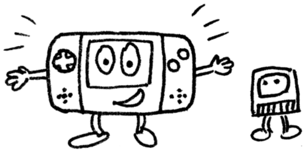
INTRODUCTION



Program retro games in BASIC

Make your own retro games on a virtual game console. Program in the classic BASIC language and create sprites, tile maps, sound and music with the included tools. As a beginner you will quickly understand how to create simple text games or show your first sprite on a tile map. As an experienced programmer you can discover the full potential of retro hardware tricks!

Virtual Game Console

Imagine LowRes NX as a handheld game console with a d-pad, two action buttons and a little rubber keyboard below a slidable touchscreen. LowRes NX was inspired by real 8- and 16-bit systems and simulates chips for graphics, sound and I/O, which actually work like classic hardware. It supports hardware sprites as well as hardware parallax scrolling, and even offers vertical blank and raster interrupts to create authentic retro effects.

Old-School Programming

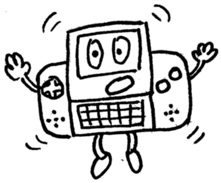
The programming language of LowRes NX is based on second-generation, structured BASIC. It offers all the classic commands, but with labels, loops and subprograms instead of line numbers. Graphics and sound are supported by additional commands and you can even access the virtual hardware directly using PEEK and POKE. You have complete control over the program flow, there is no standard update function to implement.

Creative Tools

LowRes NX includes all the tools you need: The Character Designer for editing sprites, tiles and fonts, the Background Designer for tile maps and screen layouts, as well as the Sound Composer for music and sound effects. All of these are just normal BASIC programs. You can change and improve them or even create your own custom editors.

Share and Play

Send your games directly to other users or share them via the website. All programs are open source, so you can play them, learn from them and edit them. Do you prefer making just art or music? Share your creations as assets and let other programmers use them in their projects.

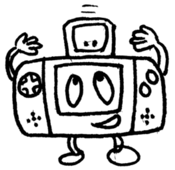


Specifications

- Cartridge ROM: 32 KB for gfx, music, any binary data
- Code: BASIC, max 16384 tokens
- Screen: 160x128 pixels, 60 Hz
- Backgrounds: Two layers, tile-based, scrollable

- Sprites: 64, max 32x32 pixels
- Colors: 8 dynamic 6-bit palettes with 4 colors each
- Sound: 4 voices, saw/tri/pulse/noise, pulse width, volume, ADSR, LFO
- Input: Two game controllers with d-pad and two buttons + pause
- Optional input: Keyboard and touchscreen/mouse

Getting Started



Try some of the included programs to see how LowRes NX can look like. Have a look at the action game *LowRes Galaxy 2*, the text adventure *LowRes Adventure*, and the demo *Star Scroller*.

> On the "My Programs" screen select a program to open the source code editor. Then tap on the Play button to run it.
>
> Find the examples in the folder "programs". Open the LowRes NX application and drag and drop any .nx file into its window. You can also select LowRes NX as default application for .nx files, so programs can be started simply by double clicking them.

Once you have played enough, you can create your first own program.

On the "My Programs" screen tap on the Plus button to create a new program. Select it to open the source code editor.

Use any text editor to create a new file. On Windows make sure the text editor supports Mac/Linux line ends, otherwise you may see everything in one line.

Type these lines:

```
PRINT "WELCOME!"
PRINT "WHAT IS YOUR NAME?"
INPUT ">";N$
PRINT "HELLO ";N$;"!"
```

Save your program file with a useful name and the extension ".nx".

Now run your program. This is a little example using the keyboard. Let's try something with a gamepad. Create another new program and type this:

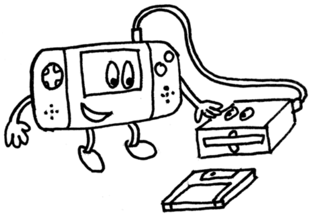
```
GAMEPAD 1
X=76
Y=60
DO
 IF UP(0) THEN Y=Y-1
 IF DOWN(0) THEN Y=Y+1
 IF LEFT(0) THEN X=X-1
 IF RIGHT(0) THEN X=X+1
 SPRITE 0,X,Y,225
 WAIT VBL
LOOP
```

Run this program and you will see an "A" on the screen which you can move around using the gamepad.

With the program still running, tap the menu button on the top right and select "Capture Program Icon". Now exit the program and return to the "My Programs" screen. There you will see your program with a new image. Long tap the icon and select "Rename..." to give it a better name.

EDITING

Programs and Data



A program file contains a complete game or application, including all its data, stored as simple text. The first part is the BASIC source code. Please read the programming chapters for further explanation.

The second part are the cartridge ROM entries. These are up to 16 numbered data blocks, which can contain any kind of data, for example graphics, level maps, music, etc. When a program is running, all its ROM entries are accessable in the first 32 KB of the memory.

You can easily create and edit ROM entries by using tools. Tools are normal NX programs, but they are specifically made for editing data. They can access any NX file as a "virtual disk" and use its ROM entries like files.

There are two ways of using tools:

- You can open tools directly like other NX programs. They will use the "Disk.nx" file in the tool's folder for loading and saving their data.
- You can open any program you want to edit and select a tool from a menu. This way the tool will access directly the data of the current program.

Let's try it. Open your program with the moving "A" (from the "Getting Started" chapter) and select "Char Designer" from the Tools menu.

> Open your program, tap the Tools button and select the tool.

> Run your program and press the Escape key to enter the development menu. Then press the Edit (ED) button and select the tool. By default the tools menu is empty, so drag and drop the programs "Char Designer", "BG Designer" and "Sound Designer" into the window.

