artisan basic

MSX-BASIC extension

Contents

[INTRODUCTION 5](#_Toc119871403)

[EXTENDED MEMORY SUPPORT 6](#_Toc119871404)

[BITMAP OPERATIONS 8](#_Toc119871405)

[ANIMATION SUPPORT 9](#_Toc119871406)

[New sprite control system 10](#_Toc119871407)

[Animation data memory handling 11](#_Toc119871408)

[BASIC program overall structure 11](#_Toc119871409)

[SOUND PLAYER 13](#_Toc119871410)

[HELPER FUNCTIONS 14](#_Toc119871411)

[BUILDING BINARIES 15](#_Toc119871412)

[ALPHABETICAL LIST OF COMMANDS 16](#_Toc119871413)

[ANIMCHAR 16](#_Toc119871414)

[ANIMDEF 17](#_Toc119871415)

[ANIMITEMPAT 18](#_Toc119871416)

[ANIMITEMPTR 19](#_Toc119871417)

[ANIMSPRITE 20](#_Toc119871418)

[ANIMSTART 21](#_Toc119871419)

[ANIMSTEP 23](#_Toc119871420)

[ANIMSTOP 24](#_Toc119871421)

[ARTINFO 25](#_Toc119871422)

[AUTOSGAMDEF 26](#_Toc119871423)

[AUTOSGAMSTART 29](#_Toc119871424)

[AUTOSGAMSTOP 30](#_Toc119871425)

[BLIT 30](#_Toc119871426)

[BOXMEMCPY 32](#_Toc119871427)

[BOXMEMVRM 34](#_Toc119871428)

[COLL 36](#_Toc119871429)

[FILRAM 38](#_Toc119871430)

[FILVRM 39](#_Toc119871431)

[GENCAL 40](#_Toc119871432)

[MAXANIMDEFS 41](#_Toc119871433)

[MAXANIMITEMS 42](#_Toc119871434)

[MAXANIMSPRS 43](#_Toc119871435)

[MAXAUTOSGAMS 44](#_Toc119871436)

[MEMCPY 45](#_Toc119871437)

[MEMVRM 46](#_Toc119871438)

[SGAM 47](#_Toc119871439)

[SNDPLYINI 49](#_Toc119871440)

[SNDPLYOFF 50](#_Toc119871441)

[SNDPLYON 50](#_Toc119871442)

[SNDSFX 51](#_Toc119871443)

[SPRDISABLE 52](#_Toc119871444)

[SPRENABLE 53](#_Toc119871445)

[SPRGRPMOV 55](#_Toc119871446)

[TILERAM 56](#_Toc119871447)

[TILEVRM 59](#_Toc119871448)

[VRMMEM 60](#_Toc119871449)

[INFORMATIONAL DATA 62](#_Toc119871450)

[APPENDIX A – HOW TO ACCESS FUNCTIONS VIA DEFUSR COMMAND 63](#_Toc119871451)

[APPENDIX B – HOW TO SAFELY LOAD EXTENSION 66](#_Toc119871452)

# INTRODUCTION

ARTISAN BASIC is an extension of MSX BASIC. It is targeted at MSX1 machines with 64Kb memory and a disk system.

The idea for development came after competing in MSX BASIC competition.

<https://www.msxblog.es/concurso-msx-basic-9o-edicion/>

I have always felt that the capabilities of the machine could have been better exploited under BASIC.

The main areas that ARTISAN extension is focusing on are:

* Extended memory support
* Bitmap operations
* Animation support
* Sound player
* Helper functions

Extension loads itself in page 1 at address #4000 and provides several new commands. Following sections will describe main functionality groups and details about each new command are given later. Refer to the table of contents.

|  |  |  |
| --- | --- | --- |
| Version | Date | Description |
| 0.8 |  | Initial version |
|  |  |  |
|  |  |  |

# EXTENDED MEMORY SUPPORT

Standard MSX BASIC allows access to 32Kb of memory. In 64Kb systems there is another 32Kb hidden beneath ROM in pages 0 and 1. ARTISAN basic allows memory to be copied to and from this upper 32Kb. Additionally copying to and from VRAM can come from and to this upper 32Kb. Commands from other sections that take memory buffers as parameters can read data from this area of memory. There are also a few commands that allow copying data from and to VRAM.

Commands included are:

* [BOXMEMCPY](#_BOXMEMCPY)
* [BOXMEMVRM](#_BOXMEMVRM)
* [FILRAM](#_FILRAM)
* [FILVRM](#_FILVRM)
* [MEMCPY](#_MEMCPY)
* [MEMVRM](#_MEMVRM)
* [VRMMEM](#_VRMMEM)

Since ARTISAN BASIC code also resides in this upper 32Kb, not all of it is free for use by programs. Memory map is given below:

Chart

Description automatically generated

ARTISAN BASIC does not occupy any memory below &H8000 allowing BASIC programs to have the same amount of free memory for code and variables as without the extension.

# BITMAP OPERATIONS

Several functions are provided to allow working with software sprites and tiling. Software sprites are defined by their data and mask that gets applied to background. Tiling functions allow placing data in a memory buffer or in video memory in a sequential fashion, when you want to apply one pattern over a larger area.

Commands included are:

* [BLIT](#_BLIT)
* [TILERAM](#_TILERAM)
* [TILEVRM](#_TILEVRM)

# ANIMATION SUPPORT

This section allows the creation of animation definitions that execute regularly based on VDP interrupt. Animation definitions allow changing of sprite pattern number, pattern data or changing character data.

