.

+01H – bit7 → RMB (removable media)  
bit6~bit0 → device type

+02H – Interface version:

00H – Not specified

01H – SCSI 1

02H – SCSI 2

+03H – bit7~bit4 → Reserved.

bit3~bit0 → Response data format.

- +04H – Additional length, contains how many subsequent bytes are valid.
- +05H~+07H – Reserved.
- +08H~+15H – Name (Ex. SEAGATE).
- +16H~+31H – Device ID (in ASCII).
- +32H – Hardware revision.
- +33H – Firmware revision.
- +34H – ROM revision.
- +35H – Reserved.

Registers: All.

### **RDSIZE** (7F95H/SCSI Interface)

Function: Returns the total space of the SCSI device.

Input: HL – Buffer address for read information.

A – Device ID. Same as RDLBLK (7F89H).

Output: CY = 1 → reading error.

A – SCSI Status. Same as RDLBLK (7F89H).

D – Device Status. Same as RDLBLK (7F89H).

E – Messages. Same as RDLBLK (7F89H).

CY = 0 → data read successfully.

(HL+0)~(HL+3) → total number of sectors (MSB/LSB).

(HL+4)~(HL+7) → sector size in bytes

(MSB/LSB). Typically 512 (00H-00H-02H-00H).

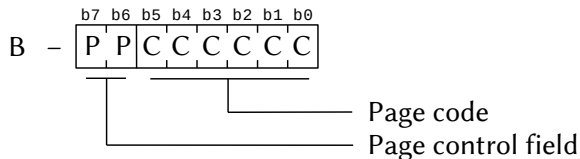
Registers: All.

### **MDSENS** (7F98H/SCSI Interface)

Function: Returns the “sense” parameters of the current mode.

Input: HL – Buffer address for read information.

A – Device ID. Same as RDLBLK (7F89H).



Output: CY = 1 → reading error.

A – SCSI Status. Same as RDLBLK (7F89H).

D – Device Status. Same as RDLBLK (7F89H).

E – Messages. Same as RDLBLK (7F89H).

CY = 0 → HL – Points to the beginning of the buffer:

- +00H – Operating parameters (SEAGATE).
- +01H – Error recovery parameters.
- +02H – Disconnected parameters.
- +03H – Format parameters.
- +04H – Geometry parameters.
- +05H~+1FH – Reserved.
- +20H – Drive serial number.
- +3FH – Returns all pages.

Registers: All.

### MDSEL (7F9BH/SCSI Interface)

Function: Mode selection. Used to boot HD.

Input: HL – Buffer address.

A – Device ID. Same as RDLBLK (7F89H).

B – Size of the parameter list.

Output: CY = 1 → reading error.

A – SCSI Status. Same as RDLBLK (7F89H).

D – Device Status. Same as RDLBLK (7F89H).

E – Messages. Same as RDLBLK (7F89H).

CY = 0 → HL points to the parameter list.

Registers: AF, BC, HL, IX.

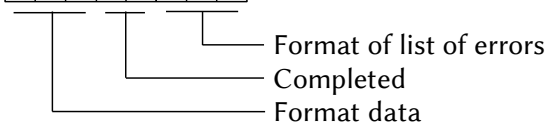
### HDFORM (7F9EH/SCSI Interface)

Function: Format the SCSI drive.

Input: A – Unit ID.

B – 

|    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|
| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
| 0  | 0  | 0  | F  | C  | D  | D  | D  |



DE – Interleave (MSB-LSB).

HL – Data address.

Output: CY = 1 → reading error.

A – SCSI Status. Same as RDLBLK (7F89H).

D – Device Status. Same as RDLBLK (7F89H).

E – Messages. Same as RDLBLK (7F89H).

CY = 0 → Formatted successfully.

Registers: AF, BC, DE, HL.

**TESTRD** (7FA1H/SCSI Interface)

Function: Tests whether the SCSI device is ready.

Input: A – Device ID. Same as RDLBLK (7F89H).

Output: A = 85H → device is ready.

A = 42H → the device is NOT ready.

Registers: All.

**SFBOOT** (7FA4H/SCSI Interface)

Function: Softboot the SCSI device.

Input: None.

Output: None.

Registers: All.

Note: This entry must not be used.

**INSWRK** (7FA7H/SCSI Interface)

Function: Mounts SCSI device table (installs desktop).

Input: None.

Output: None.

Registers: All.

Note: This input must not be used (internal routine).

**CLRLIN** (7FAAH/SCSI Interface)

Function: Cleans to end of line (prints ESC sequence).

Input: None.

Output: None.

Registers: All.

**VERIFY** (7FADH/SCSI Interface)

Function: Device verification.

Input: A – Device ID. Same as RDLBLK (7F89H).

B – Size to be checked (in blocks).

CDE – Logical block number.

HL – Address.

Output: A – SCSI Status. Same as RDLBLK (7F89H).

D – Device Status. Same as RDLBLK (7F89H).

E – Messages. Same as RDLBLK (7F89H).

Registers: AF, BC, HL, IX.

**STRSTP** (7FB0H/SCSI Interface)

Function: Starts or stops the drive.

Input: A – Device ID. Same as RDLBLK (7F89H).

B = 0 → Stops drive.

1 → Start the drive.

Output: A – SCSI Status. Same as RDLBLK (7F89H).

D – Device Status. Same as RDLBLK (7F89H).

E – Messages. Same as RDLBLK (7F89H).

Registers: All.

**SNDDGN** (7FB3H/SCSI Interface)

Function: Sends diagnostics.

Input: A – Device ID. Same as RDLBLK (7F89H).

Output: A – SCSI Status. Same as RDLBLK (7F89H).

D – Device Status. Same as RDLBLK (7F89H).

E – Messages. Same as RDLBLK (7F89H).

Registers: All.

**RESERV** (7FB6H/SCSI Interface)

Function: Reserved.

**RESER2** (7FB9H/SCSI Interface)

Function: Reserved.

**COPY** (7FBCH/SCSI Interface)

Function: Read “default” list.

Input: A – Device ID. Same as RDLBLK (7F89H).

DE – Length of the parameter list.

HL – Data address.

Output: A – SCSI Status. Same as RDLBLK (7F89H).

D – Device Status. Same as RDLBLK (7F89H).

E – Messages. Same as RDLBLK (7F89H).

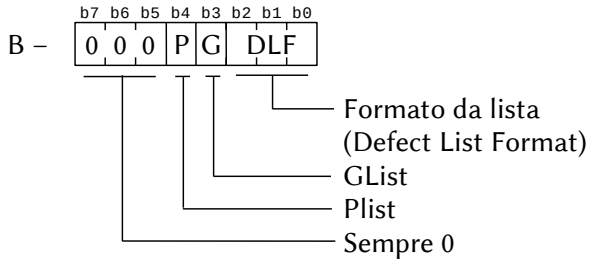
Attention: This input has different function on IDE interfaces. It is not advisable to use this call.

**RDEFCT** (7FBFH/SCSI Interface)

Function: Returns corrupted data.



Input: A – Device ID. Same as RDLBLK (7F89H).



DE – Size of allocated space.

HL – Data address.

Output: A – SCSI Status. Same as RDLBLK (7F89H).

D – Device Status. Same as RDLBLK (7F89H).

E – Messages. Same as RDLBLK (7F89H).

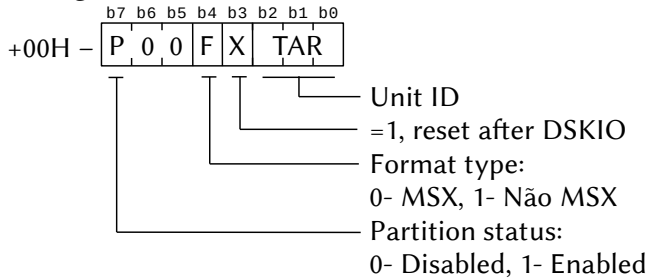
Registers: All.

### GETWRK (7FC2H/SCSI Interface)

Function: Returns the desktop address.

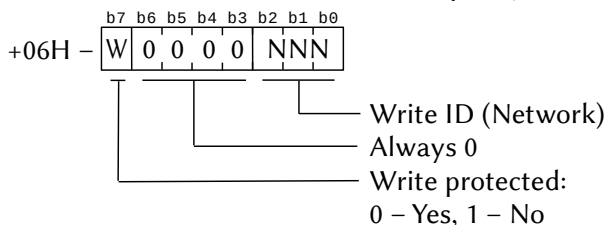
Input: None.

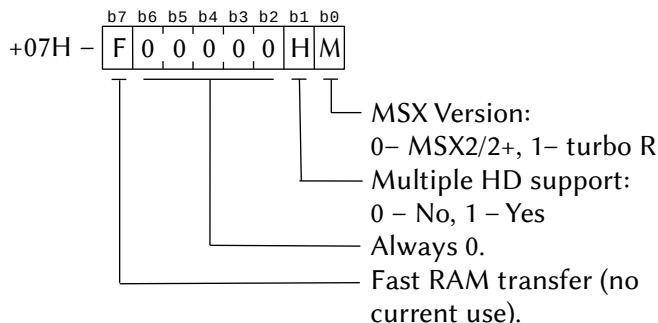
Output: HL = IX = Pointer to start of workspace. 8 bytes are reserved for each logical drive (there can be up to 6 logical drives, A: through F:). The structure for each unit is as follows:



+01H~+03H – Primeiro setor da partição.

+04H~+05H – Número de setores da partição.





Registers: AF, BC, HL, IX.

### **PRTINF** (7FC5H/SCSI Interface)

Function: Returns information about the partition.

Input: A - Drive number

Output: HL = IX = Pointer to the beginning of the workspace of the specified drive. There are 8 bytes with the same structure as GETWRK (7FC2H).

Registers: AF, BC, DE, HL, IX.

### **GTUNIT** (7FC8H/SCSI Interface)

Function: Returns the number of active units.

Input: None.

Output: A - Number of active units.

C - Vector ID.

D - Host ID.

Registers: AF, BC, DE.

### **HOSTID** (7FCBH/SCSI Interface)

Function: Select the Host ID.

Input: A - Host ID (4~7)

Output: CY = 1 → error.

Registers: AF, D.

### **TARGID** (7FCEH/SCSI Interface)

Function: Select the Target ID.

Input: A - Target ID (0~3)

Output: CY = 1 → error.

Registers: AF, D.

**GTTARG** (7FD1H/SCSI Interface)

Function: Returns the Target ID.

Input: None.

Output: A – Target ID.

Registers: AF.

**GTHOST** (7FD4H/SCSI Interface)

Function: Returns the Host ID.

Input: None.

Output: A – Host ID.

Registers: AF.

**GTSENS** (7FD7H/SCSI Interface)

Function: Returns “sense” data.

Input: A – Device ID. Same as RDLBLK (7F89H).

Output: A – Key “sense”.

C – Additional “sense” code.

D – Target Status

IX – Address data “sense”. Same as RQSENS (7F8FH).

Registers: AF, BC, DE.

**MEDREM** (7FDAH/SCSI Interface)

Function: Prevent media removal.

Input: A – Device ID. Same as RDLBLK (7F89H).

B = 0 → allows removal

1 → prevent removal

Output: A – SCSI Status. Same as RDLBLK (7F89H).

D – Device Status. Same as RDLBLK (7F89H).

E – Messages. Same as RDLBLK (7F89H).

Registers: All.

**8.7 – MSX-MUSIC ROUTINES (FM/OPLL)****WRTOPL** (4110H/FM-BIOS)

Function: Writes a byte of data into an OPLL register.

Input: A – OPLL Register

E – Data byte to be written

Output: None.

Registers: None.

**INIOPL** (4113H/FM-BIOS)

Function: Initializes the FM-BIOS/OPLL desktop.

Input: HL – Desktop start (must be even).

Output: None.

Registers: All.

**MSTART** (4116H/FM-BIOS)

Function: Starts playing music.

Input: HL – Music queue address.

A = 0 → Infinite loop.

1~254 → Number of repetitions.

255 → Reserved. Do not use.

The musical queue has the structure described below.

Header for 6 FM voices + 5 drum pieces:

```
+00~+01 0EH, 00H
+02~+03 Address for FM1CH
+04~+05 Address for FM2CH
+06~+07 Address for FM3CH
+08~+09 Address for FM4CH
+10~+11 Address for FM5CH
+12~+13 Address for FM6CH
+14 ... Data area
```

Header for 9 FM voices:

```
+00~+01 12H, 00H
+02~+03 Address for FM1CH
+04~+05 Address for FM2CH
+06~+07 Address for FM3CH
+08~+09 Address for FM4CH
+10~+11 Address for FM5CH
+12~+13 Address for FM6CH
+14~+15 Address for FM7CH
+16~+17 Address for FM8CH
+18~+19 Address for FM9CH
+20 ... Data area
```

Data area for melody:

```
+00H~+5FH Specifies the pitch. This number
    represents all musical scales, including
    the "pitch"
+60H~+6FH Volume
+70H~+7FH Instrument
+80H Release of "Sustain"
```

+81H Maintenance of "Sustain"  
 +82H Enable ROM instrument (0 to 63)  
 +83H Specify User Instrument  
 +84H Turn off legato  
 +85H Turn on legato  
 +86H Q designation (1 to 8). When legato is  
       on, the Q assignment is not performed.  
 +87H~+FEH Not used  
 +FFH End of data for each voice

#### Data area for rhythm:

| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
|----|----|----|----|----|----|----|----|
| V  | 0  | 1  | B  | S  | T  | C  | H  |

1 = enable the respective drum piece  
 B - Bass Drum  
 S - Snare Drum  
 T - Tom tom  
 C - Top Cymbal  
 H - Hi hat  
 0 = Select rhythm  
 1 = Select volume

FFH → end of rhythm data.

#### Instrument data storage format:

|    |               |     |             |     |              |          |  |
|----|---------------|-----|-------------|-----|--------------|----------|--|
| +0 | AM            | VIB | EG<br>TYP   | KSR | MULTIPLE     |          |  |
| +1 |               |     |             |     |              |          |  |
| +2 | KSL M         |     | TOTAL LEVEL |     |              |          |  |
| +3 | KSL C         |     | XX          | DC  | DM           | FEEDBACK |  |
| +4 | ATTACK RATE   |     |             |     | DECAY RATE   |          |  |
| +5 |               |     |             |     |              |          |  |
| +6 | SUSTAIN LEVEL |     |             |     | RELEASE RATE |          |  |
| +7 |               |     |             |     |              |          |  |

Output: None.

Registers: All.

#### **MSTOP** (4119H/FM-BIOS)

Function: Stop the music.

Input: None.

Output: None.

Registers: All.

**RDDATA** (411CH/FM-BIOS)

Function: Returns instrument data from ROM.

Input: HL – Buffer address for read data.  
A – Instrument number (0 to 63).

Output: None.

Registers: F.

**OPLDRV** (411FH/FM-BIOS)

Function: Input for OPLL driver. It is the routine that plays the music and must be called by the interrupt handler via the HTIMI hook.

Input: None.

Output: None.

Registers: None.

**TSTBGM** (4122H/FM-BIOS)

Function: Checks if there is still data in the music queue.

Input: None.

Output: A = 0 → no music being played  
A ≠ 0 → music is being played.

Registers: AF.

## 9 – MSX-HID (Human Interface Device)

Formula for the unique ID byte:

$HIDID = (byte1 \ll 4 | 0xF) \& (byte2 | 0xC0) \& (byte1 \ll 2 | 0x3F) \& 0xFF$

### 9.1 – FINGERPRINTS OF MSX DEVICES

|  |               |
|--|---------------|
| Unconnected, or MSX-joystick   | 3Fh, 3Fh, 3Fh |
| Mouse  | 30h, 30h, 30h |
| Trackball  | 38h, 38h, 38h |
| Touchpad(1)  | 39h, 3Dh, 39h |
| Touchpad(2)  | 3Dh, 3Dh, 3Dh |
| Lightgun   | 2Fh, 2Fh, 2Fh |
| Arkanoid Vaus Paddle   | 3Eh, 3Eh, 3Eh |
| Time encoded devices (each bit of “xx”<br>is zero for each analog channel present) | xxh, 3Fh, 3Fh |
| MSX-Paddle   | 3Eh, 3Fh, 3Fh |
| Yamaha MMP-01 music pad  | 3Ch, 3Fh, 3Fh |
| IBM-PC DA15 joystick adapter   | 3Ah, 3Fh, 3Fh |
| Atari dual-paddle adapter  | 36h, 3Fh, 3Fh |
| Dual-axis analog controller  | 30h, 3Fh, 3Fh |

### 9.2 – FINGERPRINTS OF SEGA COMPATIBLE DEVICES

|                             |                              |
|-----------------------------|------------------------------|
| Megadrive 3-button joystick | 3Fh, 33h, 3Fh                |
| Megadrive 6-button joystick | 3Fh, 33h, 3Fh, 33h, 3Fh, 30h |
| Megadrive Multi-Tap         | 33h, 3Fh, 33h                |
| Saturn digital joystick     | 3Ch, 3Fh, 3Ch                |
| Saturn Mouse                | 30h, 3Bh, 30h                |
| Sega 3line-handshake device | 31h, 31h, 31h                |

### 9.3 – FINGERPRINTS OF DEVICES THAT CONFLICT

The following devices can conflict with other MSX-HID devices. If necessary, both cases can be distinguished from the other device with one extra detection step.

|                              |               |
|------------------------------|---------------|
| Micomsoft XE1-AP analog mode | 2Fh, 2Fh, 2Fh |
| Sega-Mouse (Megadrive)       | 30h, 30h, 30h |

## 9.4 – HOMEBREW DEVICES

|                            |               |
|----------------------------|---------------|
| Ninja-tap                  | 3Fh, 1Fh, 3Fh |
| 3D glasses                 | 3Fh, 37h, 3Fh |
| 3D glasses + light gun     | 2Fh, 27h, 2Fh |
| Passive PS/2 mouse adapter | 3Fh, 3Eh, 3Fh |

## 9.5 – RESERVED FINGERPRINTS (DO NOT USE)

- Any fingerprints that can be produced by a standard MSX joystick.
- Any fingerprints that set both the pin-6 and pin-7 of the joystick port to 0 simultaneously on the two first bytes.