Draw something as character #1 (keep #0 empty), then tap on "Disk" and save as file 2 ("Main Characters"). Now return to the source code editor and you will see some hexadecimal data below your program. This is your image! To see it, change the line

```
SPRITE 0,X,Y,225
```

to

```
SPRITE 0,X,Y,1
```

and run your program. There it is!

> Press Ctrl+R in the LowRes NX application to reload and run your current program. The Run button in the developer menu does the same.

Keep in mind that tools don't save automatically, so never forget to save before you exit them.

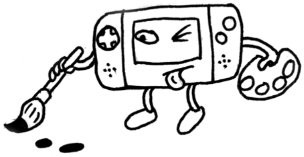
Standard ROM Entries

For an easy start you should use the ROM entry numbers of the following table. Their data is made ready for use automatically.

#0	Keep empty for default font
#1	Color palettes
#2	Characters (sprites, tiles)
#3	Background (tile map)
#15	Sounds and music

If cartridge ROM entry 0 is not used by a program, the compiler adds character data for the default font. It occupies the characters 192-255 and is automatically copied to video RAM on program start. If you want to use the default font, make sure to keep ROM entry 0 unused.

Character Designer



Use the Character Designer to draw your sprites and background tiles. It shows images in grayscale only, black is used as transparent background. Color palettes can be created later in the Background Designer.

The Character Designer loads file number 2 on startup, but keep in mind that it does not save automatically.

Main Screen

At the bottom part you see the 256 characters split into four pages. There you can select the current one for editing. Keep the last page empty, if you want to use the default font. Also character #0 should be empty for a clean background.

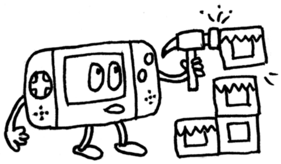
The top left square is for drawing the current character using the selected color.

FLIP	Flip the current character horizontally or vertically.
SPIN	Rotate the current character.
CLR	Clear the current character with the selected color.
CUT	Copy and clear the current character.
COP	Copy the current character.
PAS	Paste the copied character.
DISK	Go to the disk menu.
16*16	Go to the 16x16-pixel edit screen.
FON	Copy the standard font to the current page.
NEW	Clear all characters.

Disk Menu

The list shows all 16 files of the current virtual disk, or in other words the ROM entries of the program you are editing. Select one (usually number 2) and tap on Load or Save.

Background Designer



The Background Designer allows you to create tile-based screens or level maps of different sizes. Additionally it's for creating color palettes. On the bottom right there are three tabs for the different sections of the program: The map editor, the selection screen and the disk menu.

The Background Designer loads the main characters (file 1), palettes (file 2) and background (file 3) on startup, but keep in mind that it doesn't save automatically.

Selection Screen

On the top there is the character selector. Drag to select several characters at once. In the box at the left you select the current color palette. On the right you can edit each color of the current palette using the RGB sliders. The first color is usually transparent and unused, except the first color of palette 0, which is used as the screen backdrop color.

Map Editor

Here you draw your background using the character and color palette currently selected on the selection screen. On the bottom appear some tools:

Pan	Drag the visible map area.
Stamp	Draw with the selected character and color palette, using the selected flip and priority attributes.
Paint	Change the color palette only.
Priority	Change the priority only. Cells with priority 1 are shown in green, all others in red.
FLIP	Toggles the X/Y flip attributes.
PRI	Toggles the priority attribute.

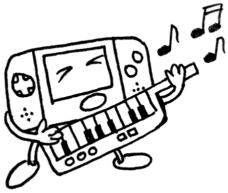
You can use the arrow keys to scroll the map when running LowRes NX on a computer.

Disk Menu

From here you load and save characters, color palettes and the background separately. There is no option to save everything at once. For the background there are two additional options:

NEW	Clear the complete background (keeping its size).
SIZE	Go to the size menu, where you can choose the width and height of the background (in cells), as well as the cell size (8x8 or 16x16 pixels).

Sound Composer



The Sound Composer serves to create sound presets, sequences ("tracks") and complete songs.

It loads the main sound data (file 15) on startup, but keep in mind that it doesn't save automatically. A sound file includes all sound and music data.

Structure

- 16 Sounds

A sound is a preset with all available sound parameters (waveform, envelope, LFO, etc.). It can be used directly with the PLAY command in your program or as an instrument in a track/song.

- 64 Tracks

A track is a sequence of 32 steps for a single voice, where each step can play a note or modify the sound. It can be used in your program for complex sound effects and short melodies (e.g. "level up") with the TRACK command. Tracks are also used to create songs.

- 64 Patterns

A pattern is a block of music, which defines which track should be played on each of the four voices. Patterns will be played one after another, except for the following cases: If the next pattern is empty, the song is stopped. When the current pattern finishes and has a "loop end" flag, the player jumps back to the previous pattern with a "loop start" flag. When a pattern finishes and has a "stop" flag, the player stops.
If you want to create music for a game and you plan to use additional sound effects, you should leave at least one voice empty.

- Songs

The 64 patterns can be used for one long song or several shorter ones. By using the "loop" and "stop" flags songs can be separated. To play one specific song, just start at its first pattern. To play a song in your program use the MUSIC command.

Editors

The Sound Composer has three tabs in the top right corner for the different sections: The pattern editor, the track editor and the sound editor. This is a "Tracker"-style program, which means the timeline goes from top to bottom, not from left to right. So the steps in a track are also called rows. Most values in the range from 0 to 15 are shown in hexadecimal format (0-F).

There is some keyboard support when running LowRes NX on a computer: The arrow keys move the cursor and the key rows simulate a musical keyboard with two octaves to enter notes. The return key enters a note stop, backspace deletes the current note and space toggles between play (pattern/track) and stop.

