



NST'S MONITOR EXTENSION BY BSZ  
FW V7.02 (A2/0123) B356 - 140119  
WARNING: ADDICTIVE!

## NST's Monitor Extension V7.02

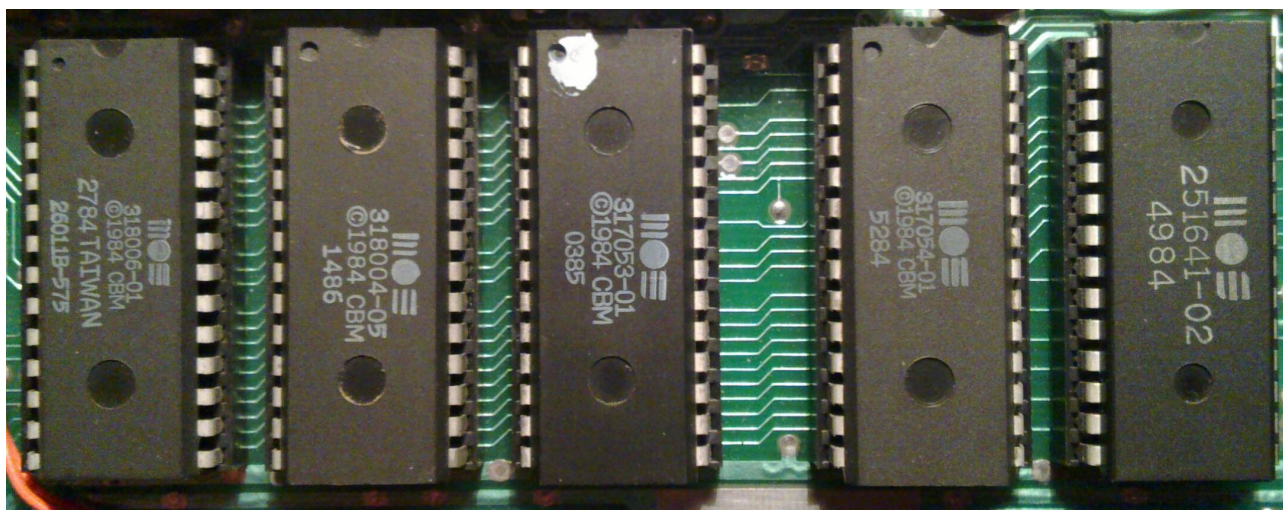
### Overview:

- Expands the features of the built-in **TEDMON**
- Handles "illegal codes"
- Scrolls screen in both directions
- Disk drive memory management
- Disk block management
- Clear Memory Function
- Etc.

### Usage:

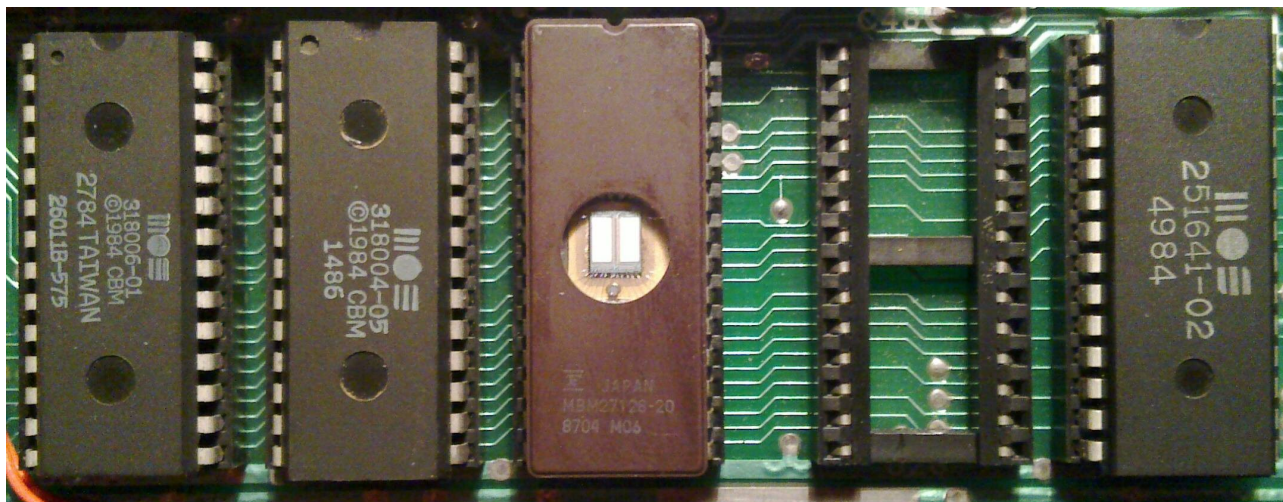
The expansion (**BMX** from now on) can be placed into lower half of any ROM-BANK, the simplest solution is probably replacing the built-in extra software (which was already outdated even at the time of release). It is easiest to try it with an emulator, e.g. **YAPE** (<http://yape.homeserver.hu/>) or **plus4emu** (<http://plus4emu.sourceforge.net/>). In these, **BMX** ROM should be loaded to FUNCTION-LO, C1L or C2L, the other half (FUNCTION-HI, C1H, C2H) should be left blank.