# 10 – Z80/R800 MNEMONICS

## 10.1 – 8-BIT LOAD GROUP

| Mnemonic                | Operation             | C Z P <sub>v</sub> S N H | Binary   | Hex                  | TZ | Z1 | TR | RW |
|-------------------------|-----------------------|--------------------------|--|----------------------|----|----|----|----|
| LD <i>r</i> , <i>r'</i> | $r \leftarrow r'$     | • • • • • •              | 01 <i>r</i> <i>r'</i>  | --                   | 04 | 05 | 01 | 01 |
| LD <i>r</i> , <i>n</i>  | $r \leftarrow n$      | • • • • • •              | 00 <i>r</i> 110<br>$\leftarrow n \rightarrow$  | --                   | 07 | 08 | 02 | 02 |
| LD <i>u</i> , <i>u'</i> | $u \leftarrow u'$     | • • • • • •              | 11 011 101<br>01 <i>u</i> <i>u'</i>  | DD                   | 08 | 10 | 02 | 02 |
| LD <i>v</i> , <i>v'</i> | $v \leftarrow v'$     | • • • • • •              | 11 111 101<br>01 <i>v</i> <i>v'</i>  | FD                   | 08 | 10 | 02 | 02 |
| LD <i>u</i> , <i>n</i>  | $u \leftarrow n$      | • • • • • •              | 11 011 101<br>00 <i>u</i> 110<br>$\leftarrow n \rightarrow$                          | DD                   | 11 | 13 | 03 | 03 |
| LD <i>v</i> , <i>n</i>  | $u \leftarrow n$      | • • • • • •              | 11 111 101<br>00 <i>v</i> 110<br>$\leftarrow n \rightarrow$                          | FD                   | 11 | 13 | 03 | 03 |
| LD <i>r</i> , (HL)      | $r \leftarrow (HL)$   | • • • • • •              | 01 <i>r</i> 110  | --                   | 07 | 08 | 02 | 04 |
| LD <i>r</i> , (IX+d)    | $r \leftarrow (IX+d)$ | • • • • • •              | 11 011 101<br>01 <i>r</i> 110<br>$\leftarrow d \rightarrow$                          | DD                   | 19 | 21 | 05 | 07 |
| LD <i>r</i> , (IY+d)    | $r \leftarrow (IY+d)$ | • • • • • •              | 11 111 101<br>01 <i>r</i> 110<br>$\leftarrow d \rightarrow$                          | FD                   | 19 | 21 | 05 | 07 |
| LD (HL), <i>r</i>       | $(HL) \leftarrow r$   | • • • • • •              | 01 110 <i>r</i>  | --                   | 07 | 08 | 02 | 04 |
| LD (IX+d), <i>r</i>     | $(IX+d) \leftarrow r$ | • • • • • •              | 11 011 101<br>01 110 <i>r</i><br>$\leftarrow d \rightarrow$                          | DD                   | 19 | 21 | 05 | 07 |
| LD (IY+d), <i>r</i>     | $(IY+d) \leftarrow r$ | • • • • • •              | 11 111 101<br>01 110 <i>r</i><br>$\leftarrow d \rightarrow$                          | FD                   | 19 | 21 | 05 | 07 |
| LD (HL), <i>n</i>       | $(HL) \leftarrow n$   | • • • • • •              | 00 110 110<br>$\leftarrow n \rightarrow$   | 36                   | 10 | 11 | 03 | 05 |
| LD (IX+d), <i>n</i>     | $(IX+d) \leftarrow n$ | • • • • • •              | 11 011 101<br>01 110 110<br>$\leftarrow d \rightarrow$<br>$\leftarrow n \rightarrow$ | DD<br>36<br>--<br>-- | 19 | 21 | 05 | 07 |

| Mnemonic       | Operation  | C Z $\frac{P}{V}$ S N H | Binary                                     | Hex                  | TZ | Z1 | TR | RW |
|----------------|------------|-------------------------|--|----------------------|----|----|----|----|
| LD<br>(IY+d),n | (IY+d) ← n | • • • • • •             | 11 111 101<br>01 110 110<br>← d →<br>← n → | FD<br>36<br>--<br>-- | 19 | 21 | 05 | 07 |
| LD A, (BC)     | A ← (BC)   | • • • • • •             | 00 001 010                                 | 0A                   | 07 | 08 | 02 | 04 |
| LD A, (DE)     | A ← (DE)   | • • • • • •             | 00 011 010                                 | 1A                   | 07 | 08 | 02 | 04 |
| LD A, (nn)     | A ← (nn)   | • • • • • •             | 00 111 010<br>← n →<br>← n →               | 1A<br>--<br>--       | 13 | 14 | 04 | 06 |
| LD (BC), A     | (BC) ← A   | • • • • • •             | 00 000 010                                 | 02                   | 07 | 08 | 02 | 04 |
| LD (DE), A     | (DE) ← A   | • • • • • •             | 00 010 010                                 | 22                   | 07 | 08 | 02 | 04 |
| LD (nn), A     | (nn) ← A   | • • • • • •             | 00 110 010<br>← n →<br>← n →               | 32<br>--<br>--       | 13 | 14 | 04 | 06 |
| LD A, I        | A ← I      | • ↑ I ↓ • •             | 11 101 101<br>01 010 111                   | ED<br>57             | 09 | 11 | 02 | 02 |
| LD A, R        | A ← R      | • ↑ I ↓ • •             | 11 101 101<br>01 011 111                   | ED<br>5F             | 09 | 11 | 02 | 02 |
| LD I, A        | I ← A      | • • • • • •             | 11 101 101<br>01 000 111                   | ED<br>47             | 09 | 11 | 02 | 02 |
| LD R, A        | R ← A      | • • • • • •             | 11 101 101<br>01 001 111                   | ED<br>4F             | 09 | 11 | 02 | 02 |

|       | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|
| r, r' | B   | C   | D   | E   | H   | L   | •   | A   |
| u, u' | B   | C   | D   | E   | IXH | IXL | •   | A   |
| v, v' | B   | C   | D   | E   | IYH | IYL | •   | A   |

TZ - Z80 T Cycles  
 Z1 - Z80 + M1  
 TR - R800 T Cycles  
 RW - R800 + Wait

Flags notation:

• = Flag not affected

↑ = Flag affected according operation results

I = The IFF content is copied to the P/V flag.

## 10.2 – 16-BIT LOAD GROUP

| Mnemonic    | Operation  | C Z $\frac{P}{V}$ S N H | Binary   | Hex                  | TZ | Z1 | TR | RW |
|-------------|--|-------------------------|--|----------------------|----|----|----|----|
| LD dd,nn    | dd $\leftarrow$ nn   | • • • • • •             | 00 dd0 001<br>$\leftarrow$ n $\rightarrow$<br>$\leftarrow$ n $\rightarrow$               | --<br>--<br>--       | 10 | 11 | 03 | 03 |
| LD IX,nn    | IX $\leftarrow$ nn   | • • • • • •             | 11 011 101<br>00 100 001<br>$\leftarrow$ n $\rightarrow$<br>$\leftarrow$ n $\rightarrow$ | DD<br>21<br>--<br>-- | 14 | 16 | 04 | 04 |
| LD IY,nn    | IY $\leftarrow$ nn   | • • • • • •             | 11 111 101<br>00 100 001<br>$\leftarrow$ n $\rightarrow$<br>$\leftarrow$ n $\rightarrow$ | FD<br>21<br>--<br>-- | 14 | 16 | 04 | 04 |
| LD HL, (nn) | H $\leftarrow$ (nn+1)<br>L $\leftarrow$ (nn)                             | • • • • • •             | 00 101 010<br>$\leftarrow$ n $\rightarrow$<br>$\leftarrow$ n $\rightarrow$               | 2A<br>--<br>--       | 16 | 17 | 05 | 07 |
| LD dd, (nn) | dd <sub>H</sub> $\leftarrow$ (nn+1)<br>dd <sub>L</sub> $\leftarrow$ (nn) | • • • • • •             | 11 101 101<br>01 dd1 011<br>$\leftarrow$ n $\rightarrow$<br>$\leftarrow$ n $\rightarrow$ | ED<br>--<br>--<br>-- | 20 | 22 | 06 | 08 |
| LD IX, (nn) | IX <sub>H</sub> $\leftarrow$ (nn+1)<br>IX <sub>L</sub> $\leftarrow$ (nn) | • • • • • •             | 11 011 101<br>00 101 010<br>$\leftarrow$ n $\rightarrow$<br>$\leftarrow$ n $\rightarrow$ | DD<br>2A<br>--<br>-- | 20 | 22 | 06 | 08 |
| LD IY, (nn) | IY <sub>H</sub> $\leftarrow$ (nn+1)<br>IY <sub>L</sub> $\leftarrow$ (nn) | • • • • • •             | 11 111 101<br>00 101 010<br>$\leftarrow$ n $\rightarrow$<br>$\leftarrow$ n $\rightarrow$ | FD<br>2A<br>--<br>-- | 20 | 22 | 06 | 08 |
| LD (nn),HL  | (nn+1) $\leftarrow$ H<br>(nn) $\leftarrow$ L                             | • • • • • •             | 00 100 010<br>$\leftarrow$ n $\rightarrow$<br>$\leftarrow$ n $\rightarrow$               | 22<br>--<br>--       | 16 | 17 | 05 | 07 |
| LD (nn),dd  | (nn+1) $\leftarrow$ dd <sub>H</sub><br>(nn) $\leftarrow$ dd <sub>L</sub> | • • • • • •             | 11 101 101<br>01 dd0 011<br>$\leftarrow$ n $\rightarrow$<br>$\leftarrow$ n $\rightarrow$ | ED<br>--<br>--<br>-- | 20 | 22 | 06 | 08 |
| LD (nn),IX  | (nn+1) $\leftarrow$ IX <sub>H</sub><br>(nn) $\leftarrow$ IX <sub>L</sub> | • • • • • •             | 11 011 101<br>01 100 010<br>$\leftarrow$ n $\rightarrow$<br>$\leftarrow$ n $\rightarrow$ | DD<br>22<br>--<br>-- | 20 | 22 | 06 | 08 |

| Mnemonic    | Operation  | C Z $\frac{P}{V}$ S N H | Binary   | Hex                  | TZ | Z1 | TR | RW |
|-------------|--|-------------------------|--|----------------------|----|----|----|----|
| LD (nn), IY | (nn+1) $\leftarrow$ IY <sub>H</sub><br>(nn) $\leftarrow$ IY <sub>L</sub>                             | • • • • •               | 11 111 101<br>01 100 010<br>$\leftarrow$ n $\rightarrow$<br>$\leftarrow$ n $\rightarrow$ | FD<br>22<br>--<br>-- | 20 | 22 | 06 | 08 |
| LD SP, HL   | SP $\leftarrow$ HL   | • • • • •               | 11 111 001   | F9                   | 06 | 07 | 01 | 01 |
| LD SP, IX   | SP $\leftarrow$ IX   | • • • • •               | 11 011 101<br>11 111 001   | DD<br>F9             | 10 | 12 | 02 | 02 |
| LD SP, IY   | SP $\leftarrow$ IY   | • • • • •               | 11 111 101<br>11 111 001   | FD<br>F9             | 10 | 12 | 02 | 02 |
| PUSH qq     | (SP-2) $\leftarrow$ qq <sub>L</sub><br>(SP-1) $\leftarrow$ qq <sub>H</sub><br>SP $\leftarrow$ SP - 2 | • • • • •               | 11 qq0 101   | --                   | 11 | 12 | 04 | 06 |
| PUSH IX     | (SP-2) $\leftarrow$ IX <sub>L</sub><br>(SP-1) $\leftarrow$ IX <sub>H</sub><br>SP $\leftarrow$ SP - 2 | • • • • •               | 11 011 101<br>11 100 101   | DD<br>E5             | 15 | 17 | 05 | 07 |
| PUSH IY     | (SP-2) $\leftarrow$ IY <sub>L</sub><br>(SP-1) $\leftarrow$ IY <sub>H</sub><br>SP $\leftarrow$ SP - 2 | • • • • •               | 11 111 101<br>11 100 101   | FD<br>E5             | 15 | 17 | 05 | 07 |
| POP qq      | qq <sub>H</sub> $\leftarrow$ (SP+1)<br>qq <sub>L</sub> $\leftarrow$ (SP)<br>SP $\leftarrow$ SP + 2   | • • • • •               | 11 qq0 001   | --                   | 10 | 11 | 03 | 05 |
| POP IX      | IX <sub>H</sub> $\leftarrow$ (SP+1)<br>IX <sub>L</sub> $\leftarrow$ (SP)<br>SP $\leftarrow$ SP + 2   | • • • • •               | 11 011 101<br>11 100 001   | DD<br>E1             | 14 | 16 | 04 | 06 |
| POP IY      | IY <sub>H</sub> $\leftarrow$ (SP+1)<br>IY <sub>L</sub> $\leftarrow$ (SP)<br>SP $\leftarrow$ SP + 2   | • • • • •               | 11 111 101<br>11 100 001   | FD<br>E1             | 14 | 16 | 04 | 06 |

|    |    |    |    |    |
|----|----|----|----|----|
|    | 00 | 01 | 10 | 11 |
| dd | BC | DE | HL | SP |
| qq | BC | DE | HL | AF |
|    |    |    |    |    |

TZ - Z80 T Cycles

Z1 - Z80 + M1

TR - R800 T Cycles

RW - R800 + Wait

Flags notation:

• = Flag not affected

## 10.3 – 8-BIT ARITHMETIC GROUP

| Mnemonic   | Operation                      | C Z P <sub>v</sub> S N H                    | Binary   | Hex            | TZ | Z1 | TR | RW |
|------------|--------------------------------|---|--|----------------|----|----|----|----|
| ADD r      | $A \leftarrow A + r$           | $\uparrow \downarrow v \uparrow 0 \uparrow$ | 10 000 r   | --             | 04 | 05 | 01 | 01 |
| ADD p      | $A \leftarrow A + p$           | $\uparrow \downarrow v \uparrow 0 \uparrow$ | 11 011 101<br>10 000 r                                 | DD<br>--       | 08 | 10 | 02 | 02 |
| ADD q      | $A \leftarrow A + q$           | $\uparrow \downarrow v \uparrow 0 \uparrow$ | 11 111 101<br>10 000 r                                 | FD<br>--       | 08 | 10 | 02 | 02 |
| ADD (HL)   | $A \leftarrow A + (HL)$        | $\uparrow \downarrow v \uparrow 0 \uparrow$ | 10 000 110   | 86             | 07 | 08 | 02 | 04 |
| ADD (IX+d) | $A \leftarrow A + (IX+d)$      | $\uparrow \downarrow v \uparrow 0 \uparrow$ | 11 011 101<br>10 000 110<br>$\leftarrow d \rightarrow$ | DD<br>86<br>-- | 19 | 21 | 05 | 07 |
| ADD (IY+d) | $A \leftarrow A + (IY+d)$      | $\uparrow \downarrow v \uparrow 0 \uparrow$ | 11 111 101<br>10 000 110<br>$\leftarrow d \rightarrow$ | FD<br>86<br>-- | 19 | 21 | 05 | 07 |
| ADD n      | $A \leftarrow A + n$           | $\uparrow \downarrow v \uparrow 0 \uparrow$ | 11 000 110<br>$\leftarrow n \rightarrow$               | C6             | 07 | 08 | 02 | 02 |
| ADC r      | $A \leftarrow A + r + CY$      | $\uparrow \downarrow v \uparrow 0 \uparrow$ | 10 001 r   | --             | 04 | 05 | 01 | 01 |
| ADC p      | $A \leftarrow A + p + CY$      | $\uparrow \downarrow v \uparrow 0 \uparrow$ | 11 011 101<br>10 001 r                                 | DD<br>--       | 08 | 10 | 02 | 02 |
| ADC p      | $A \leftarrow A + q + CY$      | $\uparrow \downarrow v \uparrow 0 \uparrow$ | 11 111 101<br>10 001 r                                 | FD<br>--       | 08 | 10 | 02 | 02 |
| ADC (HL)   | $A \leftarrow A + (HL) + CY$   | $\uparrow \downarrow v \uparrow 0 \uparrow$ | 10 001 110   | 8E             | 07 | 08 | 02 | 04 |
| ADC (IX+d) | $A \leftarrow A + (IX+d) + CY$ | $\uparrow \downarrow v \uparrow 0 \uparrow$ | 11 011 101<br>10 001 110<br>$\leftarrow d \rightarrow$ | DD<br>8E<br>-- | 19 | 21 | 05 | 07 |
| ADC (IY+d) | $A \leftarrow A + (IY+d) + CY$ | $\uparrow \downarrow v \uparrow 0 \uparrow$ | 11 111 101<br>10 001 110<br>$\leftarrow d \rightarrow$ | FD<br>8E<br>-- | 19 | 21 | 05 | 07 |
| ADC n      | $A \leftarrow A + n + CY$      | $\uparrow \downarrow v \uparrow 0 \uparrow$ | 11 001 110<br>$\leftarrow n \rightarrow$               | CE             | 07 | 08 | 02 | 02 |
| SUB r      | $A \leftarrow A - r$           | $\uparrow \downarrow v \uparrow 1 \uparrow$ | 10 010 r   | --             | 04 | 05 | 01 | 01 |
| SUB p      | $A \leftarrow A - p$           | $\uparrow \downarrow v \uparrow 1 \uparrow$ | 11 011 101<br>10 010 r                                 | DD<br>--       | 08 | 10 | 02 | 02 |
| SUB p      | $A \leftarrow A - q$           | $\uparrow \downarrow v \uparrow 1 \uparrow$ | 11 111 101<br>10 010 r                                 | FD<br>--       | 08 | 10 | 02 | 02 |

| Mnemonic   | Operation                      | C Z $\overline{P}_V$ S N H                | Binary   | Hex            | TZ | Z1 | TR | RW |
|------------|--------------------------------|---|--|----------------|----|----|----|----|
| SUB (HL)   | $A \leftarrow A - (HL)$        | $\uparrow \uparrow v \uparrow 1 \uparrow$ | 10 010 110   | 96             | 07 | 08 | 02 | 04 |
| SUB (IX+d) | $A \leftarrow A - (IX+d)$      | $\uparrow \uparrow v \uparrow 1 \uparrow$ | 11 011 101<br>10 010 110<br>$\leftarrow d \rightarrow$ | DD<br>96<br>-- | 19 | 21 | 05 | 07 |
| SUB (IY+d) | $A \leftarrow A - (IY+d)$      | $\uparrow \uparrow v \uparrow 1 \uparrow$ | 11 111 101<br>10 010 110<br>$\leftarrow d \rightarrow$ | FD<br>96<br>-- | 19 | 21 | 05 | 07 |
| SUB n      | $A \leftarrow A - n$           | $\uparrow \uparrow v \uparrow 1 \uparrow$ | 11 010 110<br>$\leftarrow n \rightarrow$               | D6             | 07 | 08 | 02 | 02 |
| SBC r      | $A \leftarrow A - r - CY$      | $\uparrow \uparrow v \uparrow 1 \uparrow$ | 10 011 r   | --             | 04 | 05 | 01 | 01 |
| SBC p      | $A \leftarrow A - p - CY$      | $\uparrow \uparrow v \uparrow 1 \uparrow$ | 11 011 101<br>10 011 r                                 | DD<br>--       | 08 | 10 | 02 | 02 |
| SBC p      | $A \leftarrow A - q - CY$      | $\uparrow \uparrow v \uparrow 1 \uparrow$ | 11 111 101<br>10 011 r                                 | FD<br>--       | 08 | 10 | 02 | 02 |
| SBC (HL)   | $A \leftarrow A - (HL) - CY$   | $\uparrow \uparrow v \uparrow 1 \uparrow$ | 10 011 110   | 8E             | 07 | 08 | 02 | 04 |
| SBC (IX+d) | $A \leftarrow A - (IX+d) - CY$ | $\uparrow \uparrow v \uparrow 1 \uparrow$ | 11 011 101<br>10 011 110<br>$\leftarrow d \rightarrow$ | DD<br>8E<br>-- | 19 | 21 | 05 | 07 |
| SBC (IY+d) | $A \leftarrow A - (IY+d) - CY$ | $\uparrow \uparrow v \uparrow 1 \uparrow$ | 11 111 101<br>10 011 110<br>$\leftarrow d \rightarrow$ | FD<br>8E<br>-- | 19 | 21 | 05 | 07 |
| SBC n      | $A \leftarrow A - n - CY$      | $\uparrow \uparrow v \uparrow 1 \uparrow$ | 11 011 110<br>$\leftarrow n \rightarrow$               | CE             | 07 | 08 | 02 | 02 |
| INC r      | $r \leftarrow r + 1$           | $\bullet \uparrow v \uparrow 0 \uparrow$  | 00 r 100   | --             | 04 | 05 | 01 | 01 |
| INC (HL)   | $(HL) \leftarrow (HL) + 1$     | $\bullet \uparrow v \uparrow 0 \uparrow$  | 00 110 100   | --             | 11 | 12 | 04 | 07 |
| INC (IX+d) | $(IX+d) \leftarrow (IX+d) + 1$ | $\bullet \uparrow v \uparrow 0 \uparrow$  | 11 011 101<br>00 110 100<br>$\leftarrow d \rightarrow$ | DD<br>34<br>-- | 23 | 25 | 07 | 10 |
| INC (IY+d) | $(IY+d) \leftarrow (IY+d) + 1$ | $\bullet \uparrow v \uparrow 0 \uparrow$  | 11 111 101<br>00 110 100<br>$\leftarrow d \rightarrow$ | FD<br>34<br>-- | 23 | 25 | 07 | 10 |
| DEC r      | $r \leftarrow r - 1$           | $\bullet \uparrow v \uparrow 1 \uparrow$  | 00 r 101   | --             | 04 | 05 | 01 | 01 |
| DEC (HL)   | $(HL) \leftarrow (HL) - 1$     | $\bullet \uparrow v \uparrow 1 \uparrow$  | 00 110 101   | --             | 11 | 03 | 04 | 07 |

| Mnemonic   | Operation           | C Z $\frac{P}{V}$ S N H | Binary                            | Hex            | TZ | Z1 | TR | RW |
|------------|---------------------|-------------------------|-----------------------------------|----------------|----|----|----|----|
| DEC (IX+d) | (IX+d) ← (IX+d) - 1 | • ↓ V ↓ 1 ↓             | 11 011 101<br>00 110 101<br>← d → | DD<br>34<br>-- | 23 | 25 | 07 | 10 |
| DEC (IY+d) | (IY+d) ← (IY+d) - 1 | • ↓ V ↓ 1 ↓             | 11 111 101<br>00 110 101<br>← d → | FD<br>34<br>-- | 23 | 25 | 07 | 10 |
| MULB r     | HL ← A * r          | ↓ ↓ 0 0 • •             | 11 101 101<br>11 r 001            | ED<br>--       | -- | -- | 14 | 14 |

|   | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
|---|-----|-----|-----|-----|-----|-----|-----|-----|
| r | B   | C   | D   | E   | H   | L   | •   | A   |
| P | •   | •   | •   | •   | IXH | IXL | •   | •   |
| q | •   | •   | •   | •   | IYH | IYL | •   | •   |

TZ - Z80 T Cycles

Z1 - Z80 + M1

TR - R800 T Cycles

RW - R800 + Wait

## Flags notation:

• = Flag not affected.