The Pattern Editor

Here you can select a pattern, choose its tracks for all voices and edit directly the notes of each selected track. There is also a toggle button for the "loop" and "stop" flags for the current pattern. If you set notes in a voice without track, it selects automatically a free track. Use the star symbol from the musical keyboard, if you want to stop/release a note in a track.

The Track Editor

Here you can select and edit a single track with additional parameters:

S	Sound
V	Volume
C	Sound command
P	Parameter

The Sound Editor

Define and test your sounds here. The currently selected sound will be used in the other editors for new notes.

Sound Commands

These commands allow you to change parameters dynamically while playback. Use them in the track editor.

C	P	Purpose
0	0	No command
0	x	Mix (1=left, 2=right, 3=center/both)
1	x	Attack Time
2	x	Decay Time
3	x	Sustain Level
4	x	Release Time

5	x	LFO Rate
6	x	LFO Frequency Amount
7	x	LFO Volume Amount
8	x	LFO Pulse Width Amount
9	x	Pulse Width
D	x	Slow Speed (like E, but +16)
E	x	Speed (ticks per row, 8 by default)
F	0	Break Track/Pattern
F	1	Cut Note / Volume 0

LANGUAGE BASICS

The programming language of LowRes NX is based on second-generation, structured BASIC (1985 style).

Types and Variables

Available data types are strings and numbers (floating point). Variable names can contain letters (A-Z), digits (0-9) and underscores (_), but cannot begin with a digit. Reserved keywords cannot be used as names, but they can contain them, for example:

Valid: `ENDING`
Invalid: `END`

String variable names must end with a $ symbol, for example:

```
NAME$
```

Variables are not explicitly declared, but they need to be initialized with a value before you can read from them. Values are assigned to variables using the equal symbol:

```
NAME$="LOWRES NX"
LIVES=3
```

Hexadecimal and binary notation can be used for number values:

```
$FF02
%11001011
```

Arrays

DIM

```
DIM [GLOBAL] var-list
```

Defines arrays with the highest index for each dimension:

```
DIM A(100)
DIM MAP(31,23,1),NAMES$(9),SCORES(9)
```

Access elements from arrays, indices start from 0:

```
SCORES(0)=100
SCORES(9)=5
PRINT SCORES(0),SCORES(9)
```

All elements of arrays are automatically initialized, using zeros (0) or empty strings ("").

With the optional GLOBAL keyword the arrays will be accessable from all subprograms.

Labels

A label marks a position in a program and is used for commands like GOTO. It consists of a name, using the same rules as for variables, followed by a colon:

```
GAMEOVER:
```

Operators

Arithmetic

Symbol	Example	Purpose
-	`-B`	Negation
^	`X^3`	Exponentiation
*	`2*Y`	Multiplication
/	`X/2`	Division
\	`X\2`	Integer Division
+	`C+2`	Addition
-	`100-D`	Subtraction
MOD	`X MOD 2`	Modulo

Operations are performed in mathematical order, for example multiplications and divisions are performed before additions and subtractions. The order can be specified explicitly through the use of parentheses, for example: (3+4*3)/5

Relational

Symbol	Example	Purpose
=	`A=10`	Equal
<>	`A<>100`	Not equal
>	`B>C`	Greater than
<	`5<X`	Less than
>=	`X>=20`	Greater than or equal
<=	`X<=30`	Less than or equal

Relational operator expressions have a value of true (-1) or false (0). For example: (2=3)=0, (4=4)=-1, (1<3)=-1

Logical/Bitwise

Symbol	Example	Purpose

NOT	`NOT (X=15)`
	`NOT 0`
		"Not"

AND	`A=1 AND B=12`
	`170 AND 15`
		"And"

OR	`X=10 OR Y=0`
	`128 OR 2`
		"Or"

XOR	`A XOR B`	"Exclusive Or"

How to Use Operators

All operators are available for numbers. Relational and addition operators are usable with strings, too:

```
SUM=1+3
IF SUM<5 THEN PRINT "LESS THAN 5"
NAME$="LOWRES NX"
GREET$="HELLO "+NAME$+"!"
IF NAME$>"LOWRES" THEN PRINT GREET$
```

PROGRAM FLOW CONTROL

Basics

REM

```
REM remark
' remark
```

Allows you to put comments into your program. REM lines are not executed. You can use an apostrophe (') in place of the word REM.

IF...THEN...ELSE

```
IF expr THEN command [ELSE command]
```

Checks if the given expression is true or false. If it's true, the command after THEN is executed, otherwise the one after ELSE. The ELSE part is optional.

If you want to execute more than one command, you can use the block version of the IF command. It must be closed with the line END IF.

```
IF expression THEN
    commands
[ELSE IF expression THEN]
    commands
[ELSE]
    commands
END IF
```

GOTO

```
GOTO label
```

Jumps to the given label and continues the program execution there.

GOSUB

```
GOSUB label
```

Adds the current program position to a stack and jumps to the given label. The program after the label is called a subroutine and must be finished using RETURN.

NOTE: Subroutines exist mostly for historical reasons. You should prefer the more powerful and safer subprograms.

RETURN

```
RETURN
```

Jumps back to the position of the last call of GOSUB and removes it from the stack.

```
RETURN label
```

Works like GOTO, but clears the whole stack. Use this to exit from a subroutine, if you want to continue your program somewhere else.

END

```
END
```

Stops the program from any position. The program is also stopped automatically after the last line of code.

WAIT VBL

```
WAIT VBL
```

Waits for the next frame. This (or WAIT n) should be the last command in all loops which do animations and/or handle input, like the main game loop.

WAIT

```
WAIT n
```

Waits *n* frames (n/60 seconds), where the minimum for *n* is 1. Subprograms running from interrupts (ON VBL/RASTER, MUSIC) will continue to work as normal during this period.