To enable sprite animations, sprite handling has been revamped. Instead of PUT SPRITE commands one needs to define an array where sprite data is kept. This is transferred to VRAM on each interrupt.

Additionally grouping of sprites is supported which allow simultaneous moves and animation.

Commands in this section are grouped into several sections:

* Basic sprite handling system
  + [SPRDISABLE](#_SPRDISABLE)
  + [SPRENABLE](#_SPRENABLE)
* Group of sprites handling
  + [SPRGRPMOV](#_SPRGRPMOV)
* Animation definitions
  + [ANIMITEMPAT](#_ANIMITEMPAT)
  + [ANIMITEMPTR](#_ANIMITEMPTR)
  + [ANIMDEF](#_ANIMDEF)
  + [ANIMSPRITE](#_ANIMSPRITE)
  + [ANIMCHAR](#_ANIMCHAR)
  + [AUTOSGAMDEF](#_AUTOSGAMDEF)
* Animation control
  + [ANIMSTART](#_ANIMSTART)
  + [ANIMSTOP](#_ANIMSTOP)
  + [ANIMSTEP](#_ANIMSTEP)
  + [AUTOSGAMSTART](#_AUTOSGAMSTART)
  + [AUTOSGAMSTOP](#_AUTOSGAMSTOP)
  + [SGAM](#_SGAM)
* Animation memory buffers
  + [MAXANIMDEFS](#_MAXANIMDEFS)
  + [MAXANIMITEMS](#_MAXANIMitems)
  + [MAXANIMSPRS](#_MAXANIMSPRS)
  + [MAXAUTOSGAMS](#_MAXAUTOSGAMS)

## New sprite control system

The use of sprites is modified in ARTISAN basic in the following ways:

* Sprite attributes (location, pattern and color) are kept in an integer BASIC array of size (3,31)
* Values from the array are passed to VRAM during vertical blank if indicated by a specified integer variable
* Sprite control system is activated by SPRENABLE command
* When the system is active one should not run any commands that modify VRAM because of possible collision with sprite update
* When the system is active no new variables can be declared as this will cause corruption of the sprite control system

## Animation data memory handling

Defining animations requires some memory usage. This is located directly after the ARTISAN basic code in the segment &H4000-&h7FFF. That is why free memory in this segment depends on how many animations are defined. It is necessary to declare the maximum amount of each type of animation information before the use of definition commands. There are 4 types of definitions:

* Animation item – defines a single state
  + Sprite pattern, color and duration
  + Sprite/character pattern definition pointer and duration
* Animation definition – list of animation items to run
* Sprite/Character animation – link between which sprite/character to animate and with which animation definition
* Automatic Sprite Group Animation and Movement – automatic animation and movement between defined bounds of a sprite group

## BASIC program overall structure

The layout of the program that uses the sprite control system and animations is as follows:

* Declaration of all variables
* Declaration of sprite attributes array and the sprite update variable
  + SU%=0:DIM SA%(3,31)
* Reset of memory buffers for animations by defining zero size
* Resizing of memory buffers to required values
* Obtain free memory location in page 1 using MEMCPY(&H4010,VARPTR(A%),2) where A% was previously defined or using [ARTINFO](#_ARTINFO)
* SPRENABLE (SA%,SU%,0/1,32)
* ON ERROR GOTO definition
* ON STOP GOSUB definition
* Main program
* On end/error/stop run:
  + Stop animations
  + SPRDISABLE

# SOUND PLAYER

ARTISAN basic includes the AKG player from ARKOS tracker

<https://www.julien-nevo.com/arkostracker/>

in version 2.01

Sound data should be exported from the Arkos tracker in binary format. Memory location can be in the first two memory pages.

Commands included in this section are:

* [SNDPLYINI](#_SNDPLYINI)
* [SNDPLYOFF](#_SNDPLYOFF)
* [SNDPLYON](#_SNDPLYON)
* [SNDSFX](#_SNDSFX)

# HELPER FUNCTIONS

This includes various functions that do not belong in previous sections and provide various functionality.

Commands included here are:

* [ARTINFO](#_ARTINFO)
* [GENCAL](#_GENCAL)
* [COLL](#_COLL)

# BUILDING BINARIES

Code is segmented in several function groups which can be excluded or included. By default, all functions are included.

Flags controlling which sections are include can be found in file main.asm at the beginning.

Additionally, code allows access to functionality via basic CALL command and/or via DEFUSR command.

What part is included is also controlled by two flags in main.asm file. Unfortunately, both types of access would result in binary that is over 16Kb and thus unusable.

More information can be found in sections [INFORMATIONAL DATA](#_INFORMATIONAL_DATA) and [APPENDIX A](#_APPENDIX_A_–)

# ALPHABETICAL LIST OF COMMANDS

## ANIMCHAR

Defines single character animation sequence.

Format:

ANIMCHAR (byte ID, integer character\_number, byte animation\_definition\_id, byte cyclic\_flag)

Where ID is between 0 and MAXANIMSPRS-1 value

Character\_number specifies the character to animate (0-767)

Animation\_definition\_id is between 0 and MAXANIMDEFS-1

Cyclic\_flag of 0 means that the animation will run one time only, other values mean a looping animation

Prerequisites:

* MAXANIMSPRS reserved memory for definition
* Animation definition prepared with ANIMDEF

Errors:

* Invalid type if incorrect type passed
* Subscript out of bounds if parameters outside of allowed range

Example:

\_ANIMCHAR(0,255,0,1)

Sample code:

* ANIMTEST.BAS
* GAME.BAS

## ANIMDEF

Defines a list of animation items which is later associated with a character or a sprite.