↓ = Flag affected according operation results.

0 = Flag off.

1 = Flag on.

V = the P/V flag contains the overflow status: V=1 -> overflow;  
V=0 -> there was no overflow.P = the P/V flag contains the parity status. P=1 means the  
parity of the result is even; P=0 means it is odd.

## 10.4 – 16-BIT ARITHMETIC GROUP

| Mnemonic   | Operation                     | C Z $\frac{P}{V}$ S N H                           | Binary                   | Hex      | TZ | Z1 | TR | RW |
|------------|-------------------------------|---|--------------------------|----------|----|----|----|----|
| ADD HL,ss  | HL $\leftarrow$ HL + ss       | $\uparrow \bullet \bullet \bullet 0 ?$            | 00 ss1 001               | --       | 11 | 12 | 01 | 01 |
| ADD IX,pp  | IX $\leftarrow$ IX + ss       | $\uparrow \bullet \bullet \bullet 0 ?$            | 11 011 101<br>00 pp1 001 | DD<br>-- | 15 | 17 | 02 | 02 |
| ADD IY,rr  | IY $\leftarrow$ IY + ss       | $\uparrow \bullet \bullet \bullet 0 ?$            | 11 111 101<br>00 rr1 001 | FD<br>-- | 15 | 17 | 02 | 02 |
| ADC HL,SS  | HL $\leftarrow$ HL+ss+CY      | $\uparrow \uparrow V \uparrow 0 ?$                | 11 101 101<br>01 ss1 010 | ED<br>-- | 15 | 17 | 02 | 02 |
| SBC HL,SS  | HL $\leftarrow$ HL-ss-CY      | $\uparrow \uparrow V \uparrow 1 ?$                | 11 101 101<br>01 ss0 010 | ED<br>-- | 15 | 17 | 02 | 02 |
| INC ss     | ss $\leftarrow$ ss + 1        | $\bullet \bullet \bullet \bullet \bullet \bullet$ | 00 ss0 011               | --       | 06 | 07 | 01 | 01 |
| INC IX     | IX $\leftarrow$ IX + ss       | $\bullet \bullet \bullet \bullet \bullet \bullet$ | 11 011 101<br>00 100 011 | DD<br>23 | 10 | 12 | 02 | 02 |
| INC IY     | IY $\leftarrow$ IY + ss       | $\bullet \bullet \bullet \bullet \bullet \bullet$ | 11 111 101<br>00 100 011 | FD<br>23 | 10 | 12 | 02 | 02 |
| DEC ss     | ss $\leftarrow$ ss - 1        | $\bullet \bullet \bullet \bullet \bullet \bullet$ | 00 ss1 011               | --       | 06 | 07 | 01 | 01 |
| DEC IX     | IX $\leftarrow$ IX - ss       | $\bullet \bullet \bullet \bullet \bullet \bullet$ | 11 011 101<br>00 101 011 | DD<br>2B | 10 | 12 | 02 | 02 |
| DEC IY     | IY $\leftarrow$ IY - ss       | $\bullet \bullet \bullet \bullet \bullet \bullet$ | 11 111 101<br>00 101 011 | FD<br>2B | 10 | 12 | 02 | 02 |
| MULW HL,tt | DE:HL $\leftarrow$<br>HL * tt | $\uparrow \uparrow 0 0 \bullet \bullet$           | 11 101 101<br>11 tt0 011 | ED<br>-- | -- | -- | 36 | 36 |

|    |    |           |           |    |
|----|----|-----------|-----------|----|
|    | 00 | 01        | 10        | 11 |
| ss | BC | DE        | HL        | SP |
| pp | BC | DE        | IX        | SP |
| rr | BC | DE        | IY        | SP |
| tt | BC | $\bullet$ | $\bullet$ | SP |

TZ - Z80 T Cycles  
 Z1 - Z80 + M1  
 TR - R800 T Cycles  
 RW - R800 + Wait

## Flags notation:

$\bullet$  = Flag not affected.

$\uparrow$  = Flag affected according operation results.

0 = Flag off.

1 = Flag on.

? = Flag unknown.

V = the P/V flag contains the overflow status: V=1 -> overflow;  
V=0 -> there was no overflow.

P = the P/V flag contains the parity status. P=1 means the  
parity of the result is even; P=0 means it is odd.



## 10.5 – EXCHANGE GROUP

| Mnemonic   | Operation  | C Z $\frac{P}{V}$ S N H | Binary                   | Hex      | TZ | Z1 | TR | RW |
|------------|--|-------------------------|--------------------------|----------|----|----|----|----|
| EX DE,HL   | DE $\leftrightarrow$ HL  | • • • • •               | 11 101 011               | EB       | 04 | 05 | 01 | 01 |
| EX AF,AF'  | AF $\leftrightarrow$ AF'   | • • • • •               | 00 001 000               | 08       | 04 | 05 | 01 | 01 |
| EXX        | BC $\leftrightarrow$ BC'<br>DE $\leftrightarrow$ DE'<br>HL $\leftrightarrow$ HL'   | • • • • •               | 11 011 001               | D9       | 04 | 05 | 01 | 01 |
| EX (SP),HL | H $\leftrightarrow$ (SP+1)<br>L $\leftrightarrow$ (SP)                             | • • • • •               | 11 100 011               | E3       | 19 | 20 | 05 | 07 |
| EX (SP),IX | IX <sub>H</sub> $\leftrightarrow$ (SP+1)<br>IX <sub>L</sub> $\leftrightarrow$ (SP) | • • • • •               | 11 011 101<br>11 100 011 | DD<br>E3 | 23 | 25 | 06 | 08 |
| EX (SP),IY | IY <sub>H</sub> $\leftrightarrow$ (SP+1)<br>IY <sub>L</sub> $\leftrightarrow$ (SP) | • • • • •               | 11 111 101<br>11 100 011 | FD<br>E3 | 23 | 25 | 06 | 08 |

TZ – Z80 T Cycles

Z1 – Z80 + M1

TR – R800 T Cycles

RW – R800 + Wait

Flags notation:

• = Flag not affected

## 12.6 – BLOCK TRANSFER GROUP

| Mnemonic | Operation  | C Z $\frac{P}{V}$ S N H | Binary                   | Hex      | TZ | Z1 | TR | RW |
|----------|--|-------------------------|--------------------------|----------|----|----|----|----|
| LDI      | (DE) ← (HL)<br>DE ← DE+1<br>HL ← HL+1<br>BC ← BC-1               | • • ↑ • 0 0             | 11 101 101<br>10 100 000 | ED<br>A0 | 16 | 18 | 04 | 07 |
| LDIR     | (DE) ← (HL)<br>DE ← DE+1<br>HL ← HL+1<br>BC ← BC-1<br>{Até BC=0} | • • 0 • 0 0             | 11 101 101<br>10 110 000 | ED<br>B0 | 21 | 23 | 04 | ?  |
|          |  |                         |                          |          | 16 | 18 | 04 | 07 |
| LDD      | (DE) ← (HL)<br>DE ← DE-1<br>HL ← HL-1<br>BC ← BC-1               | • • ↑ • 0 0             | 11 101 101<br>10 101 000 | ED<br>A8 | 16 | 18 | 04 | 07 |
| LDDR     | (DE) ← (HL)<br>DE ← DE-1<br>HL ← HL-1<br>BC ← BC-1<br>{Até BC=0} | • • 0 • 0 0             | 11 101 101<br>10 111 000 | ED<br>B8 | 21 | 23 | 04 | ?  |
|          |  |                         |                          |          | 16 | 18 | 04 | 07 |

TZ - Z80 T Cycles

Z1 - Z80 + M1

TR - R800 T Cycles

RW - R800 + Wait

## Flags notation:

• = Flag not affected

↑ = Flag affected according operation results

0 = Flag off

NOTE: When there are two descriptions of cycles, they refer to the two conditions that the instruction can assume. Thus, for LDIR, the time in T cycles for the Z80 is 21; when BC reaches 0, 16 T cycles are spent.

## 10.7 – SEARCH GROUP

| Mnemonic | Operation  | C Z $\frac{P}{V}$ S N H | Binary                   | Hex      | TZ | Z1 | TR | RW |
|----------|--|-------------------------|--------------------------|----------|----|----|----|----|
| CPI      | A ← (HL)<br>HL ← HL+1<br>BC ← BC-1                             | • ↓ ↓ ↓ ↓ 1 ↓           | 11 101 101<br>10 100 001 | ED<br>A1 | 16 | 18 | 04 | 06 |
| CPIR     | A ← (HL)<br>HL ← HL+1<br>BC ← BC-1<br>{Até BC=0<br>ou A=(HL) } | • ↓ ↓ ↓ ↓ 1 ↓           | 11 101 101<br>10 110 001 | ED<br>B1 | 21 | 23 | 05 | ?  |
|          |  |                         |                          |          | 16 | 18 | 05 | 08 |
| CPD      | A ← (HL)<br>HL ← HL-1<br>BC ← BC-1                             | • ↓ ↓ ↓ ↓ 1 ↓           | 11 101 101<br>10 101 001 | ED<br>A9 | 16 | 18 | 04 | 06 |
| CPDR     | A ← (HL)<br>HL ← HL-1<br>BC ← BC-1<br>{Até BC=0<br>ou A=(HL) } | • ↓ ↓ ↓ ↓ 1 ↓           | 11 101 101<br>10 111 001 | ED<br>B9 | 21 | 23 | 05 | ?  |
|          |  |                         |                          |          | 16 | 18 | 05 | 08 |

TZ - Z80 T Cycles

Z1 - Z80 + M1

TR - R800 T Cycles

RW - R800 + Wait

Flags notation:

• = Flag not affected

↓ = Flag affected according operation results

1 = Flag on

NOTE: When there are two descriptions of cycles, they refer to the two conditions that the instruction can assume. Thus, for LDIR, the time in T cycles for the Z80 is 21; when BC reaches 0, 16 T cycles are spent.

## 10.8 – COMPARISON GROUP

| Mnemonic     | Operation  | C Z $\frac{P}{V}$ S N H                     | Binary                            | Hex            | TZ | Z1 | TR | RW |
|--------------|------------|---|-----------------------------------|----------------|----|----|----|----|
| CP A,r       | A - R      | $\uparrow \downarrow v \uparrow 1 \uparrow$ | 10 111 r                          | --             | 04 | 05 | 01 | 01 |
| CP A,p       | A - p      | $\uparrow \downarrow v \uparrow 1 \uparrow$ | 11 011 101<br>10 111 p            | DD<br>--       | 08 | 10 | 02 | 02 |
| CP A,q       | A - q      | $\uparrow \downarrow v \uparrow 1 \uparrow$ | 11 111 101<br>10 111 p            | FD<br>--       | 08 | 10 | 02 | 02 |
| CP A, (HL)   | A - (HL)   | $\uparrow \downarrow v \uparrow 1 \uparrow$ | 10 111 110                        | BE             | 07 | 08 | 02 | 04 |
| CP A, (IX+d) | A - (IX+d) | $\uparrow \downarrow v \uparrow 1 \uparrow$ | 11 011 101<br>10 111 110<br>← d → | DD<br>BE<br>-- | 19 | 21 | 05 | 07 |
| CP A, (IY+d) | A - (IY+d) | $\uparrow \downarrow v \uparrow 1 \uparrow$ | 11 111 101<br>10 111 110<br>← d → | FD<br>BE<br>-- | 19 | 21 | 05 | 07 |
| CP A,n       | A - n      | $\uparrow \downarrow v \uparrow 1 \uparrow$ | 11 111 110<br>← n →               | FE<br>--       | 07 | 08 | 02 | 02 |

|   | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
|---|-----|-----|-----|-----|-----|-----|-----|-----|
| r | B   | C   | D   | E   | H   | L   | •   | A   |
| p | •   | •   | •   | •   | IXH | IXL | •   | •   |
| q | •   | •   | •   | •   | IYH | IYL | •   | •   |

TZ - Z80 T Cycles

Z1 - Z80 + M1

TR - R800 T Cycles

RW - R800 + Wait

## Flags notation:

 $\uparrow$  = Flag affected according operation results.

1 = Flag on.

V = the P/V flag contains the overflow status: V=1 -&gt; overflow;

V=0 -&gt; there was no overflow.

## 10.9 – LOGICAL GROUP

| Mnemonic   | Operation                      | C Z $\frac{P}{V}$ S N H       | Binary   | Hex            | TZ | Z1 | TR | RW |
|------------|--------------------------------|-------------------------------|--|----------------|----|----|----|----|
| AND r      | $A \leftarrow A \wedge r$      | 0 $\uparrow$ P $\uparrow$ 0 1 | 10 100 r   | --             | 04 | 05 | 01 | 01 |
| AND p      | $A \leftarrow A \wedge p$      | 0 $\uparrow$ P $\uparrow$ 0 1 | 11 011 101<br>10 100 p                                 | DD<br>--       | 08 | 10 | 02 | 02 |
| AND q      | $A \leftarrow A \wedge q$      | 0 $\uparrow$ P $\uparrow$ 0 1 | 11 111 101<br>10 100 p                                 | FD<br>--       | 08 | 10 | 02 | 02 |
| AND (HL)   | $A \leftarrow A \wedge (HL)$   | 0 $\uparrow$ P $\uparrow$ 0 1 | 10 100 110   | A6             | 07 | 08 | 02 | 04 |
| AND (IX+d) | $A \leftarrow A \wedge (IX+d)$ | 0 $\uparrow$ P $\uparrow$ 0 1 | 11 011 101<br>10 100 110<br>$\leftarrow d \rightarrow$ | DD<br>A6<br>-- | 19 | 21 | 05 | 07 |
| AND (IY+d) | $A \leftarrow A \wedge (IY+d)$ | 0 $\uparrow$ P $\uparrow$ 0 1 | 11 111 101<br>10 100 110<br>$\leftarrow d \rightarrow$ | FD<br>A6<br>-- | 19 | 21 | 05 | 07 |
| AND n      | $A \leftarrow A \wedge n$      | 0 $\uparrow$ P $\uparrow$ 0 1 | 11 100 110<br>$\leftarrow n \rightarrow$               | E6<br>--       | 07 | 08 | 02 | 02 |
| OR r       | $A \leftarrow A \vee r$        | 0 $\uparrow$ P $\uparrow$ 0 1 | 10 110 r   | --             | 04 | 05 | 01 | 01 |
| OR p       | $A \leftarrow A \vee p$        | 0 $\uparrow$ P $\uparrow$ 0 1 | 11 011 101<br>10 110 p                                 | DD<br>--       | 08 | 10 | 02 | 02 |
| OR q       | $A \leftarrow A \vee q$        | 0 $\uparrow$ P $\uparrow$ 0 1 | 11 111 101<br>10 110 p                                 | FD<br>--       | 08 | 10 | 02 | 02 |
| OR (HL)    | $A \leftarrow A \vee (HL)$     | 0 $\uparrow$ P $\uparrow$ 0 1 | 10 110 110   | B6             | 07 | 08 | 02 | 04 |
| OR (IX+d)  | $A \leftarrow A \vee (IX+d)$   | 0 $\uparrow$ P $\uparrow$ 0 1 | 11 011 101<br>10 110 110<br>$\leftarrow d \rightarrow$ | DD<br>B6<br>-- | 19 | 21 | 05 | 07 |
| OR (IY+d)  | $A \leftarrow A \vee (IY+d)$   | 0 $\uparrow$ P $\uparrow$ 0 1 | 11 111 101<br>10 110 110<br>$\leftarrow d \rightarrow$ | FD<br>B6<br>-- | 19 | 21 | 05 | 07 |
| OR n       | $A \leftarrow A \vee n$        | 0 $\uparrow$ P $\uparrow$ 0 1 | 11 110 110<br>$\leftarrow n \rightarrow$               | F6<br>--       | 07 | 08 | 02 | 02 |
| XOR r      | $A \leftarrow A \oplus r$      | 0 $\uparrow$ P $\uparrow$ 0 1 | 10 110 r   | --             | 04 | 05 | 01 | 01 |
| XOR p      | $A \leftarrow A \oplus p$      | 0 $\uparrow$ P $\uparrow$ 0 1 | 11 011 101<br>10 110 p                                 | DD<br>--       | 08 | 10 | 02 | 02 |
| XOR q      | $A \leftarrow A \oplus q$      | 0 $\uparrow$ P $\uparrow$ 0 1 | 11 111 101<br>10 110 p                                 | FD<br>--       | 08 | 10 | 02 | 02 |
| XOR (HL)   | $A \oplus (HL)$                | 0 $\uparrow$ P $\uparrow$ 0 1 | 10 110 110   | B6             | 07 | 08 | 02 | 04 |

| Mnemonic   | Operation                      | C Z P <sub>V</sub> S N H          | Binary   | Hex            | TZ | Z1 | TR | RW |
|------------|--------------------------------|-----------------------------------|--|----------------|----|----|----|----|
| XOR (IX+d) | $A \leftarrow A \oplus (IX+d)$ | 0 $\downarrow$ P $\downarrow$ 0 1 | 11 011 101<br>10 110 110<br>$\leftarrow d \rightarrow$ | DD<br>B6<br>-- | 19 | 21 | 05 | 07 |
| XOR (IY+d) | $A \leftarrow A \oplus (IY+d)$ | 0 $\downarrow$ P $\downarrow$ 0 1 | 11 111 101<br>10 110 110<br>$\leftarrow d \rightarrow$ | FD<br>B6<br>-- | 19 | 21 | 05 | 07 |
| XOR n      | $A \leftarrow A \oplus n$      | 0 $\downarrow$ P $\downarrow$ 0 1 | 11 110 110<br>$\leftarrow n \rightarrow$               | F6<br>--       | 07 | 08 | 02 | 02 |

|   | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
|---|-----|-----|-----|-----|-----|-----|-----|-----|
| r | B   | C   | D   | E   | H   | L   | •   | A   |
| P | •   | •   | •   | •   | IXH | IXL | •   | •   |
| q | •   | •   | •   | •   | IYH | IYL | •   | •   |

TZ - Z80 T Cycles

Z1 - Z80 + M1

TR - R800 T Cycles

RW - R800 + Wait

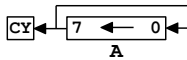
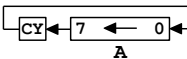
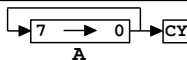
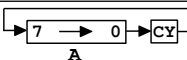
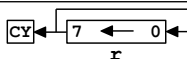
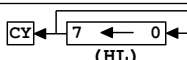
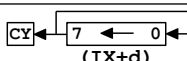
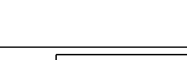
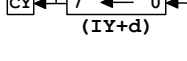
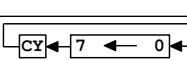
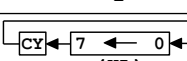
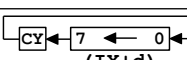

**Flags notation:** $\downarrow$  = Flag affected according operation results

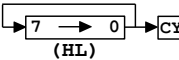
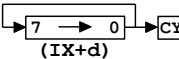
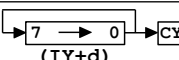
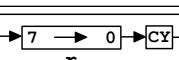
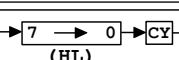
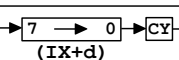
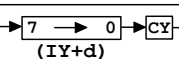
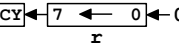
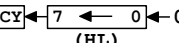
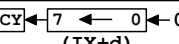
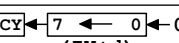
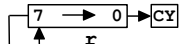
0 = Flag off

1 = Flag on

P = the P/V flag contains the parity status. P=1 means the parity of the result is even; P=0 means it is odd.