WAIT 1 is the same as WAIT VBL, so why is there WAIT VBL? Because it looks cooler and nerdier! A little guideline: Use WAIT VBL in loops for smooth animations and input handling, and WAIT n if you actually want to wait some time.

Loops

FOR...NEXT

```
FOR var=a TO b [STEP s]
    commands
NEXT var
```

Performs a series of commands in a loop a given number of times. The FOR command uses the variable *var* as a counter, starting with the value *a*. All commands until NEXT are executed, then the counter is increased by *s* (or +1 if STEP is omitted). A check is performed to see if the counter is now greater than *b*. If not, the process is repeated. If it is greater, the program continues with the lines after NEXT. If STEP *s* is negative, the loop is executed until the counter is less than value *b*.

DO...LOOP

```
DO
  commands
LOOP
```

Performs commands in an endless loop. You can use GOTO to exit it.

REPEAT...UNTIL

```
REPEAT
    commands
UNTIL expression
```

Executes the commands in a loop until the given expression is true. The loop is executed at least once.

WHILE...WEND

```
WHILE expression
    commands
WEND
```

Executes the commands in a loop as long as the given expression is true.

Subprograms

SUB...END SUB

```
SUB name [(parameter-list)]
  commands
END SUB
```

Defines a subprogram with the given name. The optional parameter list can contain two types of entries: simple variables and array variables (followed by an empty parentheses pair). Entries are separated by commas. By default all variables inside the subprogram are local.

NOTE: Don't use GOTO or GOSUB to jump out of a subprogram!

CALL

```
CALL name [(argument-list)]
```

Executes the subprogram with the given name and returns to the current position after finishing it. The argument list must match the parameters of the SUB definition. Simple variables, single array elements and entire arrays (followed by an empty parentheses pair) are passed by reference to the subprogram. Other expressions are passed by value.

EXIT SUB

```
EXIT SUB
```

Exits a subprogram before END SUB is reached.

GLOBAL

```
GLOBAL variable-list
```

Makes variables from the main program available to all subprograms. The list can contain simple variables only. For arrays you should use DIM GLOBAL. This command cannot be used within a subprogram.

TEXT

PRINT

```
PRINT expression-list
```

Outputs text to the current window. Expressions can be strings or numbers, separated by commas or semicolons. A comma separates the output with a space, a semicolon outputs without space. End the list with a comma or semicolon to keep the cursor at the end of the output, otherwise a new line is started.

INPUT

```
INPUT ["prompt";]var
```

Lets the user enter a text or number on the keyboard and stores it in the variable *var*. Optionally it can show a prompt text before (cannot be a variable).

INPUT automatically enables the keyboard.

LOCATE

```
LOCATE cx,cy
```

Moves the text cursor to column *cx* and row *cy* relative to the current window.

WINDOW

```
WINDOW cx,cy,w,h,b
```

Sets the text output window to cell position *cx,cy* and sets the size to *w* columns and *h* rows. Text will be written to background *b* (0 or 1).

CLW

```
CLW
```

Clears the window with spaces and resets the text cursor position.

USER INPUT

Gamepads

GAMEPAD

```
GAMEPAD n
```

Enables gamepads for *n* (1 or 2) players. Once the gamepad is enabled, the program cannot change to touchscreen/mouse input anymore.

=UP/DOWN/LEFT/RIGHT

```
UP(p)
DOWN(p)
LEFT(p)
RIGHT(p)
```

Returns true if the given direction is currently pressed on the direction pad of player *p* (0/1).

```
UP TAP(p)
DOWN TAP(p)
LEFT TAP(p)
RIGHT TAP(p)
```

With the optional TAP keyword, this function returns true only for the first frame the button is pressed.

=BUTTON

```
BUTTON(p[,n])
```

Returns true if button A (*n*=0) or B (*n*=1) is currently pressed by player *p* (0/1). If the parameter *n* is omitted, both buttons (A and B) are checked.

```
BUTTON TAP(p[,n])
```

With the optional TAP keyword, this function returns true only for the first frame the button is pressed.

PAUSE ON/OFF

```
PAUSE ON
PAUSE OFF
```

Enables or disables the automatic pause handling. By default it's enabled, so if you press the pause button, the program stops and shows "PAUSE" on the screen, until the button is pressed again.

PAUSE

```
PAUSE
```

Pauses the program and shows the default "PAUSE" screen, even if automatic pause handling is disabled.

=PAUSE

```
PAUSE
```

Returns true if the pause button was pressed, otherwise false. After calling this function its value is cleared, so it returns each button tap only once. The automatic pause handling needs to be disabled for this function.

Touchscreen/Mouse

Use touchscreen support only if you think it will work well with a computer mouse, too. If you want to create your own game buttons, keep in mind that your game might be unplayable on a computer, because it won't support the keyboard or a real gamepad. Always consider using the standard gamepad functions.

TOUCHSCREEN

```
TOUCHSCREEN
```

Enables the touchscreen/mouse support. Once it's enabled, the program cannot change to gamepad input anymore.

=TOUCH.X/Y

```
TOUCH.X
TOUCH.Y
```

Returns the current X or Y pixel position where the user touches the screen, or where it was touched the last time.

=TOUCH

```
TOUCH
```

Returns true if the screen is currently touched.

=TAP

```
TAP
```

Returns true if the screen is currently touched and was not touched the last frame.

Keyboard

KEYBOARD ON/OFF/OPTIONAL

```
KEYBOARD ON
KEYBOARD OFF
```

Enables or disables the keyboard. While the keyboard is enabled, gamepads don't work.

```
KEYBOARD OPTIONAL
```

Enables the keyboard, but won't show an on-screen keyboard on touchscreen devices. Programs using this mode should be completely usable with gamepad or touch control and use the keyboard for alternative input only.