On the original machine the simplest solution is to replace the **3-plus-1**, the correctly programmed (E)EPROM (e.g. 27(C)128) should be put in its place. The original line-up looks like this:



These ICs contain (from left to right) BASIC, KERNAL, FUNCTION-LO and FUNCTION-HI, they are ROMs. (The fifth IC (labeled **251641-02**) is the FPLA, ignore it for now.)

From the original ROMs the two that should be removed from the sockets are **U25** (labeled **317053-01**) and **U26** (labeled **317054-01**), and then the properly programmed (E)EPROM should be inserted to **U25** socket:



The **U26** is left blank (for now). The **BMX** is ready to go once the machine is assembled, there's nothing else to be done.

### Compatibility:

Compatibility is an important aspect of the system, in general, using the extension is not a problem. The only problem that usually arises is one that will almost always happen to anyone who never used this type of expansion before. This stems from a faulty programming habit that was – probably – spread by the "**Machine Language Programming for Beginners and Experts**" (which is otherwise a very informative and appreciated) book:

Önálló megszakító rutin írásának egyszerűbb formája az, amikor a \$FCB3–\$FCC8 és a \$CE00–\$CE0B ROM-programokat még használjuk. Mindezek feltétlenül működnek, amíg a \$FFFE–\$FFFF címek tartalma \$FCB3. Ennél a módszernél is a \$0314-es indirekt címet kell átírni úgy, hogy a saját rutinunkra mutasson. Ennek azonban JMP \$FCBE-vel vagy JMP \$FCC3-mal kell végződnie. (Mivel ritkán írunk új megszakító rutint a bővítő ROM-ok programjaihoz, gyakorlatilag a JMP \$FCC3 is mindig alkalmazható.)

89

"(Since we rarely write new IRQ routines in expansion ROMs, in practice JMP \$FCC3 can always be used.)"

This part in parentheses is the source of the problem. If the interrupt routine ends at \$FCC3, then it will fail to restore the ROM-BANKs that were in used at the point of interrupt. As BMX is in another ROM BANK, when the interrupt routine does not switch back to it, the execution will continue in the BASIC ROM, which will result in a crash. If you need to extend the ROM interrupt, then you MUST complete the routine by jumping to \$FCBE instead of \$FCC3.

In terms of memory usage, the extension attempts to be "minimalistic"; the only memory areas that were used by the original **TEDMON** are used. Exceptions: a few routines that are required by the extension are copied to \$0123 (for paging, etc.). Overwriting these routines will cause the extension to crash, if this memory area is needed, **BMX** can be turned off. By default, to enter **TEDMON** commands, a buffer is used, this can be found between \$0200..\$02FF. **BMX** only uses the area between \$0200..\$0257 (just like the BASIC interpreter), this means an 80-character long space, this is sufficient even for the most complex MONITOR commands. This area is also used as a memory buffer during operations, so modifying/overwriting it directly is not recommended.

## Features:

During power-on or reset some keys can be held down to control certain features:

- Pressing **Control + Space + Clear/Home** buttons at the same time will clear the memory
- Pressing **2** will switch **BMX** off, and it will stay off
- Pressing **1** will switch **BMX** back on, and it will stay on
- Pressing **Run/Stop** will start the monitor (just like the default)

The switched on state of **BMX** is indicated by the change border and the background, which will be set to lower luminance level. The "off" state is stored in the memory at \$0123 (which is the "top" of the stack), on that address \$44 ("D" as *Disabled*) is stored. If that value is found, then the **BMX** initialization routine returns as if nothing was done. In this case, it does not take over the implementation of RESET. (The paging routines use the same memory area. From a compatibility point of view, it would be preferable to have some hardware support, but that cannot be achieved with a simple ROM BANK exchange. In emulators, toggling ROMs is easier, but it's not convenient.)