## 10.10 – ROTATE AND SHIFT GROUP

| Mnemonic   | Operation   | C Z $\frac{P}{V}$ S N H                  | Binary   | Hex                  | TZ | Z1 | TR | RW |
|------------|---|--|--|----------------------|----|----|----|----|
| RLCA       |    | $\downarrow \bullet \bullet \bullet 0 0$ | 00 000 111   | 07                   | 04 | 05 | 01 | 01 |
| RLA        |    | $\downarrow \bullet \bullet \bullet 0 0$ | 00 010 111   | 0F                   | 04 | 05 | 01 | 01 |
| RRCA       |    | $\downarrow \bullet \bullet \bullet 0 0$ | 00 001 111   | 17                   | 04 | 05 | 01 | 01 |
| RRA        |    | $\downarrow \bullet \bullet \bullet 0 0$ | 00 011 111   | 1F                   | 04 | 05 | 01 | 01 |
| RLC r      |    | $\downarrow \downarrow P \downarrow 0 0$ | 11 001 011<br>00 000 r   | CB<br>--             | 08 | 10 | 02 | 02 |
| RLC (HL)   |    | $\downarrow \downarrow P \downarrow 0 0$ | 11 001 011<br>00 000 110   | CB<br>06             | 15 | 17 | 05 | 08 |
| RLC (IX+d) |    | $\downarrow \downarrow P \downarrow 0 0$ | 11 011 101<br>11 001 011<br>$\leftarrow d \rightarrow$<br>00 000 110 | DD<br>CB<br>--<br>06 | 23 | 25 | 07 | 10 |
| RLC (IY+d) |    | $\downarrow \downarrow P \downarrow 0 0$ | 11 111 101<br>11 001 011<br>$\leftarrow d \rightarrow$<br>00 000 110 | FD<br>CB<br>--<br>06 | 23 | 25 | 07 | 10 |
| RL r       |    | $\downarrow \downarrow P \downarrow 0 0$ | 11 001 011<br>00 010 r   | CB<br>--             | 08 | 10 | 02 | 02 |
| RL (HL)    |   | $\downarrow \downarrow P \downarrow 0 0$ | 11 001 011<br>00 010 110   | CB<br>16             | 15 | 17 | 05 | 08 |
| RL (IX+d)  |  | $\downarrow \downarrow P \downarrow 0 0$ | 11 011 101<br>11 001 011<br>$\leftarrow d \rightarrow$<br>00 010 110 | DD<br>CB<br>--<br>16 | 23 | 25 | 07 | 10 |
| RL (IY+d)  |  | $\downarrow \downarrow P \downarrow 0 0$ | 11 111 101<br>11 001 011<br>$\leftarrow d \rightarrow$<br>00 010 110 | FD<br>CB<br>--<br>16 | 23 | 25 | 07 | 10 |
| RRC r      |  | $\downarrow \downarrow P \downarrow 0 0$ | 11 001 011<br>00 001 r   | CB<br>--             | 08 | 10 | 02 | 02 |

| Mnemonic   | Operation   | C Z $\frac{P}{V}$ S N H                  | Binary   | Hex                  | TZ | Z1 | TR | RW |
|------------|---|--|--|----------------------|----|----|----|----|
| RRC (HL)   |    | $\downarrow \downarrow P \downarrow 0 0$ | 11 001 011<br>00 001 110   | CB<br>0E             | 15 | 17 | 05 | 08 |
| RRC (IX+d) |    | $\downarrow \downarrow P \downarrow 0 0$ | 11 011 101<br>11 001 011<br>$\leftarrow d \rightarrow$<br>00 001 110 | DD<br>CB<br>--<br>0E | 23 | 25 | 07 | 10 |
| RRC (IY+d) |    | $\downarrow \downarrow P \downarrow 0 0$ | 11 111 101<br>11 001 011<br>$\leftarrow d \rightarrow$<br>00 001 110 | FD<br>CB<br>--<br>0E | 23 | 25 | 07 | 10 |
| RR r       |    | $\downarrow \downarrow P \downarrow 0 0$ | 11 001 011<br>00 011 r   | CB<br>--             | 08 | 10 | 02 | 02 |
| RR (HL)    |    | $\downarrow \downarrow P \downarrow 0 0$ | 11 001 011<br>00 011 110   | CB<br>1E             | 15 | 17 | 05 | 08 |
| RR (IX+d)  |    | $\downarrow \downarrow P \downarrow 0 0$ | 11 011 101<br>11 001 011<br>$\leftarrow d \rightarrow$<br>00 011 110 | DD<br>CB<br>--<br>1E | 23 | 25 | 07 | 10 |
| RR (IY+d)  |    | $\downarrow \downarrow P \downarrow 0 0$ | 11 111 101<br>11 001 011<br>$\leftarrow d \rightarrow$<br>00 011 110 | FD<br>CB<br>--<br>1E | 23 | 25 | 07 | 10 |
| SLA r      |   | $\downarrow \downarrow P \downarrow 0 0$ | 11 001 011<br>00 100 r   | CB<br>--             | 08 | 10 | 02 | 02 |
| SLA (HL)   |  | $\downarrow \downarrow P \downarrow 0 0$ | 11 001 011<br>00 100 110   | CB<br>26             | 15 | 17 | 05 | 08 |
| SLA (IX+d) |  | $\downarrow \downarrow P \downarrow 0 0$ | 11 011 101<br>11 001 011<br>$\leftarrow d \rightarrow$<br>00 100 110 | DD<br>CB<br>--<br>26 | 23 | 25 | 07 | 10 |
| SLA (IY+d) |  | $\downarrow \downarrow P \downarrow 0 0$ | 11 111 101<br>11 001 011<br>$\leftarrow d \rightarrow$<br>00 100 110 | FD<br>CB<br>--<br>26 | 23 | 25 | 07 | 10 |
| SRA r      |  | $\downarrow \downarrow P \downarrow 0 0$ | 11 001 011<br>00 101 r   | CB<br>--             | 08 | 10 | 02 | 02 |



| Mnemonic   | Operation | C Z P <sub>V</sub> S N H | Binary  | Hex                  | TZ | Z1 | TR | RW |
|------------|-----------|--------------------------|---|----------------------|----|----|----|----|
| SRA (HL)   |           | ↓ ↓ P ↓ 0 0              | 11 001 011<br>00 101 110                        | CB<br>2E             | 15 | 17 | 05 | 08 |
| SRA (IX+d) |           | ↓ ↓ P ↓ 0 0              | 11 011 101<br>11 001 011<br>← d →<br>00 101 110 | DD<br>CB<br>--<br>2E | 23 | 25 | 07 | 10 |
| SRA (IY+d) |           | ↓ ↓ P ↓ 0 0              | 11 111 101<br>11 001 011<br>← d →<br>00 101 110 | FD<br>CB<br>--<br>2E | 23 | 25 | 07 | 10 |
| SRL r      |           | ↓ ↓ P ↓ 0 0              | 11 001 011<br>00 111 r                          | CB<br>--             | 08 | 10 | 02 | 02 |
| SRL (HL)   |           | ↓ ↓ P ↓ 0 0              | 11 001 011<br>00 111 110                        | CB<br>3E             | 15 | 17 | 05 | 08 |
| SRL (IX+d) |           | ↓ ↓ P ↓ 0 0              | 11 011 101<br>11 001 011<br>← d →<br>00 111 110 | DD<br>CB<br>--<br>3E | 23 | 25 | 07 | 10 |
| SRL (IY+d) |           | ↓ ↓ P ↓ 0 0              | 11 111 101<br>11 001 011<br>← d →<br>00 111 110 | FD<br>CB<br>--<br>3E | 23 | 25 | 07 | 10 |
| RLD        |           | • ↓ P ↓ 0 0              | 11 101 101<br>01 101 111                        | ED<br>6F             | 18 | 20 | 05 | 08 |
| RRD        |           | • ↓ P ↓ 0 0              | 11 101 101<br>01 100 111                        | ED<br>67             | 18 | 20 | 05 | 08 |

|   | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
|---|-----|-----|-----|-----|-----|-----|-----|-----|
| r | B   | C   | D   | E   | H   | L   | •   | A   |

TZ - Z80 T Cycles

Z1 - Z80 + M1

TR - R800 T Cycles

RW - R800 + Wait

## Flags notation:

• = Flag not affected

↓ = Flag affected according operation results

0 = Flag off

P = the P/V flag contains the parity status. P=1 means the parity of the result is even; P=0 means it is odd.

## 10.11 – BIT SET, RESET AND TEST GROUP

| Mnemonic      | Operation                          | C Z $\frac{P}{V}$ S N H | Binary   | Hex                  | TZ | Z1 | TR | RW |
|---------------|------------------------------------|-------------------------|--|----------------------|----|----|----|----|
| BIT b, r      | $Z \leftarrow \overline{r_b}$      | 0 $\downarrow$ ? ? 0 1  | 11 001 011<br>01 b r   | CB<br>--             | 08 | 10 | 02 | 02 |
| BIT b, (HL)   | $Z \leftarrow \overline{(HL)_b}$   | 0 $\downarrow$ ? ? 0 1  | 11 001 011<br>01 b 110   | CB<br>--             | 12 | 04 | 03 | 05 |
| BIT b, (IX+d) | $Z \leftarrow \overline{(IX+d)_b}$ | 0 $\downarrow$ ? ? 0 1  | 11 011 101<br>11 001 011<br>$\leftarrow d \rightarrow$<br>01 b 110 | DD<br>CB<br>--<br>-- | 20 | 22 | 05 | 07 |
| BIT b, (IY+d) | $Z \leftarrow \overline{(IY+d)_b}$ | 0 $\downarrow$ ? ? 0 1  | 11 111 101<br>11 001 011<br>$\leftarrow d \rightarrow$<br>01 b 110 | FD<br>CB<br>--<br>-- | 20 | 22 | 05 | 07 |
| SET b, r      | $\overline{r_b} \leftarrow 1$      | • • • • •               | 11 001 011<br>11 b r   | CB<br>--             | 08 | 10 | 02 | 02 |
| SET b, (HL)   | $\overline{(HL)_b} \leftarrow 1$   | • • • • •               | 11 001 011<br>11 b 110   | CB<br>--             | 15 | 17 | 05 | 08 |
| SET b, (IX+d) | $\overline{(IX+d)_b} \leftarrow 1$ | • • • • •               | 11 011 101<br>11 001 011<br>$\leftarrow d \rightarrow$<br>11 b 110 | DD<br>CB<br>--<br>-- | 23 | 25 | 07 | 10 |
| SET b, (IY+d) | $\overline{(IY+d)_b} \leftarrow 1$ | • • • • •               | 11 111 101<br>11 001 011<br>$\leftarrow d \rightarrow$<br>11 b 110 | FD<br>CB<br>--<br>-- | 23 | 25 | 07 | 10 |
| RES b, r      | $\overline{r_b} \leftarrow 0$      | • • • • •               | 11 001 011<br>10 b r   | CB<br>--             | 08 | 10 | 02 | 02 |
| RES b, (HL)   | $\overline{(HL)_b} \leftarrow 0$   | • • • • •               | 11 001 011<br>10 b 110   | CB<br>--             | 15 | 17 | 05 | 08 |
| RES b, (IX+d) | $\overline{(IX+d)_b} \leftarrow 0$ | • • • • •               | 11 011 101<br>11 001 011<br>$\leftarrow d \rightarrow$<br>10 b 110 | DD<br>CB<br>--<br>-- | 23 | 25 | 07 | 10 |

| Mnemonic      | Operation                          | C Z $\frac{P}{V}$ S N H | Binary   | Hex                  | TZ | Z1 | TR | RW |
|---------------|------------------------------------|-------------------------|--|----------------------|----|----|----|----|
| RES b, (IY+d) | $\overline{(IY+d)}_b \leftarrow 0$ | • • • • •               | 11 111 101<br>11 001 011<br>$\leftarrow d \rightarrow$<br>10 b 110 | FD<br>CB<br>--<br>-- | 23 | 25 | 07 | 10 |

|   |     |     |     |     |     |     |     |     |
|---|-----|-----|-----|-----|-----|-----|-----|-----|
|   | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| r | B   | C   | D   | E   | H   | L   | •   | A   |
| b | b0  | b1  | b2  | b3  | b4  | b5  | b6  | b7  |

TZ - Z80 T Cycles  
 Z1 - Z80 + M1  
 TR - R800 T Cycles  
 RW - R800 + Wait

**Flags notation:**

• = Flag not affected  
 ↓ = Flag affected according operation results  
 0 = Flag off  
 1 = Flag on  
 ? = Flag unknown

## 10.12 – JUMP GROUP

| Mnemonic | Operation                      | C Z $\frac{P}{V}$ S N H | Binary                       | Hex            | TZ       | Z1       | TR       | RW       |
|----------|--------------------------------|-------------------------|------------------------------|----------------|----------|----------|----------|----------|
| JP nn    | PC ← nn                        | • • • • • •             | 10 000 011<br>← n →<br>← n → | C3<br>--<br>-- | 10       | 11       | 03       | 05       |
| JP cc,nn | If cc=true,<br>PC ← nn         | • • • • • •             | 10 cc 011<br>← n →<br>← n →  | --<br>--<br>-- | 10<br>10 | 11<br>11 | 03<br>03 | 03<br>05 |
| JR e     | PC ← PC+e                      | • • • • • •             | 00 011 000<br>← e-2 →        | 18<br>--       | 12       | 13       | 03       | 03       |
| JR C,e   | If CY=1,<br>PC ← PC+e          | • • • • • •             | 00 111 000<br>← e-2 →        | 38<br>--       | 07<br>12 | 08<br>13 | 02<br>03 | 02<br>03 |
| JR NC,e  | If CY=0,<br>PC ← PC+e          | • • • • • •             | 00 110 000<br>← e-2 →        | 30<br>--       | 07<br>12 | 08<br>13 | 02<br>03 | 02<br>03 |
| JR Z,e   | If Z=1,<br>PC ← PC+e           | • • • • • •             | 00 101 000<br>← e-2 →        | 28<br>--       | 07<br>12 | 08<br>13 | 02<br>03 | 02<br>03 |
| JR NZ,e  | If Z=0,<br>PC ← PC+e           | • • • • • •             | 00 100 000<br>← e-2 →        | 20<br>--       | 07<br>12 | 08<br>13 | 02<br>03 | 02<br>03 |
| JP (HL)  | PC ← HL                        | • • • • • •             | 11 101 001                   | E9             | 04       | 05       | 01       | 03       |
| JP (IX)  | PC ← IX                        | • • • • • •             | 11 011 101<br>11 101 001     | DD<br>E9       | 08       | 10       | 02       | 04       |
| JP (IY)  | PC ← IY                        | • • • • • •             | 11 111 101<br>11 101 001     | FD<br>E9       | 08       | 10       | 02       | 04       |
| DJNZ e   | B ← B-1<br>If B0,<br>PC ← PC+e | • • • • • •             | 00 010 000<br>← e-2 →        | 10<br>--       | 08<br>13 | 09<br>14 | 02<br>03 | 02<br>03 |

|    |     |     |     |     |     |     |     |     |
|----|-----|-----|-----|-----|-----|-----|-----|-----|
|    | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| cc | NZ  | Z   | NC  | C   | PO  | PE  | P   | M   |

TZ – Z80 T Cycles  
 Z1 – Z80 + M1  
 TR – R800 T Cycles  
 RW – R800 + Wait

Flags notation:

• = Flag not affected

NOTE: When there are two descriptions of cycles, they refer to the two conditions that the instruction can assume. Thus, for LDIR, the time in T cycles for the Z80 is 21; when BC reaches 0, 16 T cycles are spent.

## 10.13 – CALL AND RETURN GROUP

| Mnemonic   | Operation  | C Z $\overline{P}$ / <sub>V</sub> S N H | Binary                       | Hex            | TZ | Z1 | TR | RW       |
|------------|--|---|------------------------------|----------------|----|----|----|----------|
| CALL nn    | (SP-1) ← PC <sub>H</sub><br>(SP-2) ← PC <sub>L</sub><br>PC ← nn                                      | • • • • •                               | 11 001 101<br>← n →<br>← n → | CD<br>--<br>-- | 17 | 18 | 05 | 08<br>07 |
| CALL cc,nn | If cc=true,<br>(SP-1) ← PC <sub>H</sub><br>(SP-2) ← PC <sub>L</sub><br>PC ← nn                       | • • • • •                               | 11 cc 100<br>← n →<br>← n →  | CD<br>--<br>-- | 10 | 11 | 03 | 07<br>03 |
|            |  |   |                              |                | 17 | 18 | 05 | 08       |
| RET        | PC <sub>H</sub> ← (SP+1)<br>PC <sub>L</sub> ← (SP)   | • • • • •                               | 11 101 001                   | C9             | 10 | 11 | 03 | 05       |
| RET cc     | If cc=true,<br>PC <sub>H</sub> ← (SP+1)<br>PC <sub>L</sub> ← (SP)                                    | • • • • •                               | 11 cc 000                    | --             | 05 | 06 | 01 | 01       |
|            |  |   |                              |                | 11 | 12 | 03 | 05       |
| RETI       | Return from<br>Interrupt   | • • • • •                               | 11 101 101<br>01 001 101     | ED<br>4D       | 14 | 16 | 05 | 07       |
| RETN       | Return from<br>non maskable<br>Interrupt   | • • • • •                               | 11 101 101<br>01 000 101     | ED<br>45       | 14 | 16 | 05 | 07       |
| RST p      | (SP-1) ← PC <sub>H</sub><br>(SP-2) ← PC <sub>L</sub><br>PC <sub>H</sub> ← 0<br>PC <sub>L</sub> ← t*8 | • • • • •                               | 11 t 111                     | --             | 11 | 12 | 04 | 06<br>07 |

|    |     |     |     |     |     |     |     |     |
|----|-----|-----|-----|-----|-----|-----|-----|-----|
|    | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| cc | NZ  | Z   | NC  | C   | PO  | PE  | P   | M   |
| p  | 00H | 08H | 10H | 18H | 20H | 28H | 30H | 38H |

TZ - Z80 T Cycles  
 Z1 - Z80 + M1  
 TR - R800 T Cycles  
 RW - R800 + Wait

Flags notation:

• = Flag not affected

**NOTE:** When there are two descriptions of cycles, they refer to the two conditions that the instruction can assume. Thus, for LDIR, the time in T cycles for the Z80 is 21; when BC reaches 0, 16 T cycles are spent.  
**NOTE1:** Tests have shown that a CALL followed by a series of NOPs takes 8 cycles, while if followed by a combined RET or POP AF it takes 12 cycles (7 for CALL + 5 for RET/POP AF). This also applies to the RST (only applicable to the RW highlighted value for the R800).