=INKEY$

```
INKEY$
```

Returns the last pressed key as a string. If no key was pressed, it returns an empty string (""). After calling this function its value is cleared, so it returns each pressed key only once. The keyboard needs to be enabled for this function.

GRAPHICS

All graphics in LowRes NX are based on characters. A character is an 8x8-pixel image with 3 colors plus transparent. They are usually designed in black and white, but are displayed with one of the 8 programmable color palettes.

At program start all characters from ROM entry 2 are copied to video RAM to make them immediately usable.

The display is composed of 3 layers, which are from back to front:

- Background 1 (BG 1)
- Background 0 (BG 0)
- Sprites

Each sprite and background cell has an attribute called "priority". By setting it, the cell or sprite will appear on a higher display layer. Actually there are 6 layers, from back to front:

- Background 1 (BG 1) - prio 0
- Background 0 (BG 0) - prio 0

- Sprites - prio 0
- Background 1 (BG 1) - prio 1
- Background 0 (BG 0) - prio 1
- Sprites - prio 1

Sprites

Sprites are independent objects, which can be freely moved on the screen. They can have a size of 8x8 pixels (one character) or up to 32x32 pixels by grouping several characters. Each sprite has the standard character attributes (color palette, flip X/Y, priority) and additionally its size.

SPRITE

```
SPRITE n,[x],[y],[c]
```

Sets the position *(x,y)* and character *(c)* of sprite *n* (0-63). All parameters can be omitted to keep their current settings.

SPRITE OFF

```
SPRITE OFF [n]
SPRITE OFF a TO b
```

Hides one or more sprites. If all parameters are omitted, all sprites (0 - 63) are hidden. With one parameter only the given sprite is hidden. The last option is to hide sprites in the range from *a* to *b*.

SPRITE.A

```
SPRITE.A n,(pal,fx,fy,pri,s)
```

Sets the attributes of sprite *n* (0-63). All attribute parameters can be omitted to keep their current settings.

pal	palette number (0-7)
fx	flip horizontally (0/1)
fy	flip vertically (0/1)
pri	priority flag (0/1)
s	size (0-3): 0: 1 character (8x8 px) 1: 2x2 characters (16x16 px) 2: 3x3 characters (24x24 px) 3: 4x4 characters (32x32 px)

```
SPRITE.A n,a
```

Allows setting the attributes as a single 8-bit value.

=SPRITE.X/Y

```
SPRITE.X(n)
SPRITE.Y(n)
```

Return the position of sprite *n*.

=SPRITE.C

```
SPRITE.C(n)
```

Returns the character of sprite *n*.

=SPRITE.A

```
SPRITE.A(n)
```

Returns the attributes of sprite *n* as an 8-bit value.

=SPRITE HIT

```
SPRITE HIT(n[,a [TO b]])
```

Returns true if sprite *n* collides with another sprite (which means that pixels overlap). If no more parameters are given, it will check with all other visible sprites. If the *a* parameter is added, it will check only with that sprite *a*. If all parameters are given, it will check with all sprites from number *a* to number *b*.

=HIT

```
HIT
```

Returns the number of the sprite which collided with the sprite of the last call of SPRITE HIT.

Backgrounds

A background is a map of 32x32 character cells, which is used for text and tile based maps or images. Each cell has the information of which character it contains and additional attributes (color palette, flip X/Y, priority).

As a character has the size of 8x8 pixels, the resulting background size is 256x256 pixels, which is larger than the actual screen (160x128). By modifying the scroll offset of a background, the visible area can be moved.

If the visible area moves out of the borders of the background, the display wraps around the edges. This can be used to achieve endless scrolling.

There is a mode for 16x16-pixel cells. When active, each cell will show 2x2 characters, similar to big sprites. This mode also increases the background size to 512x512 pixels. Use the DISPLAY command to enable it.

For most of the commands and functions that access backgrounds their cell co-ordinates can be outside of the background size (32x32). They will be wrapped around the edges, so for example a character drawn at position 34,-2 will actually appear at position 2,30.

CLS

```
CLS
```

Clears both backgrounds with character 0 and resets the current window to the default one.

```
CLS b
```

Clears background *b* with character 0.

ATTR

```
ATTR (pal,fx,fy,pri,s)[,m]
```

Sets the current attributes for cell and text commands. All attribute parameters can be omitted to keep their current settings.

pal	palette number (0-7)
fx	flip horizontally (0/1)
fy	flip vertically (0/1)
pri	priority flag (0/1)
s	unused (0-3)

The optional parameter *m* is an 8-bit binary number used as a mask. It defines which bits of a cell attribute should actually be changed.

```
ATTR a[,m]
```

Allows setting the attributes as a single 8-bit value.

BG

```
BG n
```

Sets the current background (0 or 1).

CELL

```
CELL cx,cy,[c]
```

Sets the cell at position *cx,cy* of the current background to character *c*. If *c* is omitted, only the attributes of the cell get changed.

=CELL.C

```
CELL.C(cx,cy)
```

Returns the character of the cell at position *cx,cy* of the current background.

=CELL.A

```
CELL.A(cx,cy)
```

Returns the attributes of the cell at position *cx,cy* of the current background as an 8-bit value.

BG FILL

```
BG FILL cx1,cy1 TO cx2,cy2 [CHAR c]
```

Sets all cells in the area from *cx1,cy1* to *cx2,cy2* of the current background to character *c*. If CHAR *c* is omitted, only the attributes of the cells get changed.

BG SOURCE

```
BG SOURCE a[,w,h]
```

Sets the current source for the BG COPY command. The two-dimensional map starts at memory address *a*, has a width of *w* and a height of *h* cells.

Without the size parameters, Background Designer's data format is assumed: The width is read from address *a*+2, the height from *a*+3 and the actual map data starts at *a*+4.