The MONITOR commands extend the functionality of the original **TEDMON** commands, if **BMX** cannot process something, it will hand it over to the original ROM. In the documentation only the commands which are new or have expanded functionality are listed.

The command which list the contents of memory to the screen in some way will also work in "scroll" mode. If the cursor is at the bottom line of the screen (or at the bottom of the set window), then pressing cursor DOWN will list new lines. The same is true going upwards, at the top of the screen (or window), pressing the UP button will move the list backwards. If needed, this feature can be disabled.

The names of commands – unlike in the original **TEDMON** – can be more than one character long. The multi-character names can be shortened just like BASIC commands. (Thus pressing by the second or subsequent characters with SHIFT, the others may be omitted.)

All numeric parameters must be entered as a hexadecimal value. (Exceptions will be noted.) Parameters can be separated by space or comma, the same way as in **TEDMON**.

## The MONITOR's (extended) commands:

```
HELP
-----
 NST'S MONITOR EXTENSION BY BSZ
FW V7.02 (A2/0123) B356 - 140119
WARNING: ADDICTIVE!
-----
HELP      KILL      GRAPHIC    DIRECTORY
UNIT      TM          TDISK      FDISK
C         *R *W       O          GR,GZ G
D         A . ,   M >
E [       F          T          I
H         $ % #   LA L U S  C
-----
HELP [COMMAND] TO DETAILS.
```

HELP:

```
HELP: COMMANDS HELP
USAGE:
HELP = LIST ALL COMMANDS
HELP <COMMAND> = COMMAND HELP
```

Gives help. Without a parameter it displays the list of all available commands, with a command parameter it will list the help for the specified command. (Since this text takes up quite a lot of space, in the future the "explanatory" function may be left out.)

KILL:

```
KILL: DISABLE EXTENSIONS
USAGE:
KILL = DISABLE ALL EXTENSIONS
KILL M = DISABLE MONITOR EXTENSION
```

Turn off various functions of **BMX**. Without a parameter, all functions are turned off, and the "off" state is set. (They won't be restarted after reset, you will need to press correct key combination to turn it back on!). With the **M** parameter the monitor extensions are turned off, but these will be restored after RESET.

## GRAPHIC:

```
GRAPHIC: SWITCH GRAPHIC MODES
USAGE:
GRAPHIC = SWITCH BACK TO CHAR MODE
GRAPHIC M = SELECT GRAPHIC MODE M
GRAPHIC M BADR = BITMAP ADDR
GRAPHIC M BADR AADR = BITMAP+ATTR ADDR
```

The TED's with graphic mode can be switched on/off. Without a parameter it switches to the default character mode. The first parameter is the number of the graphics mode (same as in BASIC, 0, 1, 2, 3, 4), the second parameter is a bitmap address (0000, 2000, 4000, 6000, 8000, A000, C000, E000, or the short version of these, 00, 20, 40, 60, 80, A0, C0, E0 can also be used), the third parameter is the address of the color memory (this can be 0000, 0800, 1000, ... or the short version 00, 08, 10, ...).

## DIRECTORY:

```
DIRECTORY: PRINT DISK DIRECTORY
USAGE:
DIRECTORY = DIR UNIT8 DISK0
DIRECTORY "N*" = DIR UNIT8 DISK0
DIRECTORY UN = DIR UNITN DISK0
DIRECTORY UN DM = DIR UNITN DISKM
DIRECTORY "N*" UN DM = DIR UNITN DISKM
```

List the contents of the disk in the drive. Without a parameter it will use the already set (default 8) drive's unit / disk 0 to list. To use a different drive number, use the **Ux** form. (For example, drive 9 is **U9**, drive 10 is **UA** (!).) To get a listing of disk 1 (which of course is only useful for drives that are capable of handling two discs), the **D1** parameter must be specified. (The original BASIC DIRECTORY command does not handle the number of disk. It is syntactically evaluated, it's not sent to the drive, so this parameter doesn't matter. The DIRECTORY command in **BMX** handles this correctly.) This command can take a name parameter as well, if provided, this parameter must be the first one. The name may contain the "?" and/or "\*" characters, only the files matching the pattern will be listed. (Works the same way as the BASIC DIRECTORY command.)