## 10.14 – INPUT AND OUTPUT GROUP

| Mnemonic   | Operation  | C Z $\frac{P}{V}$ S N H                            | Binary                                   | Hex      | TZ | Z1 | TR | RW            |
|------------|--|--|--|----------|----|----|----|---------------|
| IN A, (n)  | $A \leftarrow (n)$   | • • • • •  | 11 011 011<br>$\leftarrow n \rightarrow$ | 28<br>-- | 11 | 12 | 03 | 10<br>09      |
| IN r, (C)  | $r \leftarrow (C)$   | • $\updownarrow$ P $\updownarrow$ 0 $\updownarrow$ | 11 101 101<br>01 r 000                   | ED<br>-- | 12 | 14 | 03 | 10<br>09      |
| INI        | (HL) $\leftarrow$ (C)<br>B $\leftarrow$ B-1<br>HL $\leftarrow$ HL+1              | • $\updownarrow$ ? ? 1 ?                           | 11 101 101<br>10 100 010                 | ED<br>A2 | 16 | 18 | 04 | 12<br>11      |
| INIR       | (HL) $\leftarrow$ (C)<br>B $\leftarrow$ B-1<br>HL $\leftarrow$ HL+1<br>{Até B=0} | • 1 ? ? 1 ?  | 11 101 101<br>10 110 010                 | ED<br>B2 | 21 | 23 | 04 | ?<br>11<br>12 |
| IND        | (HL) $\leftarrow$ (C)<br>B $\leftarrow$ B-1<br>HL $\leftarrow$ HL-1              | • $\updownarrow$ ? ? 1 ?                           | 11 101 101<br>10 101 010                 | ED<br>AA | 16 | 18 | 04 | 12<br>11      |
| INDR       | (HL) $\leftarrow$ (C)<br>B $\leftarrow$ B-1<br>HL $\leftarrow$ HL-1<br>{Até B=0} | • 1 ? ? 1 ?  | 11 101 101<br>10 111 010                 | ED<br>BA | 21 | 23 | 04 | ?<br>11<br>12 |
| OUT (n), A | $(n) \leftarrow A$   | • • • • •  | 11 010 111<br>$\leftarrow n \rightarrow$ | D3<br>-- | 11 | 03 | 03 | 10<br>9       |
| OUT (C), r | $(C) \leftarrow r$   | • • • • •  | 11 101 101<br>01 r 001                   | ED<br>-- | 11 | 12 | 03 | 10<br>9       |
| OUTI       | (C) $\leftarrow$ (HL)<br>B $\leftarrow$ B-1<br>HL $\leftarrow$ HL+1              | • $\updownarrow$ ? ? 1 ?                           | 11 101 101<br>10 100 011                 | ED<br>A3 | 16 | 18 | 04 | 12<br>11      |
| OTIR       | (C) $\leftarrow$ (HL)<br>B $\leftarrow$ B-1<br>HL $\leftarrow$ HL+1<br>{Até B=0} | • 1 ? ? 1 ?  | 11 101 101<br>10 110 011                 | ED<br>B3 | 21 | 23 | 04 | ?<br>11<br>12 |
| OUTD       | (C) $\leftarrow$ (HL)<br>B $\leftarrow$ B-1<br>HL $\leftarrow$ HL-1              | • $\updownarrow$ ? ? 1 ?                           | 11 101 101<br>10 101 011                 | ED<br>AB | 16 | 18 | 04 | 12<br>11      |

| Mnemonic | Operation  | C Z $\frac{P}{V}$ S N H | Binary                   | Hex      | TZ | Z1 | TR | RW |
|----------|--|-------------------------|--------------------------|----------|----|----|----|----|
| OTDR     | (C) $\leftarrow$ (HL)<br>B $\leftarrow$ B-1<br>HL $\leftarrow$ HL-1<br>{Até B=0} | • 1 ? ? 1 ?             | 11 101 101<br>10 111 011 | ED<br>BB | 21 | 23 | 04 | ?  |
|          |  |                         |                          |          | 16 | 18 | 03 | 12 |

|   |     |     |     |     |     |     |     |     |
|---|-----|-----|-----|-----|-----|-----|-----|-----|
|   | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| r | B   | C   | D   | E   | H   | L   | F   | A   |

TZ - Z80 T Cycles  
 Z1 - Z80 + M1  
 TR - R800 T Cycles  
 RW - R800 + Wait

#### Flags notation:

- = Flag not affected
- ↓ = Flag affected according operation results
- 0 = Flag off
- 1 = Flag on
- ? = Flag unknown
- P = the P/V flag contains the parity status. P=1 means the parity of the result is even; P=0 means it is odd.

- NOTE: In the INI, IND, OUTI e OUTD instructions, the flag Z is set when B-1=0 and is reset otherwise.
- NOTE1: In the 'IN A, (n)' e 'OUT (n), A' instructions, 'n' is sent to A0~A7 and A is sent to A8~A15. In the other instructions, 'C' content is sent to A0~A7 and 'B' content is sent to A8~A15.
- NOTE2: The I/O instructions are aligned to the bus clock, so an extra wait is inserted depending on the alignment. This means that between two OUTs there may be a reduction of one cycle (applicable only to the RW value highlighted for the R800).

## 10.15 – GENERAL PURPOSE AND CONTROL GROUPS

| Mnemonic | Operation                     | C Z $\frac{P}{V}$ S N H                         | Binary                   | Hex      | TZ | Z1 | TR | RW |
|----------|-------------------------------|---|--------------------------|----------|----|----|----|----|
| CCF      | $CY \leftarrow \overline{CY}$ | 1 . . . . 0 ?                                   | 00 111 111               | 3F       | 04 | 05 | 01 | 01 |
| CPL      | $A \leftarrow \overline{A}$   | . . . . . 1 1                                   | 00 101 111               | 2F       | 04 | 05 | 01 | 01 |
| DAA      | Converts A to BCD             | $\uparrow \downarrow P \downarrow . \downarrow$ | 00 100 111               | 27       | 04 | 05 | 01 | 01 |
| DI       | $IFF \leftarrow 0$            | . . . . . .                                     | 11 110 011               | F3       | 04 | 05 | 02 | 02 |
| EI       | $IFF \leftarrow 1$            | . . . . . .                                     | 11 111 011               | FB       | 04 | 05 | 01 | 01 |
| HALT     | Halts CPU                     | . . . . . .                                     | 01 110 110               | 76       | 04 | 05 | 02 | 02 |
| IM 0     | Interrupt mode 0              | . . . . . .                                     | 11 101 101<br>01 000 110 | ED<br>46 | 08 | 10 | 03 | 03 |
| IM 1     | Interrupt mode 1              | . . . . . .                                     | 11 101 101<br>01 010 110 | ED<br>56 | 08 | 10 | 03 | 03 |
| IM 2     | Interrupt mode 2              | . . . . . .                                     | 11 101 101<br>01 011 110 | ED<br>5E | 08 | 10 | 03 | 03 |
| NEG      | $A \leftarrow 0 - A$          | $\uparrow \downarrow V \downarrow 1 \downarrow$ | 00 101 101<br>01 000 100 | ED<br>44 | 08 | 10 | 02 | 02 |
| NOP      | No operation                  | . . . . . .                                     | 00 000 000               | 00       | 04 | 05 | 01 | 01 |
| SCF      | $CY \leftarrow 1$             | 1 . . . . 0 0                                   | 00 110 111               | 37       | 04 | 05 | 01 | 01 |

TZ - Z80 T Cycles  
 Z1 - Z80 + M1  
 TR - R800 T Cycles  
 RW - R800 + Wait

## Flags notation:

• = Flag not affected  
 $\uparrow \downarrow$  = Flag affected according operation results  
 0 = Flag off  
 1 = Flag on  
 ? = Flag unknown  
 V = the P/V flag contains the overflow status: V=1 -> overflow;  
 V=0 -> there was no overflow.  
 P = the P/V flag contains the parity status. P=1 means the  
 parity of the result is even; P=0 means it is odd.

NOTE: IFF indicates the flip-flop of interrupt activation circuit.  
 CY indicates the flip-flop of the overflow circuit.



## 11 – STANDARD CHIPS REGISTERS MAPS

### 11.1 – MAP OF THE REGISTERS OF THE V9918/38/58

| Regist. | b7 | b6    | b5 | b4  | b3    | b2    | b1 | b0 | Short description          |                    |                 |
|---------|----|-------|----|---|-------|-------|----|----|----------------------------|--------------------|-----------------|
| R#0     | W  | •     | •  | •   | •     | •     | m3 | EV | Mode Register #1 (9918)    |                    |                 |
|         |    | b7~b2 |    | Not used (always“000 000”)  |       |       |    |    |                            |                    |                 |
|         |    | b1    |    | m3: Screen mode (together with R#1)   |       |       |    |    |                            |                    |                 |
|         |    | b0    |    | EV: 0=disable external input; 1=enable  |       |       |    |    |                            |                    |                 |
|         |    | •     | DG | IE2   | IE1   | m5~m3 |    | •  | Mode register #1 (9938/58) |                    |                 |
|         |    | b7    |    | Not used (always 0)   |       |       |    |    |                            |                    |                 |
|         |    | b6    |    | DG: 0=normal; 1=color bus in input mode   |       |       |    |    |                            |                    |                 |
|         |    | b5    |    | IE2: Lightpen input (eliminated in the 9958)  |       |       |    |    |                            |                    |                 |
|         |    | b4    |    | IE1: 0=enable line interrupt #1; 1=disable  |       |       |    |    |                            |                    |                 |
|         |    | b3~b1 |    | M5~M3: Screen mode (together with R#1)  |       |       |    |    |                            |                    |                 |
|         |    | b0    |    | Not used (always 0)   |       |       |    |    |                            |                    |                 |
| R#1     | W  | 16K   | BL | IE0   | m2~m1 |       | BC | SI | MA                         | Mode register #2   |                 |
|         |    | b7    |    | (9918) → 0=4027(4K x 1-bit);<br>1=4108(8K x 1-bit)/4116(16K x 1-bit)                                |       |       |    |    |                            |                    |                 |
|         |    |       |    | (9938/58) → Not used (always 0)   |       |       |    |    |                            |                    |                 |
|         |    | b6    |    | BL: 0=screen off; 1=screen on   |       |       |    |    |                            |                    |                 |
|         |    | b5    |    | IE0: 9918: 0=enable interrupt; 1=disable interrupt<br>9938/58: 0=enable line interrup #0; 1=disable |       |       |    |    |                            |                    |                 |
|         |    | b4~b3 |    | M2~M1: Screen mode (together with R#0)  |       |       |    |    |                            |                    |                 |
|         |    |       |    | M5  | M4    | M3    | M2 | M1 | b4                         | b3→ (de R#25/9958) |                 |
|         |    |       |    | 0   | 0     | 0     | 1  | 0  | 0                          | 0                  | Screen 0 wth 40 |
|         |    |       |    | 0   | 1     | 0     | 1  | 0  | 0                          | 0                  | Screen 0 wth 80 |
|         |    |       |    | 0   | 0     | 0     | 0  | 0  | 0                          | 0                  | Screen 1        |
|         |    |       |    | 0   | 0     | 1     | 0  | 0  | 0                          | 0                  | Screen 2        |
|         |    |       |    | 0   | 0     | 0     | 0  | 1  | 0                          | 0                  | Screen 3        |
|         |    |       |    | 0   | 1     | 0     | 0  | 0  | 0                          | 0                  | Screen 4        |
|         |    |       |    | 0   | 1     | 1     | 0  | 0  | 0                          | 0                  | Screen 5        |
|         |    |       |    | 1   | 0     | 0     | 0  | 0  | 0                          | 0                  | Screen 6        |
|         |    |       |    | 1   | 0     | 1     | 0  | 0  | 0                          | 0                  | Screen 7        |
|         |    |       |    | 1   | 1     | 1     | 0  | 0  | 0                          | 0                  | Screen 8        |
|         |    |       |    | 1   | 1     | 1     | 0  | 0  | 1                          | 1                  | Screen 10/11    |
|         |    |       |    | 1   | 1     | 1     | 0  | 0  | 0                          | 1                  | Screen 12       |



|      |   |                              |   |            |   |                              |     |     |     |   |
|------|---|------------------------------|---|------------|---|------------------------------|-----|-----|-----|---|
| R#10 | W | 0                            | 0 | 0          | 0 | 0                            | a16 | a15 | a14 | Pattern color table address                             |
| R#11 | W | 0                            | 0 | 0          | 0 | 0                            | 0   | a16 | a15 | Sprites attributes table                                |
| R#12 | W | f3~f0                        |   |            |   | b3~b0                        |     |     |     | f3~f0 – Blink forecolor<br>b3~b0 – Blink backcolor      |
| R#13 | W | e3~e0<br>unit:<br>1/6 second |   |            |   | o3~o0<br>unit:<br>1/6 second |     |     |     | Blink R#7/R#12<br>e3~e0 – Even page<br>o3~o0 – Odd page |
| R#14 | W | 0                            | 0 | 0          | 0 | 0                            | a16 | a15 | a14 | Base VRAM address                                       |
| R#15 | W | •                            | • | •          | • | 0 ~ 9                        |     |     |     | Status register pointer                                 |
| R#16 | W | •                            | • | •          | • | 0 ~ 15                       |     |     |     | Palette register pointer                                |
| R#17 | W | AI                           | • | R#0 a R#46 |   |                              |     |     |     | Pointer control   |
| R#18 | W | Vert –8 a +7                 |   |            |   | Hor –8 a +7                  |     |     |     | Screen adjust   |
| R#19 | W | Line number (0 a 255)        |   |            |   |                              |     |     |     | Line interrupt register                                 |
| R#20 | W | 0                            | 0 | 0          | 0 | 0                            | 0   | 0   | 0   | Color Burst #1  |
| R#21 | W | 0                            | 0 | 1          | 1 | 1                            | 0   | 1   | 1   | Color Burst #2  |
| R#22 | W | 0                            | 0 | 0          | 0 | 0                            | 1   | 0   | 1   | Color Burst #3  |
| R#23 | W | Line number (0 a 255)        |   |            |   |                              |     |     |     | Vertical scroll / adjust                                |
| R#24 | W | ---,---                      |   |            |   |                              |     |     |     | This register not exist                                 |

### Registradores adicionados para o V9958

| R#25 | W | • | CMD | VDS   | YAE | YJK | WTE | MSK | SP | Mode register #5 |
|------|---|---|-----|---|-----|-----|-----|-----|----|------------------|
|      |   |   | b7  | Not used (always 0)   |     |     |     |     |    |                  |
|      |   |   | b6  | 0=VDP commands in 5~7 screen modes only<br>1=VDP commands in all screen modes |     |     |     |     |    |                  |
|      |   |   | b5  | VDS: 0=CPUCLK output; 1= VDS output   |     |     |     |     |    |                  |
|      |   |   | b4  | YAE: 0=YJK only; 1=YJK+RGB  |     |     |     |     |    |                  |
|      |   |   | b3  | YJK: 0=RGB mode; 1=YJK mode   |     |     |     |     |    |                  |
|      |   |   | b2  | WTE: 0=wait function off; 1=wait function active                              |     |     |     |     |    |                  |
|      |   |   | b1  | MSK: 0=scroll mask off; 1=scroll mask active                                  |     |     |     |     |    |                  |
|      |   |   | b0  | SP: 0=1-page horizontal scroll; 1=2-page hor scroll                           |     |     |     |     |    |                  |

|                                   |   |  |    |  |    |               |     |    |     |   |  |
|-----------------------------------|---|--|----|--|----|---------------|-----|----|-----|---|--|
| R#26                              | W | •  | •  | h8   | h7 | h6            | h5  | h4 | h3  | Horizontal scroll                                 |  |
| R#27                              | W | •  | •  | •  | •  | •             | h2  | h1 | h0  |   |  |
| Command Registers (V9938 e V9958) |   |  |    |  |    |               |     |    |     |   |  |
| R#32                              | W | x7   | x6 | x5   | x4 | x3            | x2  | x1 | x0  | Source horizontal coordinate (0 a 511)            |  |
| R#33                              | W | •  | •  | •  | •  | •             | •   | •  | x8  |   |  |
| R#34                              | W | y7   | y6 | y5   | y4 | y3            | y2  | y1 | y0  | Source vertical coordinate (0 a 1023)             |  |
| R#35                              | W | •  | •  | •  | •  | •             | •   | y9 | y8  |   |  |
| R#36                              | W | x7   | x6 | x5   | x4 | x3            | x2  | x1 | x0  | Destination horizontal coordinate (0 a 511)       |  |
| R#37                              | W | •  | •  | •  | •  | •             | •   | •  | x8  |   |  |
| R#38                              | W | y7   | y6 | y5   | y4 | y3            | y2  | y1 | y0  | Destination vertical coordinate (0 a 1023)        |  |
| R#39                              | W | •  | •  | •  | •  | •             | •   | y9 | y8  |   |  |
| R#40                              | W | x7   | x6 | x5   | x4 | x3            | x2  | x1 | x0  | Number of points in horizontal direction(0 a 511) |  |
| R#41                              | W | •  | •  | •  | •  | •             | •   | •  | x8  |   |  |
| R#42                              | W | y7   | y6 | y5   | y4 | y3            | y2  | y1 | y0  | Number of points in vertical direction (0 a 1023) |  |
| R#43                              | W | •  | •  | •  | •  | •             | •   | y9 | y8  |   |  |
| R#44                              | W | Cor (0~3; 0~15; 0~255)                       |    |  |    |               |     |    |     | Color register                                    |  |
| R#45                              | W | •  | MC | MD   | MS | DIY           | DIX | EQ | MAJ | Argument register                                 |  |
|                                   |   | b7<br>b6<br>b5<br>b4<br>b3<br>b2<br>b1<br>b0 |    | Not used (always0)<br>MXC: 0=VRAM; 1=expanded VRAM (std memory)<br>MXD: 0=VRAM; 1=expanded VRAM (dest memory)<br>MXS: 0=VRAM; 1=expand. VRAM (source memory)<br>DIY: 0=down; 1=up<br>DIX: 0=to right; 1=to left<br>EQ: 0=specified color; 1=other color (end of SRCH)<br>MAJ: 0=horizontal major side; 1=vertical major side |    |               |     |    |     |   |  |
| R#46                              | W | Command(0~15)                                |    |  |    | Logopr (0~15) |     |    |     | Command register                                  |  |
|                                   |   | b7~b4  |    | Command code (OP-CODE)<br>0 0 0 0 STOP Stop command<br>0 0 0 1-0 0 1 1 Not implemented   |    |               |     |    |     |   |  |

|                                       |   |       |  |           |                  |   |    |                               |    |                |  |  |
|---------------------------------------|---|-------|--|-----------|------------------|---|----|-------------------------------|----|----------------|--|--|
|                                       |   |       | 0 1 0 0 POINT Read point (dot) color code<br>0 1 0 1 PSET Draw a point and advance coord<br>0 1 1 0 SRCH Search point color code<br>0 1 1 1 LINE Draw a line (logical)<br>1 0 0 0 LMMV Draw a box (logical)<br>1 0 0 1 LMMM Transf. VRAM → VRAM (logical)<br>1 0 1 1 LMMC Transf. CPU → VRAM (logical)<br>1 1 0 0 HMMV Draw a box (bytes)<br>1 1 0 1 HMMM Transf. VRAM → VRAM (bytes)<br>1 1 1 0 YMMM Transf. VRAM → VRAM in Y direct.<br>1 1 1 1 HMMC Transf. CPU → VRAM (bytes)<br>b3 Cor transparente: 0=make logical operation<br>1=not make logical operation<br>b2~b0 Logopr 000 IMP DC = SC<br>001 AND DC = SC and DC<br>010 OR DC = SC or DC<br>011 XOR DC = SC xor DC<br>100 NOT DC = not (SC)<br>101~111 Not implemented |           |                  |   |    |                               |    |                |  |  |
| Status register (TMS9918/V9938/V9958) |   |       |  |           |                  |   |    |                               |    |                |  |  |
| S#0                                   | R | FLG   | 5S   | C         | 5° sprite (0~31) |   |    | Status register               |    |                |  |  |
|                                       |   | b7    | FLG: Vertical interrupt flag   |           |                  |   |    |                               |    |                |  |  |
|                                       |   | b6    | 5S: 0=normal; 1=more than 4 (ou 8) sprites in same line  |           |                  |   |    |                               |    |                |  |  |
|                                       |   | b5    | C: 0=normal; 1=two sprites colliding   |           |                  |   |    |                               |    |                |  |  |
|                                       |   | b4~b0 | Number of 5th (ou 9th) sprite  |           |                  |   |    |                               |    |                |  |  |
| Status register (V9938 e V9958)       |   |       |  |           |                  |   |    |                               |    |                |  |  |
| S#1                                   | R | LP    | KEY  | ID number |                  |   | HI | Lightpen / ID / Hor.Interrupt |    |                |  |  |
|                                       |   | b7    | LPF: 0=LP normal; 1=LP light detect (elim. V9958)  |           |                  |   |    |                               |    |                |  |  |
|                                       |   | b6    | LPK: 0=LP key not pressed; 1=it is pressed   |           |                  |   |    |                               |    |                |  |  |
|                                       |   | b5~b1 | MSX-video number ID  |           |                  |   |    |                               |    |                |  |  |
|                                       |   | b0    | FH: 0=Horizontal interrupt disabled; 1=enabled   |           |                  |   |    |                               |    |                |  |  |
| S#2                                   | R | TR    | VR   | HR        | BD               | • | •  | EO                            | CE | Command status |  |  |
|                                       |   | b7    | TR: 0=VDP not ready for dada; 1=VDP is ready   |           |                  |   |    |                               |    |                |  |  |
|                                       |   | b6    | VR: 0=frame is not being scanned; 1=it is  |           |                  |   |    |                               |    |                |  |  |
|                                       |   | b5    | HR: 0=Line is not being scanned; 1=it is   |           |                  |   |    |                               |    |                |  |  |
|                                       |   | b4    | BD: 0=SRCH not search; 1=SRCH sucess   |           |                  |   |    |                               |    |                |  |  |

|     |   |                        |   |    |    |    |    |    |    |  |
|-----|---|------------------------|---|----|----|----|----|----|----|--|
|     |   | b3~b2<br>b1<br>b0      | Not used (always 11)<br>EO: 0=1st screen showed; 1=2nd screen showed<br>CE: 0=VDP free; 1=VDP executing command |    |    |    |    |    |    |  |
| S#3 | R | x7                     | x6  | x5 | x4 | x3 | x2 | x1 | x0 | Coordinate X+12<br>(Sprites collision)               |
| S#4 | R | .                      | .   | .  | .  | .  | .  | .  | x8 |  |
| S#5 | R | y7                     | y6  | y5 | y4 | y3 | y2 | y1 | y0 | Coordinate Y+8<br>(Sprites collision)                |
| S#6 | R | .                      | .   | .  | .  | .  | .  | y9 | y8 |  |
| S#7 | R | Cor (0~3; 0~15; 0~255) |   |    |    |    |    |    |    | Color of specified point                             |
| S#8 | R | x7                     | x6  | x5 | x4 | x3 | x2 | x1 | x0 | Horizontal coordinate of the<br>point (SRCH command) |
| S#9 | R | .                      | .   | .  | .  | .  | .  | .  | x8 |  |