By default ROM entry 3 is used as source.

BG COPY

```
BG COPY cx1,cy1,w,h TO cx2,cy2
```

Copies a two-dimensional part of the current source to the current background.

=MCELL.C/A

```
MCELL.C(cx,cy)
```

```
MCELL.A(cx,cy)
```

Work like the CELL.C and CELL.A functions, but get a cell from the source map (BG SOURCE) instead of the current background. If the co-ordinates are outside of the map bounds, the functions return -1.

MCELL

```
MCELL cx,cy,[c]
```

Works like the CELL command, but sets a cell in the source map (BG SOURCE) instead of the current background. The source must be in working RAM, otherwise you will get an "Illegal Memory Access" error.

BG SCROLL

```
BG SCROLL cx1,cy1 TO cx2,cy2 STEP dx,dy
```

Moves the content of all cells in the area from *cx1,cy1* to *cx2,cy2* horizontally by *dx* and vertically by *dy* cells.

TEXT

```
TEXT cx,cy,s$
```

Outputs the string *s$* to the current background at cell position *cx,cy*.

NUMBER

```
NUMBER cx,cy,n,d
```

Outputs the number *n* to the current background at cell position *cx,cy*. The number is formatted to show always *d* digits. This command is preferred over TEXT to show numbers, as it doesn't need to convert numbers to strings.

FONT

```
FONT c
```

Sets the current character range used for text output. *c* is the character where the font starts (space).

The default value is 192, which points to the standard font, if available.

SCROLL

```
SCROLL b,x,y
```

Sets the scroll offset of background *b* (0/1) to pixel co-ordinates *x,y*.

=SCROLL.X/Y

```
SCROLL.X(b)
SCROLL.Y(b)
```

Return the scroll offset of background *b*.

Display Settings

DISPLAY

```
DISPLAY (s,b0,b1,c0,c1)
```

Sets the display attributes. All attribute parameters can be omitted to keep their current settings.

s	sprites enabled (0/1)
b0	background 0 enabled (0/1)
b1	background 1 enabled (0/1)
c0	BG 0 cell size,
c1	BG 1 cell size: 0: 1 character (8x8 pixels) 1: 2x2 characters (16x16 pixels)

```
DISPLAY a
```

Allows setting the attributes as a single 8-bit value.

=DISPLAY

```
DISPLAY
```

Returns the display attributes as an 8-bit value.

PALETTE

```
PALETTE n,[c0],[c1],[c2],[c3]
```

Sets all four colors of palette *n* (0-7). Color 0 is only used for palette 0 and shown as the screen's backdrop color. The color parameters can be omitted to keep their current settings. Valid color values are 0-63 and can be calculated like this:

```
VALUE = RED * 16 + GREEN * 4 + BLUE
```

RED, GREEN and BLUE are values from 0 to 3.

By default all palettes are read from ROM entry 1.

=COLOR

```
COLOR(p,n)
```

Returns the value of color *n* (0-3) from palette *p* (0-7). You can get the RED, GREEN and BLUE values like this:

```
RED = INT(VALUE / 16)
GREEN = INT(VALUE / 4) MOD 4
BLUE = VALUE MOD 4
```

ON RASTER CALL/OFF

```
ON RASTER CALL name
```

Sets a subprogram which is executed for every screen line before it's drawn. Usually used to change color palettes or scroll offsets to achieve graphical effects. Raster subprograms must be short (see "CPU Cycles").

```
ON RASTER OFF
```

Removes the current subprogram.

=RASTER

```
RASTER
```

Returns the current screen line (y position). Use this in a raster subprogram.

ON VBL CALL/OFF

```
ON VBL CALL name
```

Sets a subprogram which is executed each frame. Can be used to update animations or sounds, even if the main program is blocked by WAIT or INPUT. VBL subprograms should not be very long (see "CPU Cycles").

```
ON VBL OFF
```

Removes the current subprogram.

=TIMER

```
TIMER
```

Returns the number of frames shown since LowRes NX was started. The value wraps to 0 when 5184000 is reached, which is about 24 hours.

SOUND

LowRes NX has four independent sound generators (voices). Each one can play sawtooth, triangle, pulse and noise waveforms, and has frequency, volume and pulse width settings. An additional envelope generator and LFO per voice makes complex sounds and instruments possible.

You can use the tool "Sound Composer" (or compatible programs) to create music, tracks and sound presets.

MUSIC

```
MUSIC [p]
```

Starts playback of a song at pattern *p*. If the parameter *p* is omitted, it starts at pattern 0.

TRACK

```
TRACK n,v
```

Plays track *n* once on voice *v*. Each voice can play a track independently, so this can be used for sound effects, even while music is playing.

PLAY

```
PLAY v,p[,len] [SOUND s]
```

Plays a sound on voice *v* (0-3). *p* is the pitch:

Note	Pitch (with different octaves)							
C	1	13	25	37	49	61	73	85
C#	2	14	26	38	50	62	74	86
D	3	15	27	39	51	63	75	87
D#	4	16	28	40	52	64	76	88
E	5	17	29	41	53	65	77	89
F	6	18	30	42	54	66	78	90
F#	7	19	31	43	55	67	79	91
G	8	20	32	44	56	68	80	92
G#	9	21	33	45	57	69	81	93
A	10	22	34	46	58	70	82	94
A#	11	23	35	47	59	71	83	95
B	12	24	36	48	60	72	84	96

The optional parameter *len* is the length in 1/60 seconds, the maximum is 255. 0 means, that the the sound won't stop automatically. If the parameter is omitted, the current value of the voice is kept.

By default the current sound settings of the voice are used. Add the SOUND parameter to use the sound number *s* from the Sound Composer tool.