## UNIT:

```
UNIT: CHANGE DRIVE UNIT
USAGE:
UNIT = PRINT CURRENT UNIT NO
UNIT N = SET DRIVE UNIT N
```

With this command, the default drive's unit number can be set. Without a parameter it shows the current unit number (default 8). If a parameter is passed in, it will set the unit number. After setting the drive number, all commands, which handles the drive in any way will use this drive unit number. (The unit number is a HEX value, so for drive 10 the parameter should be **A**!)

TM:

```
TM: TEDMON FALLBACK
USAGE:
TM COMMAND = EXECUTE COMMAND BY TEDMON
```

Call a **TEDMON** function. Whatever is after the "TM" command will be evaluated and executed by the built-in **TEDMON**. (If you need to some factory default function, this can be achieved with this command.)

TDISK:

```
TDISK: WRITE DATAS FROM RAM TO DRIVEMEM
USAGE:
TDISK SADR EADR DADR = WRITE DATAS (U8)
```

Move data from the computer's memory to the selected (default 8) drive's memory. The source is always the **plus/4**'s RAM, the destination is the selected drive's RAM (or other areas). In the case of the 1541-II, writing to \$8000..\$FFFF is not recommended! The first parameter is the start address, the second is the end address, this are will be copied. The third parameter is the starting address of the drive's memory, this is where the data will be written to.

FDISK:

```
FDISK: READ DATAS FROM DRIVEMEM TO RAM
USAGE:
FDISK SADR EADR DADR = READ DATAS (U8)
```

The opposite of the previous command, moves data from the drive's memory to the memory of the **plus/4**. The first parameter is the start address, the second is the end address, this are of the drive's memory will be copied. The third parameter is the start address in the **plus/4**'s memory, the data which is read from the drive will be written there.

@:

```
@: GET DRIVE STATUS / SEND COMMAND
USAGE:
@ = GET DRIVE STATUS (U8)
@ COMMAND = SEND COMMAND TO DRIVE (U8)
```

Get the status of the selected drive (default 8), or send a command. Without a parameter the drive's condition will be queries, when parameters are present, they are sent to the drive's command channel. (For example, "@N:DISK,XY" will start formatting the disk with name "DISK", ID "XY". "@I" will perform a disk initialization, etc.)

\*R:

```
*R: READ SECTOR FROM DISK (U8) TO RAM
USAGE:
*R TRACK SECTOR DADR = READ TR:SEC
```

Read a sector from the selected drive (default 8) to the **plus/4**'s memory. First parameter is the TRACK, second parameter is the SECTOR, and third is the machine's memory address. The memory address must always end in \$00, so **1234** cannot be used, in that case, \$1200.. \$12FF range will be used. (The memory address can be specified in "short" mode, specifying **12** will mean \$1200..\$12FF will be the target.) All parameters are of course HEX values! (For example, the disk's BAM sector can be read with the "**\*R 12 00 1200**" command, to the \$1200..\$12FF area. This is track 18, sector 0.)

\*W:

```
*W: WRITE SECTOR FROM RAM TO DISK (U8)
USAGE:
*W = WRITE PREVIOUSLY READED SECTOR
*W TRACK SECTOR SADR = WRITE TR:SEC
```

One sector of the selected drive (default 8) can be written with this command. Without any parameters, the sector which was previously read by the "**\*R**" command will be written to the same location from where it was read from. If there are parameters, they must be target TRACK and SECTOR, specified the same was as in the "**\*R**" command.

O:

```
O: CONFIGURE MONITOR
USAGE:
O = PRINT CURRENT SETUP
O <N> = SELECT ROM BANK <N>
O R = SELECT RAM
O Z = SELECT DRIVE (U8)
O I = EN/DIS ILLEGAL OPCODES
O S = EN/DIS SCROLL
```

The configuration of the MONITOR. The **TEDMON** knows only one such parameter, switching between the ROM/RAM. This is set by bit 7 of the byte at \$07F8. **BMX** also uses the same byte, but the other bits are in use as well, therefore it's manually changing that address is not recommended. (B7 still serves to switch the ROM/RAM, so the ">**07F8 80**" command works as expected. However, ">**07F8 FF**" will turn on other things and therefore it will probably cause unexpected results!) Without parameters, it displays the currently selected configuration. The parameter can be specified as follows:

- **0.F**: Enabled access to the specified ROM BANK. So not only the built in BASIC+KERNAL can be read, but also the expansion ROMs as well (including **BMX** itself)! Writing the ROM-BANK's number to \$FB (which is what the **TEDMON** used to read the other BANKs lower half) can no longer be done, because **BMX** does paging actively, this address is often overwritten.