Format:

ANIMDEF (byte ID, byte size, integer[] values)

Where ID is between 0 and MAXANIMDEFS-1 value

Size is number of animation items (1-15)

Values holds animation item IDs that form this animation definition

Prerequisites:

* MAXANIMDEFS reserved memory for definition
* Animation items prepared with ANIMITEMPTR/ANIMITEMPAT

Errors:

* Invalid type if incorrect type passed
* Subscript out of range if ID invalid
* Overflow if size outside 1-15 range
* Index out of bounds if values array smaller than size parameter

Example:

DIM V%(1):V%(0)=0:V%(1)=1

\_ANIMDEF(0,2,V%)

Sample code:

* ANIMTEST.BAS
* GAME.BAS

## ANIMITEMPAT

Defines a single animation state where sprite pattern and color are specified. Usable for sprites only.

Format:

ANIMITEMPAT (byte ID, integer ticks, byte pattern, byte color)

Where ID is between 0 and MAXANIMITEMS-1 value

Ticks is number of interrupts that this animation item lasts before stepping over to the next state as defined in animation definition (>0)

Pattern specifies sprite pattern to apply to a sprite

Color specifies the color to apply to a sprite

Prerequisites:

* MAXANIMITEMS reserved memory for definition

Errors:

* Subscript out of range if ID invalid
* Overflow if ticks=0

Example:

\_ANIMITEMPAT(0,4,5,6)

Sample code:

* ANIMTEST.BAS
* GAME.BAS

## ANIMITEMPTR

Defines a single animation state where pattern data is specified. Applicable to sprites and characters.

Format:

ANIMITEMPTR (byte ID, integer ticks, integer pointer)

Where ID is between 0 and MAXANIMITEMS-1 value

Ticks is number of interrupts that this animation item lasts before stepping over to the next state as defined in animation definition (>0)

Pointer is a memory location where pattern data is located, can be in pages 0 and 1.

Prerequisites:

* MAXANIMITEMS reserved memory for definition

Errors:

* Subscript out of range if ID invalid
* Overflow if ticks=0

Example:

\_ANIMITEMPTR(1,3,&H2000)

Sample code:

* ANIMTEST.BAS
* GAME.BAS

## ANIMSPRITE

Defines single sprite animation sequence.

Format:

ANIMSPRITE (byte ID, integer sprite\_number, byte animation\_definition\_id, byte cyclic\_flag)

Where ID is between 0 and MAXANIMSPRS-1 value

sprite\_number specifies the sprite to animate (0-31)

Animation\_definition\_id is between 0 and MAXANIMDEFS

Cyclic\_flag of 0 means that the animation will run one time only, other values mean a looping animation

Prerequisites:

* MAXANIMSPRS reserved memory for definition
* Animation definition prepared with ANIMDEF

Errors:

* Invalid type if incorrect type passed
* Subscript out of bounds if parameters outside of allowed range

Example:

\_ANIMSPRITE(0,5,0,1)

Sample code:

* ANIMTEST.BAS
* GAME.BAS

## ANIMSTART

Starts animation sequence.

Format:

ANIMSTART (byte ID)

or

ANIMSTART (byte item\_number, integer[] sprite\_animations)

Where ID is between 0 and MAXANIMSPRS-1 value

item\_number specifies the number of animations in the array

sprite\_animations array holds animation ids to start simultaneously

Prerequisites:

* Animation definition prepared with ANIMDEF

Errors:

* Invalid type if incorrect type passed
* Subscript out of bounds if parameters outside of allowed range

Example:

\_ANIMSTART(1)

Or

DIM A%(2):A%(0)=0:A%(1)=1:A%(2)=2

\_ANIMSTART(3,A%)

Sample code:

* ANIMTEST.BAS
* GAME.BAS

## ANIMSTEP

Manually progresses animation which is not started with ANIMSTART.

Format:

ANIMSTEP (byte ID)

or

ANIMSTEP (byte item\_number, integer[] sprite\_animations)

Where ID is between 0 and MAXANIMSPRS-1 value

item\_number specifies the number of animations in the array

sprite\_animations array holds animation ids to step simultaneously

Prerequisites:

* Animation definition prepared with ANIMDEF

Errors:

* Invalid type if incorrect type passed
* Subscript out of bounds if parameters outside of allowed range

Example:

\_ANIMSTEP(1)

Or

DIM A%(2):A%(0)=0:A%(1)=1:A%(2)=2

\_ANIMSTEP(3,A%)

Sample code:

* ANIMTEST.BAS
* GAME.BAS

## ANIMSTOP

Stops animation sequence.

Format:

ANIMSTOP (byte ID)

or

ANIMSTOP (byte item\_number, integer[] sprite\_animations)

Where ID is between 0 and MAXANIMSPRS-1 value

item\_number specifies the number of animations in the array

sprite\_animations array holds animation ids to stop simultaneously

Prerequisites:

* Animation definition prepared with ANIMDEF

Errors:

* Invalid type if incorrect type passed
* Subscript out of bounds if parameters outside of allowed range

Example:

\_ANIMSTOP(1)

Or

DIM A%(2):A%(0)=0:A%(1)=1:A%(2)=2

\_ANIMSTOP(3,A%)

Sample code:

* ANIMTEST.BAS
* GAME.BAS

## ARTINFO

Provides information about ARTISAN basic extension. This includes version, flags used to build it and free memory start position in page 1.