### 11.1.1 – Access ports for VDPs V9918/38/38

| Porta |     |     | b7            | b6 | b5   | b4               | b3 | b2 | b1 | b0                         | Short description          |  |
|-------|-----|-----|---------------|----|--|------------------|----|----|----|----------------------------|----------------------------|--|
| P#0   | 98H | R/W | Byte de dados |    |  |                  |    |    |    | Write/read VRAM data       |                            |  |
| P#1   | 99H | R   | FLG           | 5S | C  | 5° sprite (0~31) |    |    |    | Read status register       |                            |  |
|       |     |     | b7            |    | FLG: Vertical interrupt flag                         |                  |    |    |    |                            |                            |  |
|       |     |     | b6            |    | 5S: 0=normal; 1=more than 4 (ou 8) sprites same line |                  |    |    |    |                            |                            |  |
|       |     |     | b5            |    | C: 0=normal; 1=two sprites colliding                 |                  |    |    |    |                            |                            |  |
|       |     |     | b4~b0         |    | Number of 5th (ou 9th) sprite                        |                  |    |    |    |                            |                            |  |
| P#1   | 99H | W   | a7~a0 adress  |    |  |                  |    |    |    | Select VRAM address        |                            |  |
|       |     |     | 0             | f  | a13~a8 adress  |                  |    |    |    | f: 0=read; 1=write         |                            |  |
|       |     |     | Data byte     |    |  |                  |    |    |    | Write control reg (9918)   |                            |  |
|       |     |     | 1             | 0  | N° reg. (0~46)                                       |                  |    |    |    | Select register. (9938/58) |                            |  |
| P#2   | 9AH | W   | 0             | r  | r  | r                | 0  | b  | b  | b                          | Write in palette registers |  |
|       |     |     | 0             | 0  | 0  | 0                | 0  | g  | g  | g                          |                            |  |
| P#3   | 9BH | W   | Data byte     |    |  |                  |    |    |    | Write in indirect register |                            |  |

### 11.1.2 – Standard color chart

The color chart illustrated below is the standard color chart for the MSX1. For MSX2 onwards, the table is loaded when the computer is reset.

| Palette number | Color        | Red Level | Blue Level | Green Level |
|----------------|--------------|-----------|------------|-------------|
| 0              | Transparent  | 0         | 0          | 0           |
| 1              | Black        | 0         | 0          | 0           |
| 2              | Green        | 1         | 1          | 6           |
| 3              | Light Green  | 3         | 3          | 7           |
| 4              | Deep Blue    | 1         | 7          | 1           |
| 5              | Blue         | 2         | 7          | 3           |
| 6              | Deep Red     | 5         | 1          | 1           |
| 7              | Light Blue   | 2         | 7          | 6           |
| 8              | Red          | 7         | 1          | 1           |
| 9              | Light Red    | 7         | 3          | 3           |
| 10             | Yellow       | 6         | 1          | 6           |
| 11             | Light Yellow | 6         | 3          | 6           |
| 12             | Deep Green   | 1         | 1          | 4           |
| 13             | Purple       | 6         | 5          | 2           |
| 14             | Grey         | 5         | 5          | 5           |
| 15             | White        | 7         | 7          | 7           |





|      |     |                |   |               |    |                |       |     |                 |   |  |                          |  |
|------|-----|----------------|---|---------------|----|----------------|-------|-----|-----------------|---|--|--------------------------|--|
|      |     | b3             | VWM VRAM write digitization:<br>0=No, 1=While horizontal blank                                      |               |    |                |       |     |                 |   |  |                          |  |
|      |     | b2             | DMAE 0=DREQ output in high level, 1=Enabled   |               |    |                |       |     |                 |   |  |                          |  |
|      |     | b1~b0          | VSL1/0 00=64K x 4-bit, 128K total<br>01=128K x 8-bit, 256K total<br>10=256K x 4-bit, 512K total     |               |    |                |       |     |                 |   |  |                          |  |
| R#9  | R/W | •              | •   | •             | •  | •              | IE    | IH  | IV              | Interrupt control   |  |                          |  |
|      |     | b7~b3          | Not used (always "00 000")  |               |    |                |       |     |                 |   |  |                          |  |
|      |     | b2             | IECE 0-interrupt disabled, 1-enabled  |               |    |                |       |     |                 |   |  |                          |  |
|      |     | b1             | IEH 0-line interrupt disabled, 1-enabled  |               |    |                |       |     |                 |   |  |                          |  |
|      |     | b0             | IEV 0-frame interrupt disabled, 1-enabled   |               |    |                |       |     |                 |   |  |                          |  |
| R#10 | R/W | I7             | I6  | I5            | I4 | I3             | I2    | I1  | I0              | Interrupt line num (I9~I0)                                    |  |                          |  |
| R#11 | R/W | IE             | •   | •             | •  | •              | •     | I9  | I8              | IEHM – 0=specific line<br>1=all lines                         |  |                          |  |
| R#12 | R/W | •              | •   | •             | •  | ix3            | ix2   | ix1 | ix0             | Interrupt horizontal position<br>(IX) × (64) × (master clock) |  |                          |  |
| R#13 | W   | PLTM           |   | IE            | AI | PLTO5~2        |       |     |                 | Palette control   |  |                          |  |
|      |     | b7~b6          | PLTM - 00=palette; 01=256 colors; 10=YJK; 11=YUV  |               |    |                |       |     |                 |   |  |                          |  |
|      |     | b5             | YAE – 0=YJK/YUV only; 1=YJK/YUV + RGB   |               |    |                |       |     |                 |   |  |                          |  |
|      |     | b4             | PLTAIH – Autoinc palette read (0=Yes, 1=No)   |               |    |                |       |     |                 |   |  |                          |  |
|      |     | b3~b0          | PLTO5-2 – Palette offset (0 a 15)   |               |    |                |       |     |                 |   |  |                          |  |
| R#14 | W   | PLTA5~0        |   |               |    |                | PL2~1 |     | Palette control |   |  |                          |  |
|      |     | b7~b2          | PLTA – Num of the color in the palette (0 a 63)   |               |    |                |       |     |                 |   |  |                          |  |
|      |     | b1~b0          | PLTP – 00-Red; 01-Green; 10-Blue; 11-N/C  |               |    |                |       |     |                 |   |  |                          |  |
| R#15 | R/W | •              | •   | b5            | b4 | b3             | b2    | b1  | b0              | Backcolor   |  |                          |  |
| R#16 | R/W | ADJV (-8 a +7) |   |               |    | ADJH (-8 a +7) |       |     |                 | Screen adjust   |  |                          |  |
| R#17 | R/W | SCAY (b7~b0)   |   |               |    |                |       |     |                 | Scroll control  |  |                          |  |
| R#18 | R/W | ROLL           | •   | SCAY (b12~b8) |    |                |       |     |                 |   |  | Y-coord in the plane "A" |  |
|      |     | b7~b0          | SCAY – Y-coordinate of start of display for the "A" plane of P1 mode and the screens of other modes |               |    |                |       |     |                 |   |  |                          |  |
|      |     | b4~b0          | Not used (always "0")   |               |    |                |       |     |                 |   |  |                          |  |
|      |     | b5             | ROLL – Direct. Y scroll: 00-full screen 10-512 lines<br>01-256 lines 11-N/C                         |               |    |                |       |     |                 |   |  |                          |  |

|      |     |                                      |   |  |    |              |            |   |  |                              |  |  |
|------|-----|--------------------------------------|---|--|----|--------------|------------|---|--|------------------------------|--|--|
| R#19 | R/W | .                                    | . | .  | .  | .            | SX (b2~b0) | Scroll control<br>“A” plane X-coord and B0~B7 |  |                              |  |  |
| R#20 | R/W | SCAX (b10~b3)                        |   |  |    |              |            |   |  |                              |  |  |
| R#21 | R/W | SCBY (b7~b0)                         |   |  |    |              |            |   | Scroll control                             |                              |  |  |
| R#22 | R/W | A                                    | B | .  | .  | .            | .          | .   | b8   | “B” plane Y-coord            |  |  |
|      |     | b7~b0<br>b0<br>b7<br>b6<br>b5~b1     |   | SCBY – Start Y-coord for showing “B” plane in the P1 mode (b8~b0)<br>SDB – If=1, disable sprites and “A” plane<br>SDB – If=1, disable sprites and “B” plane<br>Not used (always “00 000”)  |    |              |            |   |  |                              |  |  |
| R#23 | R/W | .                                    | . | .  | .  | .            | BX(b2~b0)  |   | SCBX - Start X-coord for showing “B” plane |                              |  |  |
| R#24 | R/W | .                                    | . | SCBX (b8~b3)   |    |              |            |   |  |                              |  |  |
| R#25 | R/W | .                                    | . | .  | .  | a17          | a16        | a15   | .  | Pattern sprites address (P1) |  |  |
|      |     | .                                    | . | .  | .  | a18          | a17        | a16   | a15  | Pattern sprites address (P2) |  |  |
| R#26 | R/W | .                                    | . | .  | VR | PNS          | PL         | PD  | PNE  | LCD panel control            |  |  |
|      |     | b7~b5<br>b4<br>b3<br>b2<br>b1<br>b0  |   | Not used (always “000”)<br>VRI 0=equal CRT 1=one line vertical blank<br>PNSL 0=400 vertical points 1=480 vert points<br>PLVO 0=Greyscale (D3~0 pins)<br>1=Color (CB7~0 pins)<br>PDUAL 0=one screen 1=two screens<br>PNEN 0=CRT cycle 1=LCD cycle |    |              |            |   |  |                              |  |  |
| R#27 | R/W | .                                    | . | .  | .  | PRY          |            | PRX   |  | P1 mode priority             |  |  |
|      |     | b7~b4<br>b3~b2<br>b1~b0              |   | Not used (always “0000”)<br>PRX – 00 – X=256 10 – X=128<br>01 – X=64 11 – X=192<br>PRY – 00 – Y=256 10 – Y=128<br>01 – Y=64 11 – Y=192   |    |              |            |   |  |                              |  |  |
| R#28 | W   | .                                    | . | .  | .  | CSPO (b5~b2) |            |   | Cursor palette offset                      |                              |  |  |
|      |     | The R#29 to R#31 registers not exist |   |  |    |              |            |   |  |                              |  |  |


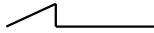
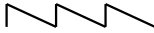

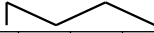
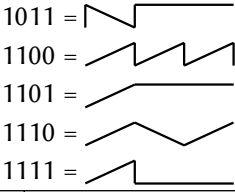
|             |  |   |                                 |   |   |               |             |           |   |                           |  |  |
|-------------|--|---|---------------------------------|---|---|---------------|-------------|-----------|---|---------------------------|--|--|
| R#32        | W  | SX, SA, KA (b7~b0)  |                                 |   |   |               |             |           | VDP Commands – SOURCE<br>X,Y-coordinates (SX, SY)<br>Linear address (SA)<br>Kanji-ROM address (KA)  |                           |  |  |
| R#33        | W  | •   | •                               | • | • | •             | SX(b10~b8)  |           |   |                           |  |  |
| R#34        | W  | SY (b7~b0); SA, KA (b15~b8)   |                                 |   |   |               |             |           |   |                           |  |  |
| R#35        | W  | •   | •                               | • | • | SY (b7~b0)    |             |           |   |                           |  |  |
|             |  | •   | •                               | • | • | •             | SA(b18~b16) |           |   |                           |  |  |
|             |  | •   | •                               | • | • | •             | •           | K(b17~16) |   |                           |  |  |
| R#36        | W  | DX, DA (b7~b0)  |                                 |   |   |               |             |           | VDP Commands – DEST<br>X,Y-coordinates (DX, DY)<br>Linear address (DA)  |                           |  |  |
| R#37        | W  | •   | •                               | • | • | •             | DX(b10~b8)  |           |   |                           |  |  |
| R#38        | W  | DY (b7~b0); DA (b15~b8)   |                                 |   |   |               |             |           |   |                           |  |  |
| R#39        | W  | •   | •                               | • | • | DY (b7~b0)    |             |           |   |                           |  |  |
|             |  | •   | •                               | • | • | •             | DA(b18~b16) |           |   |                           |  |  |
| R#40        | W  | NX, NA, MJ (b7~b0)  |                                 |   |   |               |             |           | VDP Commands – DIVS<br>Number of pixels to transfer XY (NX, NY)<br>No of bytes to transfer (NA)<br>Major side of the line (MJ)<br>Minor side of the line (MI) |                           |  |  |
| R#41        | W  | •   | •                               | • | • | MJ (b11~b8)   |             |           |   |                           |  |  |
|             |  | •   | •                               | • | • | •             | NX(b10~b8)  |           |   |                           |  |  |
| R#42        | W  | NY, MI (b7~b0); NA (b15~b8)   |                                 |   |   |               |             |           |   |                           |  |  |
| R#43        | W  | •   | •                               | • | • | NY,MI(b11~b8) |             |           |   |                           |  |  |
|             |  | •   | •                               | • | • | •             | NA(b18~b16) |           |   |                           |  |  |
| R#44        | W  | •   | •                               | • | • | DIY           | DIX         | NEQ       | MAJ   | Argument reg (write only) |  |  |
|             |  | b7~b4   | Not used (always “0000”)        |   |   |               |             |           |   |                           |  |  |
|             |  | b3  | DIY: 0 – Down; 1 – Up;          |   |   |               |             |           |   |                           |  |  |
|             |  | b2  | DIX: 0 – To right; 1 – To left. |   |   |               |             |           |   |                           |  |  |
|             |  | Note: BMXL and BMLX are fixed em “+” and BMLL, X and Y are fixed in same direction.         |                                 |   |   |               |             |           |   |                           |  |  |
| b1          | NEQ (SRCH end): 0=spec color 1=other color   |   |                                 |   |   |               |             |           |   |                           |  |  |
| b0          | MAJ (LINE maj side): 0=horizontal 1=vertical |   |                                 |   |   |               |             |           |   |                           |  |  |
| R#45        | W  | •   | •                               | • | T | Logopr        |             |           | Logical operation   |                           |  |  |
| b7~b5<br>b4 |  | Always “000”<br>Transparent color: 0=make logical operation<br>1=not make logical operation |                                 |   |   |               |             |           |   |                           |  |  |







### 11.3 – MAP OF PSG REGISTERS (AY-3-8910)

| Reg  | b7     | b6  | b5  | b4 | b3 | b2  | b1 | b0 | Short description                                  |
|------|--------|-----|---|----|----|---|----|----|--|
| R#0  | 7      | 6   | 5   | 4  | 3  | 2   | 1  | 0  | Channel “A” frequency                              |
| R#1  | •      | •   | •   | •  | 11 | 10  | 9  | 8  | 111860,87 / F_num (b11~b0)                         |
| R#2  | 7      | 6   | 5   | 4  | 3  | 2   | 1  | 0  | Channel “B” frequency                              |
| R#3  | •      | •   | •   | •  | 11 | 10  | 9  | 8  | 111860,87 / F_num (b11~b0)                         |
| R#4  | 7      | 6   | 5   | 4  | 3  | 2   | 1  | 0  | Channel “C” frequency                              |
| R#5  | •      | •   | •   | •  | 11 | 10  | 9  | 8  | 111860,87 / F_num (b11~b0)                         |
| R#6  | •      | •   | •   | 4  | 3  | 2   | 1  | 0  | White noise frequency<br>111860,87 / F_num (b4~b0) |
| R#7  | ioB    | ioA | rC  | rB | rA | tC  | tB | tA | Enable/disable sounds                              |
|      | b0~b2  |     | Enable/disable tones (0=enable)   |    |    |   |    |    |  |
|      | b5~b4  |     | Enable/disable white noise (0=enable)   |    |    |   |    |    |  |
|      | b6     |     | Configures “A” I/O port (0=in, 1=out)   |    |    |   |    |    |  |
|      | b7     |     | Configures “B” I/O port (0=in, 1=out)   |    |    |   |    |    |  |
| R#8  | •      | •   | •   | m  | v  | v   | v  | v  | Volume of channel A                                |
| R#9  | •      | •   | •   | m  | v  | v   | v  | v  | Volume of channel B                                |
| R#10 | •      | •   | •   | m  | v  | v   | v  | v  | Volume of channel C                                |
|      | b7~b5  |     | Not used (always “000”)   |    |    |   |    |    |  |
|      | b4     |     | 0=Not use envelope; 1=Use envelope  |    |    |   |    |    |  |
|      | b3~b0  |     | 0000=Minimum volume; 1111=Maximum volume  |    |    |   |    |    |  |
| R#11 | 7      | 6   | 5   | 4  | 3  | 2   | 1  | 0  | Envelope frequency                                 |
| R#12 | 15     | 14  | 13  | 12 | 11 | 10  | 9  | 8  | 6983,3 / F_num (b15~b0)                            |
| R#13 | •      | •   | •   | •  | e  | e   | e  | e  | Envelope shape                                     |
|      | b7~b4  |     | Not used (always “0000”)  |    |    |   |    |    |  |
|      | b3~b0  |     | Defines envelope shape:   |    |    |   |    |    |  |
|      | 00xx = |     |  |    |    | 1011 =  |    |    |  |
|      | 01xx = |     |  |    |    | 1100 =  |    |    |  |
|      | 1000 = |     |  |    |    | 1101 =  |    |    |  |
|      | 1001 = |     |  |    |    | 1110 =  |    |    |  |
|      | 1010 = |     |  |    |    | 1111 =  |    |    |  |
|      |        |     |   |    |    |  |    |    |  |
| R#14 | a7     | a6  | a5  | a4 | a3 | a2  | a1 | a0 | Read/write “A” I/O port                            |
| R#15 | b7     | b6  | b5  | b4 | b3 | b2  | b1 | b0 | Read/write “B” I/O port                            |