- **R**: Reading the RAM can be turned on with this option. To return to reading the ROM, specify one of the BANKs from **0..F** (preferably **0**).
- **Z**: Access the drive's memory. From then on, ALL commands are accessing the DRIVE'S MEMORY! (To the user it appears as if the drive's memory is the machine's, as if the drive itself would be running the MONITOR program.) The exceptions are the "**G**" command (there's a separate command for starting a program in the drive's memory), and all other commands which are designed for moving data between the drive and the machine. All other commands however ("**A**", "**D**", "**M**", ">", ...) are working in the drive's memory. In the case of the 1541-II, the memory area between \$8000..\$FFFF should not be written! To switch back to the machine's memory use the **R** (RAM) or the **0..F** (ROM) parameters.
- **I**: Turn handling illegal codes on/off. When turned on, not only displaying them ("**D**" command), but also the inputting them ("**A**" command) is allowed!
- **S**: Turns the scroll function off/on.

G:

```
G : GO
  USAGE :
G = GO PROGRAM IN SELECTED ROM
G SADR = GO PROGRAM IN ROM FROM ADDR
```

Start program, same as the TEDMON "G" command. Without a parameter, it runs from the saved address, if a parameter is specified, it runs from that address. Addresses between \$8000..\$FFFF will run the ROM! Before starting, the saved register values will be set. (These may be viewed with the "R" TEDMON command).

GR:

```
GR : GO RAM
  USAGE :
GR = GO PROGRAM IN RAM
GR SADR = GO PROGRAM IN RAM FROM ADDR
```

Same as the "plain" "G" command, but before the start interrupts will be disabled, and RAM will be paged. RAM will be started even in the \$8000..\$FFFF range. Without a parameter it starts from the saved address, with a parameter it starts from the specified address. The registers will be set before start here as well!

GZ:

```
GZ : GO DRIVE (U8)
  USAGE :
GZ SADR = GO PROGRAM IN DRIVE FROM ADDR
```

Start a program in the drive's memory. (This, of course, will run on the drive's processor.) You must always enter an address for this command, and the registers will not be set, since the DOS does not allow this.



D:

```
D: DISASSEMBLE
  USAGE:
D = CONTINUE DISASSEMBLING
D SADR = DISASSEMBLE FROM ADDR
D SADR EADR = DISASS. FROM ADDR TO ADDR
```

Disassemble the memory; decode the memory and display the mnemonics and the associated parameters. The parameters can be empty, or a start and end address can be provided. If the illegal codes option is enabled, they are also shown. The list may be scrolled up/down with the cursor keys (just like other list), going down is straightforward. When going up, we have to move backwards in the memory, which is not always obvious. MISTAKES MAY OCCUR, but in most cases the function works well.

A/./,;

```
A: ASSEMBLE
.: ASSEMBLE, SEE [A] COMMAND
^: ASSEMBLE, SEE [A] COMMAND
A: ASSEMBLE
  USAGE:
A DADR OP OP OP MNE OPER = ASSEMBLE
```

Assemble, the entered mnemonics and parameters will be translated to machine code and stored in the memory. (All three commands perform the same function.) If the illegal codes option is enabled, those can also be entered. Unlike in the **TEDMON**, even the disassembled code column (produced with the "D" command) can be modified - with certain restrictions. (For example, changing the address of a parameter will update the address of the memory.)

M:

```
M: MEMORY DUMP
  USAGE:
M = CONTINUE MEMORY DUMP
M SADR = DUMP FROM ADDR
M SADR EADR = DUMP FROM ADDR TO ADDR
```

Displays the memory contents in HEX values and ASCII/PETSCII characters. Same as the **TEDMON** "M" command, except that bit 7 of the ASCII/PETSCII characters isn't cut off. The parameters can be nothing, or start and end addresses.

>:

```
>: EDIT MEMORY
  USAGE:
> DADR OP, ... = WRITE BYTE(S) TO ADDR
```

The memory contents can be edited as HEX values. The first parameter must be a memory address, followed by 0..8 bytes, which will be written into the memory. After that, the "ready" line (8 bytes) will be printed the same way as the the "M" command does. If only the address is given, nothing is written to the memory, but the eight bytes of data is displayed on the screen.