Note that this command is always available regardless of how ARTISAN basic was compiled and can be used to test if the extension is installed.

Format:

ARTINFO (int variable version, int variable flags, int variable free\_memory)

Where:

Version is a variable receiving version info in the form as described under [INFORMATIONAL DATA](#_INFORMATIONAL_DATA)

Flags variable holds build flags for the solution as described under [INFORMATIONAL DATA](#_INFORMATIONAL_DATA)

Free\_memory varaiable holds memory location in page 1 where free memory begins.

Prerequisites:

* None

Errors:

* Invalid type if incorrect type passed

Example:

V%=0:F%=0:M%=0

\_ARTINFO(V%,F%,M%)

Sample code:

* AUTOEXEC.BAS

## AUTOSGAMDEF

Defines automatic sprite group animation and movement between specified bounds.

Format:

AUTOSGAMDEF (byte ID, integer variable x, integer variable y, integer minimum, integer maximum, integer delta, integer direction, integer ticks, byte sprite\_group\_size, integer[2][] variable sprite\_group, byte item\_number, integer[] variable sprite\_animations\_negative\_direction, integer[] variable sprite\_animations\_positive\_direction )

Where ID is between 0 and MAXAUTOSGAMS-1 value

X is integer variable that holds horizontal sprite group location

Y is integer variable that holds vertical sprite group location

Minimum is the low range value of possible locations

Maximum is the high range value of possible locations

Delta is the step value for movement

Directions defines horizontal (=0) or vertical (!=0) direction

Ticks is the number of interrupts between sprite group movement and stepping through animations

Sprite\_group\_size defines number of sprites in a sprite group

Sprite\_group is an array describing a sprite group, for details refer to SPRGRPMOV command

Item\_number defines number of animations to step through

Sprite\_animations\_negative\_directions holds animations for when sprite group is going backwards

Sprite\_animations\_positive\_directions holds animations for when sprite group is going forward

Prerequisites:

* MAXAUTOSGAMS reserved memory for definition
* Animations prepared with ANIMSPRITE

Errors:

* Invalid type if incorrect type passed
* Subscript out of range if ID invalid

Example:

DIM AL%(2):AL%(0)=0:AL%(1)=1:AL%(2)=2

DIM AR%(2):AR%(0)=3:AR%(1)=4:AR%(3)=5

DIM SG%(2,1):SG%(0,0)=0:SG%(1,0)=0:SG%(2,0)=0

SG%(0,1)=1:SG%(1,1)=0:SG%(2,1)=0

X%=0:Y%=0

\_AUTOSGAMDEF(0,X%,Y%,0,100,1,0,1,2,SG%,3,AL%,AR%)

Sample code:

* ANIMTEST.BAS
* GAME.BAS

## AUTOSGAMSTART

Starts automatic sprite group movement and animation.

Format:

AUTOSGAMSTART (byte ID)

Where ID is between 0 and MAXAUTOSGAMS-1 value

Prerequisites:

* Animation definition prepared with AUTOSGAMDEF

Errors:

* Invalid type if incorrect type passed
* Subscript out of bounds if parameters outside of allowed range

Example:

\_AUTOSGAMSTART(1)

Sample code:

* ANIMTEST.BAS
* GAME.BAS

## AUTOSGAMSTOP

Stops automatic sprite group movement and animation.

Format:

AUTOSGAMSTOP (byte ID)

Where ID is between 0 and MAXAUTOSGAMS-1 value

Prerequisites:

* Animation definition prepared with AUTOSGAMDEF

Errors:

* Invalid type if incorrect type passed
* Subscript out of bounds if parameters outside of allowed range

Example:

\_AUTOSGAMSTOP(1)

Sample code:

* ANIMTEST.BAS
* GAME.BAS

## BLIT

Command implements software sprite functionality. It applies monochrome object of defined size onto defined memory background with 1 pixel precision. Object is defined with mask and data. Mask will be ANDed with background and then data will be ORed with background. All memory locations can be in pages 0 and 1.

Format:

BLIT (integer x, integer y, integer object\_data\_pointer, integer object\_mask\_pointer, integer width, integer height, integer background\_pointer, integer background\_width)

Where

X is location in the background (>=0)

Y is location in the background (>=0)

Object\_data\_pointer is a memory location where object foreground is defined

Object\_mask\_pointer is a memory location where object mask is defined

Width is object width in characters (8 pixels)

Height is object height in characters (8 pixels)

Background\_pointer is a memory location where background is located

Background\_width is background width in characters (8 pixels)

Prerequisites:

* None

Errors:

* Invalid type if incorrect type passed

Example:

\_BLIT(55,31,&h7000,&h7800,12,5,&h100,32,24)

Sample code:

* DEMO2.BAS
* FONT2.BAS
* GAME.BAS

## BOXMEMCPY

Copies window like data segment from one location into another. Locations can be in pages 0 and 1.

Diagram

Description automatically generated

Format:

BOXMEMCPY (integer P1, integer B3, integer number\_of\_rows, integer B1, integer P2, integer B2)

Where

P1 is memory location where source data begins

B3 is number of bytes in a single row of source data

Number\_of\_row is number of rows of source data

B1 is number of bytes of a source window row

P2 is memory location where to copy data

B2 is number of bytes of destination window row

Prerequisites:

* None

Errors:

* Invalid type if incorrect type passed

Example:

\_BOXMEMCPY(&H1000,80,256,5,&H7000,80)

Sample code:

* DEMO2.BAS

## BOXMEMVRM

Copies window like data segment from one location in RAM into another in VRAM. Source location can be in pages 0 and 1. Command parameters are the same as for BOXMEMCPY. B2 value should be 256 for SCREEN 2 mode.