### 11.4 – MAP OF FM-OPLL REGISTERS (YM2413)

| Reg   | b7            | b6    | b5  | b4          | b3       | b2       | b1  | b0                     | Short Description  |  |
|-------|---------------|-------|---|-------------|----------|----------|---|------------------------|--|--|
| \$00H | AM            | VIB   | EGT   | KSR         | Multiple |          |   | → (m) – Modulated wave |  |  |
| \$01H | AM            | VIB   | EGT   | KSR         | Multiple |          |   | → (c) – Carrier wave   |  |  |
|       | b7            | b6    | b5  | b4          | b0~b3    |          |   |                        | AM: 0=tremolo off; 1=tremolo on<br>VIB: 0=vibrato off; 1=vibrato on<br>EGT: 0=decaying sound; 1=sustained sound<br>KSR: 0=same level; 1=frequency attenuation (KSL)<br>Multiplication factor (0=1/2, 1=1, 2=2, ....., 15=15) |  |
| \$02H | KSL(m)        |       | Total level modul. (m)  |             |          |          | Instrument definition   |                        |  |  |
|       | b6~b7         | b6~b0 | KSL (m): 00=0dB/oct, 01=1,5dB, 10=3dB, 11=6dB<br>Total level: b0=0,75dB, b1=1,5dB, ....., b5=24dB |             |          |          |   |                        |  |  |
| \$03H | KSL(c)        |       | •   | DC          | DM       | Feedback |   | Instrument definition  |  |  |
|       | b6~b7         | b5    | b4  | b3          | b2~b0    |          | KSL (c): 00=0dB/oct, 01=1,5dB, 10=3dB, 11=6dB<br>Not used (always 0)<br>DC: 0=integer carrier wave, 1=half-wave<br>DM: 0=integer modulated wavw, 1=half-wave<br>Feedback: (0=0; 1=π/16; 2=π/8; ...; 6=2π; 7=4π) |                        |  |  |
| \$04H | Attack (m)    |       |   | Decay (m)   |          |          | Attack (0dB a 48dB → min. 0,14 mS; max 1730 mS)   |                        |  |  |
| \$05H | Attack (c)    |       |   | Decay (c)   |          |          | Decay (0dB a 48dB → min. 1,27 mS; max 20 926 mS)  |                        |  |  |
| \$06H | Sustain (m)   |       |   | Release (m) |          |          | Sustain (b7=24dB, b6=12dB, b5=6dB, b4=3dB)  |                        |  |  |
| \$07H | Sustain (c)   |       |   | Release (c) |          |          | Release (0dB a 48dB → min. 1,27 mS; max 28 926 mS)  |                        |  |  |
| \$0EH | •             | •     | R   | BD          | SD       | TOM      | TCY   | HH                     | Rhythm control   |  |
|       | b7~b6         | b5    | b0~b4   |             |          |          |   |                        |  | Not used (always “00”)<br>0=melody mode; 1=rhythm mode<br>0=rhythm instrument off; 1=on<br>BD–Bass Drum    SD–Snare Drum    TOM–Tom tom<br>TCY–Top cymbal    HH–Hi-hat |
| \$0FH | Test register |       |   |             |          |          |   | OPLL test              |  |  |



## 11.5 – MSX-AUDIO REGISTERS MAP (Y8950)

| Reg            | b7  | b6  | b5    | b4  | b3  | b2    | b1  | b0  | Short Description   |
|----------------|---|-----|-------|-----|-----|-------|-----|-----|---|
| \$01H          | Test  |     |       |     |     |       |     |     | Test register   |
| \$02H<br>\$03H | 1st Timer (80 $\mu$ S)<br>2nd Timer (320 $\mu$ S)       |     |       |     |     |       |     |     | Time registers  |
| \$04H          | IRQ   | T1M | T2M   | EOS | BR  | •     | ST2 | ST1 | Flags register  |
|                | b7  | b6  | b5    | b4  | b3  | b2    | b1  | b0  | IRQ – If write 1, reset all flags.<br>T1M – If write 1, b0 will reset.<br>T2M – If write 1, b1 will reset.<br>EOS – B3 mask, indicating the end of current operation<br>BR – ADPCM / Audio memory mask (1=enable)<br>Not used (always0)<br>ST2 – \$03 Start/stop control (1=start counter)<br>ST1 – \$02 Start/stop control (1=start counter) |
| \$05H<br>\$06H | External keyboard (input)<br>External keyboard (output) |     |       |     |     |       |     |     | Registers for access to external musical keyboard   |
| \$07H          | STA   | REC | MEM   | REP | OFF | •     | •   | RST | Control register (1)  |
|                | b7  | b6  | b5    | b4  | b3  | b2~b1 | b0  |     | STA – Must be 1 to start data read/write<br>REC – Must be 1 to write data in the memory<br>MEM – Must be 1 to access audio memory<br>REP – When 1, enable ADPCM data repeat<br>OFF – When 1, cut off audio output<br>Not used (always“00”)<br>RST – When 1, puts ADPCM in the initial state   |
| \$08H          | CSM   | SEL | •     | •   | SAM | DAD   | 64K | ROM | Control register (2)  |
|                | b7  | b6  | b5~b4 | b3  | b2  | b1    | b0  |     | CSM – 1=composite sinusoisal modulation mode<br>SEL – External keyboard octaves separation point<br>Not used (always“00”)<br>SAM – 0=start DA conversion; 1=start AD conversion<br>DAD – 0=conv. AD / mus. output; 1=\$15~\$16 → output<br>64K – Memory size: 0=256K; 1=64K<br>ROM – Memory type: 0=RAM; 1=ROM                                |





|                     |       |   |   |     |          |   |     |  |                 |
|---------------------|-------|---|---|-----|----------|---|-----|--|-----------------|
| \$C0H<br>⋮<br>\$C8H | •     | •   | •   | •   | Feedback |   | CON | FM Feedback factor and connection type |                 |
|                     | b7~b4 |   | Not used (always“0000”)                         |     |          |   |     |  |                 |
|                     | b3~b1 |   | Feedback (0=0; 1=π/16; 2=π/8; ...; 6=2π; 7=4π)  |     |          |   |     |  |                 |
|                     | b0    |   | Operators connection type (0=série; 1=paralelo) |     |          |   |     |  |                 |
| STAT                | INT   | T1  | T2  | EOS | BUF      | • | •   | PCM                                    | Status register |
|                     | b7    | Will be 1 when one or more bits b3 to b6 contains 1                 |   |     |          |   |     |  |                 |
|                     | b6    | Will be 1 after timer 1 ends counting (\$02)                        |   |     |          |   |     |  |                 |
|                     | b5    | Will be 1 after timer 2 ends counting (\$03)                        |   |     |          |   |     |  |                 |
|                     | b4    | Will be 1 when ADPCM analysis/synthesis ends                        |   |     |          |   |     |  |                 |
|                     | b3    | Will be 1 at the end of read/write/analysis/synthesis               |   |     |          |   |     |  |                 |
|                     | b2-b1 | Not used (always“00”)   |   |     |          |   |     |  |                 |
|                     | b0    | Will be 1 during the ADPCM analysis/synthesis (if b7 of \$07 are 1) |   |     |          |   |     |  |                 |

### 11.5.1 – MSX-Audio access ports

| Porta |     | b7                          | b6 | b5 | b4  | b3  | b2 | b1 | b0  | Short Description                     |
|-------|-----|-----------------------------|----|----|-----|-----|----|----|-----|---------------------------------------|
| C0H   | W   | Register number (01H a C8H) |    |    |     |     |    |    |     | Select register                       |
|       | R   | INT                         | T1 | T2 | EOS | BUF | •  | •  | PCM | Read status register                  |
| C1H   | W/R | Data byte                   |    |    |     |     |    |    |     | Write/read data in the/from MSX Audio |

## 11.6 – MAP OF THE OPL4 REGISTERS (YMF278)

### 11.6.1 – Register Array #0

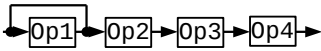
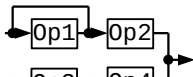
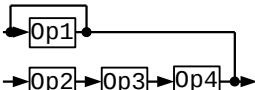
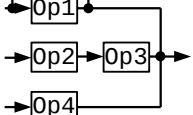
| FM generator – Register Array 0 (A1 = “1”) |   |   |             |     |                    |    |     |   |                        |  |
|--|---|---|-------------|-----|--------------------|----|-----|---|------------------------|--|
| Reg  | b7  | b6  | b5          | b4  | b3                 | b2 | b1  | b0  | Short Description      |  |
| \$00H<br>\$01H                             | Test  |   |             |     |                    |    |     |   | Test registers         |  |
| \$02H<br>\$03H                             | 1st Timer (80,8 μS)<br>2nd Timer (323,1 μS) |   |             |     |                    |    |     |   | Time registers         |  |
| \$04H                                      | RST   | MT1   | MT2         | •   | •                  | •  | ST2 | ST1   | Flag register          |  |
|  | b7<br>b6<br>b5<br>b4-b2<br>b1<br>b0         | RST – If write 1, resets b5, b6 and b7.<br>MT1 – If write 1, b0 will be 0.<br>MT2 – If write 1, b1 will be 0.<br>Not used (always“000”)<br>ST2 – \$03 Start/stop control (1=start counter)<br>ST1 – \$032Start/stop control (1=start counter) |             |     |                    |    |     |   |                        |  |
| \$08H                                      | •   | NTS   | •           | •   | •                  | •  | •   | •   | Keyboard configuration |  |
|  | b7<br>b6<br><br>b5~b0                       | Not used (always“0”)<br>NTS – If 0, the separation point is determined by the higher 2 bits of F_number. If 1, the separation point is determined only by the MSB of F_number.<br>Not used (always“000 000”)                                  |             |     |                    |    |     |   |                        |  |
| \$20H<br>⋮<br>\$35H                        | AM  | VIB   | EGT         | KSR | Multiple           |    |     | Instruments definition  |                        |  |
|  | b7<br>b6<br>b5<br>b4<br>b3~b0               | AM (1=tremolo on (Frequency: 3,7Hz)<br>VIB (1=vibrato on (Frequency: 6,4Hz)<br>EG-TYP (0=decaying sound; 1=sustained sound)<br>If 0, KSR→0~3; If 1, KSR→0~15<br>Multiplication factor (0=1/2, 1=1, 2=2, 3=3, ..., 15=15)                      |             |     |                    |    |     |   |                        |  |
| \$40H<br>⋮<br>\$55H                        | KSL   |   | Total level |     |                    |    |     | KSL (00= 0dB/octave,<br>01=1,5dB, 10=3dB, 11=6dB)<br>Total level (b0=0,75dB,<br>b1=1,5dB ..... b5=24dB) |                        |  |
| \$60H<br>⋮<br>\$75H                        | Attack Rate<br>(AR)                         |   |             |     | Decay Rate<br>(DR) |    |     | Attack (0dB a 96dB →<br>min. 0,2 mS; max 2826 mS)<br>Decay (0dB a 96dB →<br>min. 2,4 mS; max 39 280 mS) |                        |  |

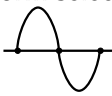
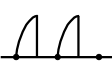
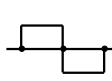
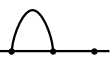
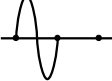
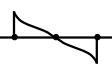








|                     |                               |     |  |                      |          |                        |   |     |                    |  |
|---------------------|-------------------------------|-----|--|----------------------|----------|------------------------|---|-----|--------------------|--|
| \$04H               | •                             | •   | Connection SEL   |                      |          |                        | 4-operators selection   |     |                    |  |
|                     | b7~b6<br>b5~b0                |     | Not used (always“00”)<br>Enable 4-operators mode for the respective slot:<br>Bit:    b5 b4 b3 b2 b1 b0<br>Slot:    6 5 4 3 2 1   |                      |          |                        |   |     |                    |  |
| \$05h               | •                             | •   | •  | •                    | •        | •                      | NEW2  | NEW | Expansion register |  |
|                     | b7~b2<br>b1<br>b0             |     | Not used (always“000 000”)<br>If 1, enable OPL4 mode (Register Array 1)<br>If 1, enable OPL3 mode (Register Array 0)   |                      |          |                        |   |     |                    |  |
| \$20H<br>⋮<br>\$35H | AM                            | VIB | EGT  | KSR                  | Multiple |                        |   |     |                    |  |
|                     | b7<br>b6<br>b5<br>b4<br>b3~b0 |     | AM (1=trêmolo on (frequency: 3,7Hz)<br>VIB (1=vibrato on (frequency: 6,4Hz)<br>EG-TYP (0=decaying sound; 1=sustained sound)<br>If 0, KSR→0~3; If 1, KSR→0~15<br>Multiplication factor (0=1/2, 1=1, 2=2, 3=3, ..., 15=15) |                      |          |                        |   |     |                    |  |
| \$40H<br>⋮<br>\$55H | KSL                           |     | Nível total  |                      |          |                        | KSL (00= 0dB/octave,<br>01=1,5dB, 10=3dB, 11=6dB)<br>Total level (b0=0,75dB,<br>b1=1,5dB ..... b5=24dB) |     |                    |  |
| \$60H<br>⋮<br>\$75H | Attack Rate<br>(AR)           |     |  | Decay Rate<br>(DR)   |          |                        | Attack (0dB a 96dB →<br>min. 0,2 mS; max 2826 mS)<br>Decay (0dB a 96dB →<br>min. 2,4 mS; max 39 280 mS) |     |                    |  |
| \$80H<br>⋮<br>\$95H | Sustain Level<br>(SL)         |     |  | Release Rate<br>(RL) |          |                        | Sustain (b7=24dB, b6=12dB,<br>b5=6dB, b4=3dB)<br>Release (0dB a 96dB →<br>min. 2,4 mS; max 39 280 mS)   |     |                    |  |
| \$A0H<br>⋮<br>\$A8H | Frequency (LSB 8 bits)        |     |  |                      |          |                        | Frequency (b7-b8)   |     |                    |  |
| \$B0H<br>⋮<br>\$B8H | •                             | •   | KEY  | Octave               |          | Freq.<br>MSB<br>2 bits | Freq. MSB 2 bits (FM)<br>Octave (FM)<br>Key on/off (FM)   |     |                    |  |
|                     | b7~b6<br>b5<br>b4~b2          |     | Not used (always“00”)<br>0=key off; 1=key on (active voice)<br>Defines the octave. The fourth is 011.  |                      |          |                        |   |     |                    |  |

|  |   |   |   |   |          |             |                 |                              |  |  |  |  |  |  |
|--|---|---|---|---|----------|-------------|-----------------|------------------------------|--|--|--|--|--|--|
|  | b1~b0   | Frequency MSB 2 bits. The C central note of 440 Hz is obtained with b1~b0=10 and \$A0H~\$A8H=01 000 110 |   |   |          |             |                 |                              |  |  |  |  |  |  |
|  | Operators (para\$20H~\$35H e\$A0H~\$A8H)  |   |   |   |          |             |                 |                              |  |  |  |  |  |  |
|  | Oper: 19 20 21 22 23 24 25 26 27  |   |   |   |          |             |                 |                              |  |  |  |  |  |  |
|  | Voz: 1 2 3 1 2 3 4 5 6  |   |   |   |          |             |                 |                              |  |  |  |  |  |  |
|  | Reg: \$20\$21\$22\$23\$24\$25\$28\$29\$2A   |   |   |   |          |             |                 |                              |  |  |  |  |  |  |
|  | Freq: \$A0\$A1\$A2\$A0\$A1\$A2\$A3\$A4\$A5  |   |   |   |          |             |                 |                              |  |  |  |  |  |  |
|  | Oper: 28 29 30 31 32 33 34 35 36  |   |   |   |          |             |                 |                              |  |  |  |  |  |  |
|  | Voz: 4 5 6 7 8 9 7 8 9  |   |   |   |          |             |                 |                              |  |  |  |  |  |  |
|  | Reg: \$2B\$2C\$2D\$30\$31\$32\$33\$34\$35   |   |   |   |          |             |                 |                              |  |  |  |  |  |  |
|  | Freq: \$A3\$A4\$A5\$A6\$A7\$A8\$A6\$A7\$A8  |   |   |   |          |             |                 |                              |  |  |  |  |  |  |
| <div>The operators are associated as below:<br/>\$20/\$40/\$60/\$80/\$A0/\$B0/\$C0<br/>or<br/>\$23/\$43/\$63/\$83/\$A0/\$B0/\$C0</div> |   |   |   |   |          |             |                 |                              |  |  |  |  |  |  |
|  |   |   |   |   |          |             |                 |                              |  |  |  |  |  |  |
|  |   |   |   |   |          |             |                 |                              |  |  |  |  |  |  |
|  |   |   |   |   |          |             |                 |                              |  |  |  |  |  |  |
|  |   |   |   |   |          |             |                 |                              |  |  |  |  |  |  |
| \$C0H<br>⋮<br>\$C8H  | •   | •   | •   | • | •        | Feedback    | CON             | Feedback factor/connect type |  |  |  |  |  |  |
|  | b7~b4   |   | Not used (always“0000”)                               |   |          |             |                 |                              |  |  |  |  |  |  |
|  | b3~b1   |   | Feedback factor (0=0; 1=π/16; 2=π/8; ...; 6=2π; 7=4π) |   |          |             |                 |                              |  |  |  |  |  |  |
|  | b0  |   | Connection type (0=serie; 1=paralell)                 |   |          |             |                 |                              |  |  |  |  |  |  |
|  | For 4 operators:  |   |   |   |          |             |                 |                              |  |  |  |  |  |  |
|  | A1  | Canal   | CNT(n)  |   | CNT(n+3) |             |                 |                              |  |  |  |  |  |  |
|  | 0   | 1   | C0H   |   | C3H      |             |                 |                              |  |  |  |  |  |  |
|  | 0   | 2   | C1H   |   | C4H      |             |                 |                              |  |  |  |  |  |  |
|  | 0   | 3   | C2H   |   | C5H      |             |                 |                              |  |  |  |  |  |  |
|  | 1   | 4   | C0H   |   | C3H      |             |                 |                              |  |  |  |  |  |  |
|  | 1   | 5   | C1H   |   | C4H      |             |                 |                              |  |  |  |  |  |  |
|  | 1   | 6   | C2H   |   | C5H      |             |                 |                              |  |  |  |  |  |  |
|  | CNT(n)=0 CNT(n+3)=0   |   |   |   |          |             |                 |                              |  |  |  |  |  |  |
|  |  |   |   |   |          |             |                 |                              |  |  |  |  |  |  |
|  | CNT(n)=0 CNT(n+3)=1   |   |   |   |          |             |                 |                              |  |  |  |  |  |  |
|  |  |   |   |   |          |             |                 |                              |  |  |  |  |  |  |
|  | CNT(n)=1 CNT(n+3)=0   |   |   |   |          |             |                 |                              |  |  |  |  |  |  |
|  |  |   |   |   |          |             |                 |                              |  |  |  |  |  |  |
|  | CNT(n)=1 CNT(n+3)=1   |   |   |   |          |             |                 |                              |  |  |  |  |  |  |
|  |  |   |   |   |          |             |                 |                              |  |  |  |  |  |  |
| \$E0H<br>⋮<br>\$F5H  | •   | •   | •   | • | •        | Wave Select | Waveform select |                              |  |  |  |  |  |  |
|  | b7~b3   |   | Not used (always“00 000”)                             |   |          |             |                 |                              |  |  |  |  |  |  |

|  |       |   |
|--|-------|---|
|  | b2~b1 | <p>Waveform select</p> <p>000 -  011 -  110 - </p> <p>001 -  100 -  111 - </p> <p>010 -  101 - </p> |
|--|-------|---|