I:

```
I: CHARCODE DUMP
  USAGE:
I = CONTINUE CHARCODE DUMP
I $ADR = DUMP FROM ADDR
I $ADR EADR = DUMP FROM ADDR TO ADDR
```

The memory contents can be displayed as CHARACTER CODES. (So for example, if the memory contains \$01, the screen will display the letter "A". Not ASCII/PETSCII code, but the screen code!) 32 characters per line are displayed. The parameter can nothing, or start and end address.

':

```
': EDIT MEMORY IN CHARCODE MODE
  USAGE:
' DADR CHARCODES(MAX32) = EDIT MEM
```

Edit the data that was written to the screen by the "I" command. After the address, up to 32 characters can be entered, which will be stored in the memory as character codes. The address parameter is mandatory, after that 0..32 characters can be entered. After writing to the memory type (if there was data given), the 32 characters will be printed.

E:

```
E: BINARY DUMP
  USAGE:
E = CONTINUE BINARY DUMP
E $ADR = DUMP FROM ADDR
E $ADR EADR = DUMP FROM ADDR TO ADDR
```

Display the memory in "binary" mode. (For example, the "E D000" command will display the character set from the ROM.) Zero bits will be shown as ".", ones will be shown as "\*". Each line displays 8 bits of one byte. The parameter can be empty, or a start and end address can be given.

[:

```
[ : EDIT MEMORY IN BINARY MODE
  USAGE :
[ DADR 8BIT[.*] = EDIT MEM
```

Edit the binary data that was written to the screen by the "E" command. The address parameter is given, then it will display that memory location's contents. To modify the memory, all 8 bits must be specified. The bits must be "." and "\*", "0" and "1" are also acceptable. The final result will be printed with "."/\*".

F:

```
F : FILL MEMORY
  USAGE :
F SADR EADR OP      = WRITE PATTERN
F SADR EADR 'STRING' = WRITE STRING
F SADR EADR "STRING" = WRITE STRING
```

Fill the memory with data. The parameters should be the start and end addresses, as well as the data that will be written to the memory. Unlike **TEDMON**, it can take more than byte (up to 32), and "strings" are also accepted.

T:

```
T : MEMORY BLOCK TRANSFER
  USAGE :
T SADR EADR DADR = TRANSFER BYTES
```

Moves memory contents. Three parameters are needed: the start and end address, and the new destination address to which the memory contents to be moved. Unlike **TEDMON**, the data is always copied in the correct direction, therefore it works well even in cases of overlapping addresses.

C:

```
C : MEMORY BLOCK COMPARE
  USAGE :
C SADR EADR DADR = COMPARE BYTES
```

Comparison of memory contents. Three parameters are needed: start and end address, and the new destination address. The memory between the start and end address will be compared to the area start from the destination address, and the addresses of the differences will be printed. The comparison can be stopped by pressing **Run/Stop**.

H:

```
H: HUNT BYTE(S) OR STRING
  USAGE:
H SADR EADR OP = HUNT BYTES
H SADR EADR 'STRING' = HUNT STRING
H SADR EADR "STRING" = HUNT STRING
```

Search the memory for BYTES. Three (or more) parameters are expected: start and end address, between which the data will be searched, and at least one (max. 32) bytes of data, which you are looking for. Unlike TEDMON, "strings" in double quotes are also accepted. Pressing **Run/Stop** will stop the search.

\$:

```
$: HEX CONVERTER OR DIRECTORY
  USAGE:
$ = SEE DIRECTORY COMMAND
$ WORD = CONVERT HEX TO OTHER
```

Converting a HEX number to "all other formats". It expects a hexadecimal number as the parameter (up to 4 digits), which will be displayed in other forms. Without a parameter, it lists the contents of the currently selected disk drive, or if Ux parameter is given then the x drive's contents.

%:

```
?: BIN CONVERTER
  USAGE:
% BINWORD = CONVERT BIN TO OTHER
```

Convert a BINARY number to "all other formats". It expects a binary number as the parameter (up to 16 digits), which will be displayed in other forms.

#:

```
#: DEC CONVERTER
  USAGE:
# DECWORD = CONVERT DEC TO OTHER
```

Convert a decimal number to "all other formats" It expects a decimal number (0..65535), which will be displayed in other forms.