Format:

BOXMEMVRM (integer P1, integer B3, integer number\_of\_rows, integer B1, integer P2, integer B2)

Where

P1 is memory location where source data begins

B3 is number of bytes in a single row of source data

Number\_of\_row is number of rows of source data

B1 is number of bytes of a source window row

P2 is memory location where to copy data

B2 is number of bytes of destination window row

Prerequisites:

* None

Errors:

* Invalid type if incorrect type passed

Example:

\_BOXMEMVRM(&H1000,80,256,5,BASE(12),256)

Sample code:

* DEMO2.BAS

## COLL

Collision detection between one rectangular object and a list of other rectangular objects.

Format:

COLL (integer variable result, integer x, integer y, integer width, integer height, integer list\_size, integer[7][] objects)

Where

Result is an integer variable where the result is stored, -1 if no collision, 0..list\_size-1 if collision

X is horizontal location of upper left edge

Y is vertical location of upper left edge

Width is the last column of an object, for a 16x16 sprite this is 15

Height is the last row of an object, for a 16x16 sprite this is 15

List\_size is the number of objects to check collision agains and stored in objects variable

Objects is a two dimensional array that describes collidable objects. These can either be static or sprites. For of a single array element is:

(0,n) – active flag, if 0 collision will not be checked

(1,n) – is horizontal location of upper left edge OR sprite ID depending on (7,n)

(2,n) – is horizontal location of upper left edge OR not used depending on (7,n)

(3,n) – horizontal offset where actual object begins, for example if a sprite pattern does not actually begin at (0,0)

(4,n) – vertical offset where actual object begins, for example if a sprite pattern does not actually begin at (0,0)

(5,n) – width or the last column of the object

(6,n) – height or the last row of the object

(7,n) – type, 0=generic, <>0 sprite

Prerequisites:

* None

Errors:

* Invalid type if incorrect type passed
* Subscript out of bounds if parameters outside of allowed range

Example:

(X% and Y% already defined)

R%=0:DIM O%(7,1)

O%(0,0)=1:O%(1,0)=100:O%(2,0)=80:O%(3,0)=0:O%(4,0)=0:O%(5,0)=9:O%(6,0)=9:O%(7,0)=0

O%(0,1)=1:O%(1,1)=31:O%(3,1)=4:O%(4,1)=4:O%(5,1)=5:O%(6,1)=5:O%(7,1)=1

\_COLL(R%,X%,Y%,15,15,2,o%)

Sample code:

* COLLTEST.BAS
* GAME.BAS

## FILRAM

Fills memory block with a specified value. Can be used for pages 0 and 1.

Format:

FILRAM (integer address, integer count, byte value)

Where

Address is the starting memory block location

Count is the number of bytes to write

Value is the number to fill the block with

Prerequisites:

* None

Errors:

* Invalid type if incorrect type passed

Example:

\_FILRAM (&h1000,1024,0)

Sample code:

* None

## FILVRM

Fills video memory block with a specified value.

Format:

FILVRM (integer address, integer count, byte value)

Where

Address is the starting video memory block location

Count is the number of bytes to write

Value is the number to fill the block with

Prerequisites:

* None

Errors:

* Invalid type if incorrect type passed

Example:

\_FILVRM (BASE(12),6144,0)

Sample code:

* BLIT.BAS
* FONT2.BAS

## GENCAL

Generic assembly call. Allows specifying registers AF, BC, DE, HL, IX and IY before calling specified address. Resulting register values are store back in the input array. Routine does not put RAM in pages 0 and 1 so one can call BIOS routines.

Format:

GENCAL (integer address, integer[5] registers)

Where

Address is the location of the routine to call

Registers in an array holding input and output register values. Order of registers in the array is: AF, BC, DE, HL, IX , IY

Prerequisites:

* None

Errors:

* Invalid type if incorrect type passed
* Subscript out of bounds if register array too short

Example:

REM COPY MSX FONT TO VRAM IN SCREEN 2

DIM R%(5)

R%(1)=2048:R%(2)=256\*PEEK(5)+PEEK(4):R%(3)=BASE(12)

\_GENCAL(&H5C,R%)

Sample code:

* COLLTEST.BAS

## MAXANIMDEFS

Allocates or deallocates memory for animation definitions.

Each definition consumes 16 bytes.

Format:

MAXANIMDEFS (integer number)

Where

Number is the maximum number of animation definitions.

Prerequisites:

* None

Errors:

* Invalid type if incorrect type passed

Example:

\_MAXANIMDEFS(5)

Sample code:

* ANIMTEST.BAS
* GAME.BAS

## MAXANIMITEMS

Allocates or deallocates memory for animation items.

Each definition consumes 5 bytes.

Format:

MAXANIMITEMS (integer number)

Where

Number is the maximum number of animation items.

Prerequisites:

* None

Errors:

* Invalid type if incorrect type passed

Example:

\_MAXANIMITEMS(5)

Sample code:

* ANIMTEST.BAS
* GAME.BAS

## MAXANIMSPRS

Allocates or deallocates memory for sprite or character animations.

Each definition consumes 8 bytes.

Format:

MAXANIMSPRS (integer number)

Where

Number is the maximum number of sprite or character animations.