STOP

```
STOP [v]
```

Stops the current sound and track on voice *v*. If the parameter is omitted, all voices, tracks and music are stopped. If a voice's envelope has a release time, the sound won't stop immediately, but fade out.

VOLUME

```
VOLUME v,[vol],[mix]
```

Sets the volume of voice *n* (0-3) to *vol* (0-15) and its outputs to *mix* (0-3):

0	Muted
1	Left
2	Right
3	Left and right (center)

All parameters can be omitted to keep their current settings.

SOUND

```
SOUND v,[w],[pw],[len]
```

Sets the basic sound parameters of voice *v* (0-3).

w is the waveform:

0	Sawtooth
1	Triangle
2	Pulse
3	Noise

pw is the pulse width (0-15), a value of 8 results in a square wave. This parameter only has an effect for the pulse waveform.

len is the sound length in 1/60 seconds, the maximum is 255. 0 means, that the sound won't stop automatically. If the length is set using this command, the length parameter of PLAY can be omitted.

All parameters can be omitted to keep their current settings.

ENVELOPE

```
ENVELOPE v,[a],[d],[s],[r]
```

Sets the volume envelope generator of voice *v* (0-3).

a is the attack time, *d* is the decay time, and *r* is the release time. All times are non-linear and range from 0 (2 ms) to 15 (12 s)

s is the sustain level (0-15), which is the volume after the decay time and before the sound gets released.

All parameters can be omitted to keep their current settings.

LFO

```
LFO v,[r],[fr],[vol],[pw]
```

Sets the LFO (low frequency oscillator) of voice *v* (0-3).

r is the LFO rate and ranges from 0 (0.12 Hz) to 15 (18 Hz) in a non-linear manner.

The other paramters set the amount of the effect on different sound parameters: *fr* for frequency/pitch, *vol* for volume and *pw* for pulse width. These values range from 0 to 15.

All parameters can be omitted to keep their current settings.

LFO.A

```
LFO.A v,(w,r,e,t)
```

Sets additional LFO attributes of voice *v* (0-3). All attribute parameters can be omitted to keep their current settings.

w	wave (0-3): 0: triangle

	1: sawtooth 2: square 3: random
r	revert (0/1)
e	env mode enabled (0/1)
t	trigger enabled (0/1)

By default the LFO adds its output to the normal sound parameters. If revert is enabled, it subtracts. By enabling the env mode, the LFO stops after one cycle, so it can be used as an additional envelope generator. If the trigger is enabled, the LFO restarts for each played sound, otherwise it runs continuously. Trigger is enabled implicitly with the env mode.

SOUND SOURCE

```
SOUND SOURCE a
```

Sets the current data source for the PLAY, MUSIC and TRACK commands to the memory address *a*. This only affects the following calls to these commands, already started playback keeps its own data source. The data is assumed to be in the format of the Sound Composer tool.

By default ROM entry 15 is used as source.

DATA

DATA

```
DATA constant-list
```

Stores comma separated numeric and string constants (values, but no variables or expressions) that are accessed by the READ command. DATA commands are not executed and may be placed anywhere in the program.

READ commands access DATA in order, from the top of a program until the bottom. All constants of all DATA commands are read as one continuous list of items.

READ

```
READ var-list
```

Reads values from DATA commands and assigns them to the comma separated variables in *var-list*. The program has an internal pointer to the current DATA value. With each value read, the pointer will move to the next DATA value.

RESTORE

```
RESTORE [label]
```

Changes the internal read pointer to another position. This allows to reread data or to select specific data. If the label parameter is omitted, READ will start again from the top of the program. Otherwise the pointer will be set to the jump label.

MEMORY ACCESS

LowRes NX simulates chips for graphics, sound and I/O, the cartridge ROM, working RAM and persistent RAM. Everything is accessible in a 64 KB memory map, which is described in the chapter "Hardware Reference".

=PEEK

```
PEEK(a)
```

Returns the byte value (0-255) at memory address *a*.

POKE

```
POKE a,v
```

Sets the memory at address *a* to value *v*. *v* is a numeric expression from 0 to 255; numeric expressions outside this range are truncated to 8 bits.

=PEEKW

```
PEEKW(a)
```

Returns the two-byte value (-65536 to 65535) at memory address *a*.

POKEW

```
POKEW a,v
```

Writes a two-byte value at memory address *a*. *v* is a numeric expression from -65536 to 65535; numeric expressions outside this range are truncated to 16 bits.

=PEEKL

```
PEEKL(a)
```

Returns the four-byte value (-2147483648 to 2147483647) at memory address *a*.

POKEL

```
POKEL a,v
```

Writes a four-byte value at memory address *a*. *v* is a numeric expression from -2147483648 to 2147483647; numeric expressions outside this range are truncated to 32 bits.

COPY

```
COPY a,n TO d
```

Copies *n* bytes starting from memory address *a* to address *d*. The source and the destination areas may overlap.

FILL

```
FILL a,n[,v]
```

Sets *n* bytes starting from memory address *a* to value *v*, or 0 if the parameter is omitted.

ROL

```
ROL a,n
```

Takes the byte at address *a* and rotates its bits left by *n* places.

ROR

```
ROR a,n
```

Takes the byte at address *a* and rotates its bits right by *n* places.

=ROM

```
ROM(n)
```

Returns the memory address of ROM entry *n*.

=SIZE

```
SIZE(n)
```

Returns the number of bytes of ROM entry *n*.

FILES

The file commands can be used to store data on a virtual disk, which can contain up to 16 files. Its format is the same as the ROM entries part in a program file. This makes it possible to use any NX program directly as a virtual disk to edit its data.

Virtual disks are meant to be used for development tools only, for example image and map editors or music programs. Games should use persistent memory instead. Imagine that the standard LowRes NX console wouldn't have a disk drive.