LA:

```
LA: PRINT LOAD ADDRESS & FILESIZE
USAGE:
LA "NAME" DN = PRINT FILE ON DN DATAS
```

Displays the file's start and end address. The first parameter should be the file's name in quotes, followed by the unit number. It will read the entire file, display the start address and the calculated end address. The file is NOT loaded into memory. Requires a disk drive!

L:

```
L: LOAD FILE
USAGE:
L = LOAD FROM TAPE
L "NAME" = LOAD "NAME" FROM TAPE
L "NAME" DN = LOAD "NAME" FROM DN
L "NAME" DN SADR = LOAD
L "NAME" DN SADR SEEK = LOAD
L "NAME" DN SADR SEEK BYNO = LOAD
```

Load the file into memory. Without a parameter, it loads the first file from device 1 (tape). The first parameter can be a file name in quotes, in this case it will attempt to load the file from the same place. The second parameter can be the device number, in this case it loads the given file from the given device. (E.g.: L "NAME",8 will load a file called "NAME" from the disk drive.) So far this works exactly like the **TEDMON**, the files will be loaded back to their originally saved address (same as the **BASIC LOAD "NAME",8,1** command). However, a third parameter can be a new memory address, the file will be loaded to that address. These functions will call the **KERNAL LOAD** routine, therefore if a "turbo loader" is used, it will work for this command as well. Optionally a fourth parameter can be give, if specified, that many bytes from the beginning of the file will be "skipped", and won't be loaded. Additional, an optional fifth parameter can be specified, this will determine the total number of bytes loaded. The last two parameters require a disk drive! The disk drive does not support the **SEEK** function, therefore the skipped data will be "loaded", but not stored. The number of skipped bytes should NOT include the file's load address (which are the actual two bytes in the file). The **KERNAL LOAD** routines do not support these functions, they were written by scratch. Therefore any optional "turbo loaders" will not work.

V:

```
V: VERIFY FILE
USAGE:
V = SEE L COMMAND
```

Verify the file (compare the file with the memory contents). The possible parameters are the same as the "L" commands, all functions described there will work the same. The only difference is, the file contents will not be loaded into the memory, it's only used for comparison. In case of any differences, "VERIFY ERROR" will be displayed.

S:

```
S: SAVE FILE
  USAGE:
S "NAME" DN SADR EADR = SAVE
S "NAME" DN SADR EADR DADR = SAVE
```

Save file. The first parameter should be the name in quotes, the memory contents will be saved with this name. The second parameter must be the device number, the file will be saved to the given device. The third and fourth parameters should be the start and end addresses, this is the memory area that will be saved. The byte on the end address will NOT BE SAVED! This function works the same as the **TEDMON**'s "S" command, and uses the KERNAL SAVE routine. Therefore any optional "turbo savers" will work as expected. An optional fifth parameter can be given, if provided, the file will be saved with this address as the load address. This function requires a disk drive! Since the KERNAL SAVE routine does not support changing the load address, this function was written from scratch. Consequently any used "turbo savers" will not work.

X:

```
X: EXIT TO BASIC
  USAGE:
X = EXIT
```

Exit monitor to the BASIC prompt. Same function as the **TEDMON** "X" command.

While working, the file handling commands ("LA", "L", "V", "S") may display some data related to the files or to the current operation. This data may include the following information:

- OS: Original start address
- SK: Number of skipped bytes
- MX: Maximum possible number of bytes to be loaded
- S: Start address, the file will be loaded from this address into the memory
- E: End address, the last byte into which data was loaded
- U: The first unused byte where no data was loaded (one byte higher than the end address, this was the end address during save!)
- C: Byte count, this many total bytes were loaded

The commands that run for a long time without printing anything onscreen ("TDISK", "FDISK", "F", "T", "C", "H", "LA", as well as the extra functions of "L", "V", "S") now have a simple visual feedback display. All of these commands may be interrupted by pressing the **Run/Stop** key.

**WARNING: USE AT YOUR OWN RISK! THERE ARE NO WARRANTIES OF ANY KIND!** (Any bug reports are welcome.)

**WARNING: EXTENDED USE MAY BE ADDICTIVE!**

©2013-2014, BSZ

Translated by Csabo / [www.plus4world.com](http://www.plus4world.com)

---

2013.11.26. First public version of the documentation  
2014.01.19. V7.02 - ROM-BANKs, scrolling, DIRECTORY, LA, L, V, S, "working feedback"