Prerequisites:

* None

Errors:

* Invalid type if incorrect type passed

Example:

\_MAXANIMSPRS(5)

Sample code:

* ANIMTEST.BAS
* GAME.BAS

## MAXAUTOSGAMS

Allocates or deallocates memory for automatic sprite group animation and movement definitions.

Each definition consumes 24 bytes.

Format:

MAXAUTOSGAMS (integer number)

Where

Number is the maximum number of automatic sprite group animation and movement definitions.

Prerequisites:

* None

Errors:

* Invalid type if incorrect type passed

Example:

\_MAXAUTOSGAMS(5)

Sample code:

* ANIMTEST.BAS
* GAME.BAS

## MEMCPY

Copies a memory block from source to destination address. Can be used for pages 0 and 1.

Format:

MEMCPY (integer source, integer destination, integer count)

Where

source is the memory block location start location

destination is the address where to copy

count is the number to bytes to copy

Prerequisites:

* None

Errors:

* Invalid type if incorrect type passed

Example:

REM GET FREE MEMORY START ADDRESS IN PAGE 1

MB%=0

\_MEMCPY (&H4010, VARPTR(MB%), 2)

Sample code:

* DEMO2.BAS
* GAME.BAS

## MEMVRM

Copies a memory block from source address in RAM to destination address in VRAM. Can be used for pages 0 and 1.

Format:

MEMVRM (integer source, integer destination, integer count, byte wait\_vblank)

Where

source is the memory block location start location in RAM

destination is the address where to copy in VRAM

count is the number to bytes to copy

wait\_vblank flag to wait for vblank to copy data (0=no, >0 yes). If yes then assembler command HALT is issued before data copy.

Prerequisites:

* None

Errors:

* Invalid type if incorrect type passed

Example:

\_MEMVRM (&H100, BASE(12), 6144,0)

Sample code:

* DEMO2.BAS
* GAME.BAS
* FONT2.BAS

## SGAM

Sprite group animation and movement based on a description of a sprite group and animations.

Format:

SGAM (integer x, integer y, byte sprite\_group\_size, integer[2][] variable sprite\_group, byte item\_number, integer[] variable sprite\_animations )

Where:

X is horizontal sprite group location

Y is vertical sprite group location

Sprite\_group\_size defines number of sprites in a sprite group

Sprite\_group is an array describing a sprite group, for details refer to SPRGRPMOV command

Item\_number defines number of animations to step through

Sprite\_animations holds animation definitions for each sprite of a group

Prerequisites:

* Animations prepared with ANIMSPRITE

Errors:

* Invalid type if incorrect type passed
* Subscript out of range if ID invalid

Example:

DIM AL%(2):AL%(0)=0:AL%(1)=1:AL%(2)=2

DIM SG%(2,1):SG%(0,0)=0:SG%(1,0)=0:SG%(2,0)=0

SG%(0,1)=1:SG%(1,1)=0:SG%(2,1)=0

\_SGAM(50,60,2,SG%,3,AL%)

Sample code:

* ANIMTEST.BAS
* GAME.BAS

## SNDPLYINI

Initializes the sound player with music and optional sound effects data.

Format:

SNDPLYINI (integer music\_data, integer sfx\_data)

Where:

Music\_data is a memory location where music for AKG player is located

Sfx\_data is a memory location where sound effects for AKG player are located, if -1 no sound effects

Prerequisites:

* None

Errors:

* Invalid type if incorrect type passed

Example:

\_SNDPLYINI (&H100, &H1000)

Sample code:

* GAME.BAS

## SNDPLYOFF

Disables sound player and stops any running sounds.

Format:

SNDPLYOFF

Prerequisites:

* None

Errors:

* None

Example:

\_SNDPLYOFF

Sample code:

* GAME.BAS

## SNDPLYON

Starts the music player and disables key click.

Format:

SNDPLYON

Prerequisites:

* Player initialized with SNDPLYINI

Errors:

* Out of data if SNDPLYINI not called

Example:

\_SNDPLYON

Sample code:

* GAME.BAS

## SNDSFX

Plays sound effect on a specified channel.

Format:

SNDSFX (byte sfx\_number, byte channel, byte volume)

Where:

Sfx\_number is the ID of the sound effect (>0)

Channel is the channel number on which to play the effect (0, 1 or 2)

Volume is the inverted volume scale (0-16), where 0 is full volume and 16 is silent

Prerequisites:

* Player initialized with SNDPLYINI and sound effects

Errors:

* Out of data if SNDPLYINI not called with sound effect data specified
* Illegal function call if SNDPLYINI not called at all

Example:

\_SNDSFX (5,0,0)

Sample code:

* GAME.BAS

## SPRDISABLE

Disables sprites system.

Format:

SPRDISABLE

Prerequisites:

* None

Errors:

* None

Example:

\_SPRDISABLE

Sample code:

* GAME.BAS
* SPRITES.BAS
* ANIMTEST.BAS

## SPRENABLE

Initializes the sprite system.

Format:

SPRENABLE (integer[3][] variable sprite\_attributes, integer variable sprite\_update, byte flicker, byte num\_sprites\_handled)

Where:

Sprite\_attributes is an array describing sprite attributes in the form:

(0,n) – y coordinate

(1,n) – x coordinate

(2,n) – pattern

(3,n) - color

Sprite\_update is a variable to trigger VRAM update from sprite\_attributes, when set to <>0 an update will occur, and value set to 0. The use of animations will updates this flag to 1 as needed too.