### 11.6.3 – Wave synthesis

| Wave synthesis      |                                  |   |             |     |     |     |                     |   |  |
|---------------------|----------------------------------|---|-------------|-----|-----|-----|---------------------|---|--|
| Reg                 | b7                               | b6  | b5          | b4  | b3  | b2  | b1                  | b0  | Short Description  |
| \$00H<br>\$01H      | Test                             |   |             |     |     |     |                     |   | Test registers   |
| \$02H               | ID disp                          |   | Wave header |     |     | MT  | MM                  | Special functions   |  |
|                     | b7~b5<br>b4~b2                   | OPL4-ID (b7=0; b6=0; b7=1)<br>Wave table header:<br>000=0 to 511 (000 000H)    100=384 to 511 (200 000H)<br>001=384 to 511 (080 000H)    101=384 to 511 (280 000H)<br>010=384 to 511 (100 000H)    110=384 to 511 (300 000H)<br>011=384 to 511 (180 000H)    111=384 to 511 (380 000H)<br>b1    Audio memory type (0=ROM; 1=RAM)<br>b0    Audio memnry access (0=OPL4; 1=CPU) |             |     |     |     |                     |   |  |
| \$03H               | •                                | •   | a21         | a20 | a19 | a18 | a17                 | a16   | Audio memory adress  |
| \$04H               | a15                              | a14   | a13         | a12 | a11 | a10 | a9                  | a8  |  |
| \$05H               | a7                               | a6  | a5          | a4  | a3  | a2  | a1                  | a0  |  |
| \$06H               | Memory data                      |   |             |     |     |     |                     |   | Data register  |
| \$08H<br>⋮<br>\$1FH | Wave table number<br>LSB (n7~n0) |   |             |     |     |     |                     |   | 24 registers with the<br>LSB number (n7~n0) of<br>the wave table |
| \$20H<br>⋮<br>\$37H | F_number (f6~f0)                 |   |             |     |     |     | Tab<br>Wave<br>(n8) | 24 registers with the<br>7 bits LSB frequency and<br>MSB wave table number (n8) |  |

|                     |                               |   |                   |                     |   |   |
|---------------------|-------------------------------|---|-------------------|---------------------|---|---|
| \$38H<br>⋮<br>\$4FH | Octave<br>(o3~o0)             |   | Pseudo-rev        | F_number<br>(f9~f7) | Octave (-7 a +7)<br>Pseudo-reverberation<br>Frequency (3 bits MSB)                                    |   |
|                     | b7~b0<br>b7~b6<br>b3<br>b7~b4 | With “b0” (n8) select up to 512 samples (0~511)<br>With “b2-b1-b0” (f9~f7) defines the frequency<br>If “1” turn pn the pseudo-reverberation; if “0”, turn off<br>Octave. Range from -7 to +7 (-8 can’t used). With<br>F_number defines the frequency. For octave = 1 and<br>F_number = 0, the frequency is 44,1 Khz.<br>$f(\text{Hz}) = 1200 * (\text{octave} - 1) + 1200 * \log_2 \frac{1024 + F\_number}{1024}$ |                   |                     |   |   |
| \$50H<br>⋮<br>\$67H | Total level (l6~l0)           |   |                   | DL                  | Total level 7 bits (l6~l0)<br>Direct level  |   |
|                     | b7-b1<br>b0                   | Total level (b7=-24dB, b6=-12dB, ... b1=-0,375dB)<br>Direct level (0=change envelope during interpolation;<br>1=change envelope immediately)  |                   |                     |   |   |
| \$68H<br>⋮<br>\$7FH | Key on                        | Damp  | LFO RST           | CH                  | Panpot  | miscellaneous functions and<br>stereo balancing<br>(Panpot) |
|                     | b7<br>b6<br>b5<br>b4<br>b3~b0 | 0=key on; 1=key off<br>0=Damp off; 1=Damp on<br>LFO RST (0=turn on the LFO; 1=turn off the LFO)<br>0=Wave mixed with FM; 1=No mixing<br>Panpot:    0    1    2    ...    6    7    8    9    ...    13    14    15<br>Left (dB)   0   -3   -6    ...   -18   -∞   -∞   0    ...   0    0    0<br>Right (dB)   0    0    0    ...    0    0    -∞   -18   ...   -9   -6   -3                                       |                   |                     |   |   |
| \$80H<br>⋮<br>\$97H | •                             | •   | LFO<br>(s2~s0)    | VIB<br>(v2~v0)      | Tremolo and vibrato<br>frequency (LFO)<br>Vibrato level (VIB)   |   |
|                     | b7~b6<br>b5~b3<br>b2~b0       | Not used<br>LFO (0=0,168Hz, 1=2,019Hz, ... 7=7,066Hz)<br>Vibrato level (0=off, 1=3,378; 2=5,065, ... 7=79,31)   |                   |                     |   |   |
| \$98H<br>⋮<br>\$AFH | Attack<br>Rate                |   | Decay<br>Rate (1) |                     | Attack Rate 10-90% →<br>(1=3715 mS; 14=0,23 mS)<br>Decay 1 Rate 10-90% →<br>(1=19 040 mS; 14=1,18 mS) |   |



|     |       |  |                            |
|-----|-------|--|----------------------------|
|     | b5    | FT2 – Will be “1” when timer 2 end counting  |                            |
|     | b4~b2 | Not used (always“000”)   |                            |
|     | b1    | LD – Will be “1” during PCM header reading by the OPL4<br>(Valid when 05H_NEW2 bit of the array 1 is “1”.) |                            |
|     | b0    | BUSY – Will be “1” during registers data writing<br>(Valid when 05H_NEW2 bit of the array 1 is “1”.)       |                            |
| C5H | W     | Data byte  | Write data in the register |
| C4H | W     | Register number (00H a F5H)  | Select FM reg array 0      |
| C7H | R     | Mirror of C5H  | C5H access is preferred    |
| 7EH | W     | Register number (00H a F9H)  | Select PCM registers       |
| 7FH | W/R   | Data byte  | Read/write data in regs    |

### 11.6.5 – Wave table synthesis header

| End | b7  | b6  | b5  | b4  | b3  | b2  | b1  | b0  |   |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|---|
| 00H | d1  | d0  | s21 | s20 | s19 | s18 | s17 | s16 | d1, d0 → 00=8bits;<br>01=12 bits; 10=16 bits<br>s21~s0 = start adress |
| 01H | s15 | s14 | s13 | s12 | s11 | s10 | s9  | s8  |   |
| 02H | s7  | s6  | s5  | s4  | s3  | s2  | s1  | s0  |   |
| 03H | l15 | l14 | l13 | l12 | l11 | l10 | l9  | l8  | Loop adress   |
| 04H | l7  | l6  | l5  | l4  | l3  | l2  | l1  | l0  |   |
| 05H | e15 | e14 | e13 | e12 | e11 | e10 | e9  | e8  | End adress  |
| 06H | e7  | e6  | e5  | e4  | e3  | e2  | e1  | e0  |   |
| 07H | •   | •   | f2  | f1  | f0  | v2  | v1  | v0  | LFO freq. and vibrato level   |
| 08H | ar3 | ar2 | ar1 | ar0 | dr3 | dr2 | dr1 | dr0 | Attack Rate; Decay 1 Rate   |
| 09H | dl3 | dl2 | dl1 | dl0 | dr3 | dr2 | dr1 | dr0 | Decay Level; Decay 2 Rate   |
| 0AH | rc3 | rc2 | rc1 | rc0 | rr3 | rr2 | rr1 | rr0 | Rate correct; Release Rate  |
| 0BH | •   | •   | •   | •   | •   | am2 | am1 | am0 | AM level (tremolo)  |

11.6.6 – Wave data lenght

|         |     |     |     |     |     |     |    |    |      |
|---------|-----|-----|-----|-----|-----|-----|----|----|------|
| 16 bits | d15 | d14 | d13 | d12 | d11 | d10 | d9 | d8 | +00H |
|         | d7  | d6  | d5  | d4  | d3  | d2  | d1 | d0 | +01H |
| 12 bits | d11 | d10 | d9  | d8  | d7  | d6  | d5 | d4 | +00H |
|         | d3  | d2  | d1  | d0  | d3  | d2  | d1 | d0 | +01H |
|         | d11 | d10 | d9  | d8  | d7  | d6  | d5 | d4 | +02H |
| 8 bits  | d7  | d6  | d5  | d4  | d3  | d2  | d1 | d0 | +00H |



## 11.7 – MAP OF THE SCC REGISTERS (2212/2312)

| Adresses    | Short Description (SCC)  |    |    |     |     |    |    |    |    |   |   |   |   |     |     |    |    |
|-------------|--|----|----|-----|-----|----|----|----|----|---|---|---|---|-----|-----|----|----|
| 9800H~981FH | Waveform of the voice #1   |    |    |     |     |    |    |    |    |   |   |   |   |     |     |    |    |
| 9820H~983FH | Waveform of the voice #2   |    |    |     |     |    |    |    |    |   |   |   |   |     |     |    |    |
| 9840H~985FH | Waveform of the voice #3   |    |    |     |     |    |    |    |    |   |   |   |   |     |     |    |    |
| 9860H~987FH | SCC : Write/read: Waveform of the voices #4 e #5<br>SCC+: Read: Waveform of the voice #4   |    |    |     |     |    |    |    |    |   |   |   |   |     |     |    |    |
| 9880H~9881H | Frequency of the voice #1  |    |    |     |     |    |    |    |    |   |   |   |   |     |     |    |    |
| 9882H~9883H | Frequency of the voice #2  |    |    |     |     |    |    |    |    |   |   |   |   |     |     |    |    |
| 9884H~9885H | Frequency of the voice #3  |    |    |     |     |    |    |    |    |   |   |   |   |     |     |    |    |
| 9886H~9887H | Frequency of the voice #4  |    |    |     |     |    |    |    |    |   |   |   |   |     |     |    |    |
| 9888H~9889H | Frequency of the voice #5  |    |    |     |     |    |    |    |    |   |   |   |   |     |     |    |    |
|             | Example:<br>9880= <table><tr><td>f7</td><td>f6</td><td>f5</td><td>f4</td><td>f3</td><td>f2</td><td>f1</td><td>f0</td></tr></table> 9881= <table><tr><td>•</td><td>•</td><td>•</td><td>•</td><td>f11</td><td>f10</td><td>f9</td><td>f8</td></tr></table><br>$F_{\text{tone}} = \frac{F_{\text{clock}}}{32 * ((f_{11} \sim f_0) + 1)} \text{ (F\_clock=3,579545 MHz)}$ | f7 | f6 | f5  | f4  | f3 | f2 | f1 | f0 | • | • | • | • | f11 | f10 | f9 | f8 |
| f7          | f6   | f5 | f4 | f3  | f2  | f1 | f0 |    |    |   |   |   |   |     |     |    |    |
| •           | •  | •  | •  | f11 | f10 | f9 | f8 |    |    |   |   |   |   |     |     |    |    |
| 988AH       | Volume of the voice #1 (0 to 15)   |    |    |     |     |    |    |    |    |   |   |   |   |     |     |    |    |
| 988BH       | Volume of the voice #2 (0 to 15)   |    |    |     |     |    |    |    |    |   |   |   |   |     |     |    |    |
| 988CH       | Volume of the voice #3 (0 to 15)   |    |    |     |     |    |    |    |    |   |   |   |   |     |     |    |    |
| 988DH       | Volume of the voice #4 (0 to 15)   |    |    |     |     |    |    |    |    |   |   |   |   |     |     |    |    |
| 988EH       | Volume of the voice #5 (0 to 15)   |    |    |     |     |    |    |    |    |   |   |   |   |     |     |    |    |
| 998FH       | <table><tr><td>•</td><td>•</td><td>•</td><td>v5</td><td>v4</td><td>v3</td><td>v2</td><td>v1</td></tr></table> v5=1 → turn on voice #5<br>v4=1 → turn on voice #4, etc  | •  | •  | •   | v5  | v4 | v3 | v2 | v1 |   |   |   |   |     |     |    |    |
| •           | •  | •  | v5 | v4  | v3  | v2 | v1 |    |    |   |   |   |   |     |     |    |    |
| 9890H~989FH | Mirror of 9880H~988FH  |    |    |     |     |    |    |    |    |   |   |   |   |     |     |    |    |
| 98A0H       | SCC: no function<br>SCC+: waveform read of the voice #5 (no write allowed)   |    |    |     |     |    |    |    |    |   |   |   |   |     |     |    |    |
| 98A1H~98BFH | Mirrors of 98A0H   |    |    |     |     |    |    |    |    |   |   |   |   |     |     |    |    |
| 98C0H       | SCC: Mirror of 98A0H<br>SCC+: Deformation register   |    |    |     |     |    |    |    |    |   |   |   |   |     |     |    |    |

|             |   |
|-------------|---|
| 98C1H~98DFH | SCC: Mirror of 98A0H<br>SCC+: Mirror of 98C0H   |
| 98E0H       | SCC: Deformation register<br><div style="border: 1px solid black; padding: 2px; display: inline-block;"> R R . . . P P </div><br>PP: 11/10→Ftone*16; 01→Ftone*256; 00→ Ftone*1<br>RR: 11 → white noise voices 4 and 5 cfe waveform<br>01 → continuous white noise<br>00 → no white noise<br>SCC+: No function |
| 98E1H~98FFH | Mirrors of 98E0H  |

### 11.7.1 – Access addresses for SCC

| Endereços   | Short Description (SCC+)   |
|-------------|--|
| B800H~B81FH | Voice #1 waveform  |
| B820H~B83FH | Voice #2 waveform  |
| B840H~B85FH | Voice #3 waveform  |
| B860H~B87FH | Voice #4 waveform  |
| B880H~B89FH | Voice #5 waveform  |
| B8A0H~B8A1H | Voice #1 frequency   |
| B8A2H~B8A3H | Voice #2 frequency   |
| B8A4H~B8A5H | Voice #3 frequency   |
| B8A6H~B8A7H | Voice #4 frequency   |
| B8A8H~B8A9H | Voice #5 frequency   |
|             | Example:<br>B8A0= <div style="border: 1px solid black; padding: 2px; display: inline-block;">f7f6f5f4f3f2f1f0</div> B8A1= <div style="border: 1px solid black; padding: 2px; display: inline-block;">. . . . f11f10f9f8</div><br>$F_{tone} = \frac{F_{clock}}{32 * ((f_{11} - f_0) + 1)}$ (F_clock=3,579545 MHz) |
| B8AAH       | Voice #1 volume (0 to 15)  |
| B8ABH       | Voice #2 volume (0 to 15)  |
| B8ACH       | Voice #3 volume (0 to 15)  |
| B8ADH       | Voice #4 volume (0 to 15)  |

|             |  |
|-------------|--|
| B8AEH       | Voice #5 volume (0 to 15)  |
| B8AFH       | <div> <div> <div>•</div> <div>•</div> <div>•</div> <div>v5</div> <div>v4</div> <div>v3</div> <div>v2</div> <div>v1</div> </div> v5=1 → turn on voice #5<br/> v4=1 → turn on voice #4, etc </div>   |
| B8B0H~B8BFH | Mirror of B8A0H~B8AFH  |
| B8C0H       | <div> <div> <div>R</div> <div>R</div> <div>•</div> <div>•</div> <div>•</div> <div>•</div> <div>P</div> <div>P</div> </div> – Deformation register<br/> PP: 11/10→Ftone*16; 01→Ftone*256; 00→ Ftone*1<br/> RR: 11 → white noise voices 4 and 5 cfe waveform<br/> 01 → continuous white noise<br/> 00 → no white noise </div>  |
| B8C1H~B8DFH | Mirrors of B8C0H   |
| B8E0H~B8FFH | No function  |
| B900H~BFFDH | ???  |
| BFFEH~BFFFH | <div> <div> <div>•</div> <div>•</div> <div>S</div> <div>M</div> <div>•</div> <div>B3</div> <div>B2</div> <div>B1</div> </div> – Mode register<br/> S – SCC mode (0=SCC; 1=SCC+)<br/> M – Memory mode (0=bank select; 1=RAM)<br/> B3 – Memory bank #3 (0=bank select; 1=RAM)<br/> B2 – Memory bank #2 (0=bank select; 1=RAM)<br/> B1 – Memory bank #1 (0=bank select; 1=RAM) </div> |

## BIBLIOGRAPHIC REFERENCES

APROFUNDANDO-SE NO MSX

Piazzzi – Maldonado – Oliveira (Editora Aleph, 1986)

CARTÃO DE 80 COLUNAS & RS232C – MANUAL DE OPERAÇÕES

Gradiente (1989)

FM MUSIC MACRO YRM-104 – Owner’s Manual

Yamaha (1984)

GR8NET Technical Databook and Programmer’s Guide

Age Labs (2019)

HBI-232MKII – Basic Manual

Age Labs & Ebsoft (2014)

LIVRO VERMELHO DO MSX, O (The Red Book)

McGraw Hill /Avalon Software (1988 / 1985)

MANUAL DO MICROPROCESSADOR Z-80

William Barden Jr. (Editora Campus, 1985)

MIDI MACRO MONITOR YRM-303 – Owner’s Manual

Yamaha (1986)

MSX DATAPACK volumes 1, 2 e 3

ASCII Corporation (1991)

MSX-DOS version 2 – The advanced disk operating system for MSX 2 computers – ASCII Corp (1988)

MSX MAGAZINE, Edição Dezembro de 1990

ASCII Corporation (1990)

MSX MAGAZINE, Edição ???

ASCII Corporation (1990)

MSX MOZAIK, Edição nº 33  
 Editora desconhecida, Ano desconhecido

MSX TECHNICAL GUIDE BOOK  
 Ayumu Kimura (ASCAT Ashigaka, NIPPON, 1992)

MSX TECHNICAL DATA BOOK  
 Sony Corp (1984)

MSX turbo R TECHNICAL HANDBOOK  
 ASCII Corporation (1991)

MSX2 TECHNICAL HANDBOOK  
 ASCII Corporation (1985)

NEXTOR 2.0 User Manual  
 Konamiman (2014)

OPL4 YMF278B – APPLICATION MANUAL  
 Yamaha Corporation (1994)

PROGRAMAÇÃO AVANÇADA EM MSX  
 Figueredo – Maldonado – Rosseto (Editora Aleph, 1986)

PX-7 P-BASIC Reference Manual  
 Pioneer (1985)

TMS9918A/TMS9928A/TMS9929A Video Display Processors Data  
 Manual – Texas Instruments (1982)

V9938 MSX-VIDEO – APPLICATION MANUAL  
 Nippon Gakki Co. Ltd. (Yamaha, 1985)

V9938 MSX-VIDEO – TECHNICAL DATA BOOK  
 Nippon Gakki Co. Ltd. (Yamaha, 1985)

V9958 MSX-VIDEO – TECHNICAL DATA BOOK  
 Yamaha Corporation (1989)

V9990 E-VDP-III – APPLICATION MANUAL

Yamaha Corporation (1992)

Y9850 MSX-AUDIO – APPLICATION MANUAL

Nippon Gakki Co. Ltd. (Yamaha, 1985)

YM2413 FM OPERATOR TYPE LL (OPLL) – APPLICATION MANUAL

Yamaha Corporation (1987)

<https://www.gigamix.jp/ds2/>

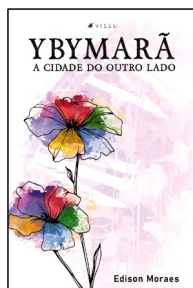
[https://www.msx.org/wiki/I/](https://www.msx.org/wiki/I/O_Ports_List#The_register_of_internal_I2FO_ports_control)

[O\\_Ports\\_List#The\\_register\\_of\\_internal\\_I2FO\\_ports\\_control](https://www.msx.org/wiki/I/O_Ports_List#The_register_of_internal_I2FO_ports_control)

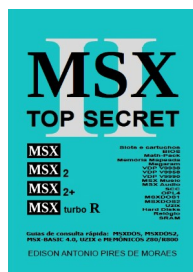
<http://msxbanzai.tni.nl/v9990/manual.html>

[https://github.com/Konamiman/MSX-UNAPI-specification/tree/](https://github.com/Konamiman/MSX-UNAPI-specification/tree/master/docs)  
[master/docs](https://github.com/Konamiman/MSX-UNAPI-specification/tree/master/docs)

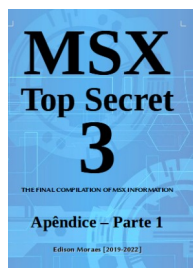
## OTHER BOOKS BY THE AUTHOR



YBYMARÃ – A Cidade do Outro Lado



MSX Top Secret 2



MSX Top Secret 3 – Appendix I

MSX Top Secret 3 – Appendix II



MSX Top Secret



MSX Top Secret 3