Flicker <>0 will cause that sprite attributes are not applied to VRAM in the same order as in sprite\_attributes but cyclically effectively alleviating 4 sprites per line limitation.

Num\_sprites\_handled sets how many sprites are updated in each cycle. Use this for performance reasons if you do not work with full 32 sprites. Valid range is 0 <= num\_sprites\_handled <= 32.

Prerequisites:

* None

Errors:

* Invalid type if incorrect type passed
* Subscript out of range if incorrectly sized array provided

Example:

DIM SA%(3,31):SU%=0

\_SPRENABLE (SA%, SU%, 1, 32)

Sample code:

* GAME.BAS
* SPRITES.BAS
* ANIMTEST.BAS

## SPRGRPMOV

Command moves a group of sprites at the same time.

Format:

SPRGRPMOV (integer x, integer y, byte sprite\_group\_size, integer[2][] variable sprite\_group )

Where:

X is horizontal sprite group location

Y is vertical sprite group location

Sprite\_group\_size defines number of sprites in a sprite group

Sprite\_group is an array describing a sprite group.

(0,n) – sprite number

(1,n) – Dx

(2,n) – Dy

Prerequisites:

* Sprite system enabled

Errors:

* Invalid type if incorrect type passed
* Subscript out of range if array too small
* Illegal function call if sprite system disabled

Example:

DIM SG%(2,1):SG%(0,0)=0:SG%(1,0)=0:SG%(2,0)=0

SG%(0,1)=1:SG%(1,1)=0:SG%(2,1)=0

\_SPRGRPMOV(50,60,2,SG%)

Sample code:

* ANIMTEST.BAS
* GAME.BAS
* SPRITES.BAS

## TILERAM

Copies rectangular shape (tile) several times to destination location in RAM in a tiled fashion.

Diagram

Description automatically generated

Format:

TILERAM (integer TA, integer tile\_columns, integer tile\_rows, integer DA, integer dest\_columns, integer dest\_rows, integer x, integer y, integer num\_tiles\_horizontally, integer num\_tiles\_vertically)

Where

TA is memory location where tile data begins

Tile\_columns is the number of 8x8 pixel columns in a tile

Tile\_rows is the number of 8x8 pixel rows in a tile

DA is memory location where destination window begins

Dest\_columns is the number of 8x8 pixel columns in destination

Dest\_rows is the number of 8x8 pixel rows in destination

X is column in destination where to start applying tiles

Y is row in destination where to start applying tiles

Tiles\_horizontally is the number of tiles to apply in horizontal direction

Tiles\_vertically is the number of tiles to apply in vertical direction

Prerequisites:

* None

Errors:

* Invalid type if incorrect type passed

Example:

\_TILERAM(&HB000,1,1,&H100,32,24,0,0,32,24)

Sample code:

* DEMO2.BAS
* FONT2.BAS

## TILEVRM

Copies rectangular shape (tile) several times to destination location in VRAM in a tiled fashion. Function is used exclusively in SCREEN 2.

Format:

TILEVRM (integer TA, integer tile\_columns, integer tile\_rows, integer x, integer y, integer num\_tiles\_horizontally, integer num\_tiles\_vertically)

Where

TA is memory location where tile data begins

Tile\_columns is the number of 8x8 pixel columns in a tile

Tile\_rows is the number of 8x8 pixel rows in a tile

X is column in destination where to start applying tiles

Y is row in destination where to start applying tiles

Tiles\_horizontally is the number of tiles to apply in horizontal direction

Tiles\_vertically is the number of tiles to apply in vertical direction

Prerequisites:

* None

Errors:

* Invalid type if incorrect type passed

Example:

\_TILEVRM(&HB000,1,1,0,0,32,24)

Sample code:

* DEMO2.BAS

## VRMMEM

Copies a memory block from source address in VRAM to destination address in RAM. Can be used for pages 0 and 1.

Format:

VRMMEM (integer source, integer destination, integer count)

Where

source is the memory block location start location in VRAM

destination is the address where to copy in RAM

count is the number to bytes to copy

Prerequisites:

* None

Errors:

* Invalid type if incorrect type passed

Example:

\_VRMMEM (BASE(12), &H100, 6144)

Sample code:

None

# INFORMATIONAL DATA

Following memory locations contain useful data about ARTISAN basic extension

* &H4010 – free memory location start address in memory page 1. This is also returned by [ARTINFO](#_ARTINFO) command
* &H4012 – ARTISAN basic version in DAA format ab.cd. This is also returned by [ARTINFO](#_ARTINFO) command
  + &H4012 = aaaabbbb
  + &H4013 = ccccdddd

ARTISAN basic extension can be compiled with certain parts included or excluded. [ARTINFO](#_ARTINFO) will return flags used during compilation. Meaning of flags is given below:

Bit 0 – sound related commands: [SNDPLYINI](#_SNDPLYINI), [SNDPLYON](#_SNDPLYON), [SNDPLYOFF](#_SNDPLYOFF) and [SNDSFX](#_SNDSFX)

Bit 1 – main memory related commands: [MEMCPY](#_MEMCPY) and [FILRAM](#_FILRAM)

Bit 2 – video memory related commands: [FILVRM](#_FILVRM), [MEMVRM](#_MEMVRM) and [VRMMEM](#_VRMMEM)

Bit 3 – bitmap operations: [BLIT](#_BLIT)

Bit 4 – sprites related commands: [SPRENABLE](#_SPRENABLE), [SPRDISABLE](#_SPRDISABLE) and [SPRGRPMOV](#_SPRGRPMOV)

Bit 5 – generic assembly call: [GENCAL](#_GENCAL)

Bit 6 – tiling commands: [TILERAM](#_SPRSET) and [TILEVRM](#_TILEVRM)

Bit 7 – box commands: [BOXMEMCPY](#_BOXMEMCPY) and [BOXMEMVRM](#_BOXMEMVRM)

Bit 8 – animation commands: [MAXANIMITEMS](#_MAXANIMitems), [ANIMITEMPAT](#_ANIMITEMPAT), [ANIMITEMPTR](#_ANIMITEMPTR), [MAXANIMDEFS](#_MAXANIMDEFS_1), [ANIMDEF](#_ANIMDEF), [MAXANIMSPRS](#_MAXANIMSPRS), [ANIMSPRITE](#_ANIMSPRITE), [ANIMCHAR](#_ANIMCHAR), [MAXAUTOSGAMS](#_MAXAUTOSGAMS), [AUTOSGAMDEF](#_AUTOSGAMDEF), [AUTOSGAMSTART](#_AUTOSGAMSTART), [AUTOSGAMSTOP](#_AUTOSGAMSTOP), [ANIMSTEP](#_ANIMSTEP), [ANIMSTART](#_ANIMSTART), [ANIMSTOP](#_ANIMSTOP) and [SGAM](#_SGAM)

Bit 9 – collision detection: [COLL](#_COLL)

Bit 10 – signifies if basic extension commands are available through CALL or \_ syntax ([ARTINFO](#_ARTINFO) is always available)

Bit 11 – signifies if functionality is available through DEFUSR

Sample code can be found in AUTOEXEC.BAS file

# APPENDIX A – HOW TO ACCESS FUNCTIONS VIA DEFUSR COMMAND

When extension is compiled with access through DEFUSR (compile flags bit 11 = 1) one should prepare an integer array holding function ID followed by parameters, prepare a machine language call to extension, and pass an address of the parameters array in the DEFUSR call.

The assembly code of a routine to access ARTISAN extension from basic is similar to this:

RST #30

DB <SLOT ID>

DW #4014

EI

RET

Where SLOT ID is the value from RAMAD1 (#F342) location and #4014 is the entry point in ARTISAN basic.

In plain basic, one can use an array to hold this routine since it is fully relocatable.

REM JUMP ROUTINE

DIM JR%(2)

REM RST #30, SLOT ID

JR%(0)=&HF7:POKE VARPTR(JR%(0))+1,PEEK(&HF342)

REM ADDRESS

JR%(1)=&H4014

REM EI, RET

JR%(2)=&HC9FB

REM DEFUSR DEFINITION

DEFUSR=VARPTR(JR%(0))

Note that DEFUSR needs to be ran each time, before a call, if definition of JR array is not the last variable defined, since interpreter keeps changing memory locations of variables upon definition or removal.

To make an actual call, for example SPRENABLE, do the following:

REM PARAMETERS ARRAY

DIM ZZ%(4)

ZZ%(0)=0

ZZ%(1)=VARPTR(SA%(0,0))

ZZ%(2)=VARPTR(SU%)

ZZ%(3)=1

ZZ%(4)=32

DEFUSR=VARPTR(JR%(0))

O%=USR(VARPTR(ZZ%(0)))

Parameters are in the same order as if the commands are used. Any exceptions are noted in the command descriptions.

Output value from DEFUSR command holds no meaning and can’t be used as an indication if the executed command was successful.

Following table holds a list of function IDs.

|  |  |
| --- | --- |
| Function ID | Function |
| 0 | SPRENABLE |
| 1 | SPRDISABLE |
| 2 | MEMCPY |
| 3 | MEMVRM |
| 4 | BLIT |
| 5 | SGAM |
| 6 | SPRGRPMOV |
| 7 | COLL |
| 8 | SNDSFX |
| 9 | ANIMSTEP (single item) |
| 10 | ANIMSTEP (multiple items) |
| 11 | ANIMSTART (single item) |
| 12 | ANIMSTART (multiple items) |
| 13 | ANIMSTOP (single item) |
| 14 | ANIMSTOP (multiple items) |
| 15 | BOXMEMCPY |
| 16 | BOXMEMVRM |
| 17 | MAXANIMITEMS |
| 18 | MAXANIMDEFS |
| 19 | MAXANIMSPRS |
| 20 | MAXAUTOSGAMS |
| 21 | ANIMITEMPAT |
| 22 | ANIMITEMPTR |
| 23 | ANIMDEF |
| 24 | ANIMSPRITE |
| 25 | ANIMCHAR |
| 26 | AUTOSGAMDEF |
| 27 | AUTOSGAMSTART |
| 28 | AUTOSGAMSTOP |

# APPENDIX B – HOW TO SAFELY LOAD EXTENSION

ARTISAN basic extension should be loaded only once via BLOAD command. Doing it multiple times will lead to unexpected results. The safe way to load can be via ON ERROR and [ARTINFO](#_ARTINFO) commands.

10 REM ARTISAN BASIC LOADER

20 REM CHECK IF ALREADY LOADED

30 F%=0:V%=0:M%=0

40 ON ERROR GOTO 70

50 \_ARTINFO(V%,F%,M%)

60 GOTO 100

70 RESUME 80:ON ERROR GOTO 0

80 BLOAD "ARTISAN.bin",R

90 \_ARTINFO(V%,F%,M%)

100 PRINT "ARTISAN BASIC available"

Sample code can be found in AUTOEXEC.BAS file