LOAD

```
LOAD f,a
```

Loads the file number *f* from the current virtual disk to memory starting at address *a*.

LOAD is meant to be used for tools only. Use ROM entries for game data.

SAVE

```
SAVE f,c$,a,n
```

Saves *n* bytes starting at memory address *a* to the current virtual disk as a file number *f* (0-15) with comment *c$* (up to

31 characters).

If this file was loaded before, consider keeping its original comment or allow the user to edit it before saving. If the file is new, the comment should contain at least the type of data, e.g. "CHARACTERS" or "MUSIC".

SAVE is meant to be used for tools only. Use persistent memory to store game states.

FILES

```
FILES
```

Loads the current file directory for use with FILES$.

=FILE$

```
FILE$(f)
```

Returns the comment string of file number *f*. Call FILES before accessing the file directory to update its content, or use FILE$ directly after LOAD or SAVE.

=FSIZE

```
FSIZE(n)
```

Returns the number of bytes of file number *n*. Call FILES before accessing the file directory to update its content, or use FSIZE directly after LOAD or SAVE.

DEBUGGING

TRACE

```
TRACE expression-list
```

Outputs text to the debugging window. Expressions can be strings or numbers, separated by commas. This command is ignored if the debugging mode is not enabled.

MATH FUNCTIONS

Trigonometric

=PI

```
PI
```

PI is the ratio of the circumference of a circle to its diameter: 3.1415926535...

=SIN

```
SIN(x)
```

The sine of *x*, where *x* is in radians.

=COS

```
COS(x)
```

The cosine of *x*, where *x* is in radians.

=TAN

```
TAN(x)
```

The tangent of *x*, where *x* is in radians.

=ASIN

```
ASIN(x)
```

The arc sine of *x*, where *x* must be in the range of -1 to +1. The range of the function is -(PI/2) < ASIN(x) < (PI/2).

=ACOS

```
ACOS(x)
```

The arc cosine of *x*, where *x* must be in the range of -1 to +1. The range of the function is -(PI/2) < ACOS(x) < (PI/2).

=ATAN

```
ATAN(x)
```

The arctangent of *x* in radians, i.e. the angle whose tangent is *x*. The range of the function is -(PI/2) < ATAN(x) < (PI/2).

=HSIN

```
HSIN(x)
```

The hyperbolic sine of *x*.

=HCOS

```
HCOS(x)
```

The hyperbolic cosine of *x*.

=HTAN

```
HTAN(x)
```

The hyperbolic tangent of *x*.

Standard Math

=ABS

```
ABS(x)
```

The absolute value of *x*.

=SGN

```
SGN(x)
```

The sign of *x*: -1 if $x < 0$, 0 if $x = 0$ and +1 if $x > 0$.

=INT

```
INT(x)
```

The largest integer not greater than *x*; e.g. INT(1.3) = 1 and INT(-1.3) = -2.

=EXP

```
EXP(x)
```

The exponential of *x*, i.e. the value of the base of natural logarithms (e = 2,71828...) raised to the power *x*.

=LOG

```
LOG(x)
```

The natural logarithm of *x*; *x* must be greater than zero.

=SQR

```
SQR(x)
```

The nonnegative square root of *x*; *x* must be nonnegative.

Random Sequences

=RND

```
RND
```

The next number in a sequence of random numbers uniformly distributed in the range 0 <= RND < 1.

```
RND(n)
```

The second syntax generates a random integer between 0 and *n* inclusive.

RANDOMIZE

```
RANDOMIZE x
```

Sets the seed for random numbers to *x*, which should be an integer value. By default a program starts with seed 0, so the sequence of random numbers is always the same.

```
RANDOMIZE TIMER
```

If you want different random numbers each time you run your program, you should insert this line at the beginning.

Manipulating Numbers

=MIN

```
MIN(x,y)
```

The MIN function returns the smallest value of two expressions.

=MAX

```
MAX(x,y)
```

The MAX function returns the largest value of two expressions.

SWAP

```
SWAP var1,var2
```

Swaps the data between any two variables of the same type.

INC

```
INC var
```

Increases the value of the variable by one. INC A does the same as A=A+1, but costs less CPU cycles.

DEC

```
DEC var
```

Decreases the value of the variable by one. DEC A does the same as A=A-1, but costs less CPU cycles.

ADD

```
ADD var,x
```

Adds the value *x* to the variable, where *x* can the positive or negative. ADD A,X does the same as A=A+X, but costs less CPU cycles.

```
ADD var,x,base TO top
```

The second syntax of ADD helps with repeating counters. It's the same as:

```
A=A+X
IF A>TOP THEN A=BASE
IF A<BASE THEN A=TOP
```

But again the ADD command costs less CPU cycles.

STRING FUNCTIONS

=LEFT$=

```
LEFT$(s$,n)
```

Returns a new string with the first *n* characters of *s$*.

```
LEFT$(s$,n)=a$
```

Overwrites the first characters in the variable *s$* with the first *n* characters of *a$*.

=RIGHT$=

```
RIGHT$(s$,n)
```

Returns a new string with the last *n* characters of *s$*.

```
RIGHT$(s$,n)=a$
```

Overwrites the last characters in the variable *s$* with the last *n* characters of *a$*.

=MID$=

```
MID$(s$,p,n)
```

Returns a new string with *n* characters of *s$*, starting at character *p*. The first character has the position 1.

```
MID$(s$,p,n)=a$
```

Overwrites the given text range in the variable *s$* with the first *n* characters of *a$*.

=INSTR

```
INSTR(d$,s$[,p])
```

Searches the first occurrence of *s$* inside of *d$* and returns its start position. If it's not found, the function returns 0. Usually the function starts searching at the beginning of the string. Optionally it can start searching at position *p*.

=CHR$

```
CHR$(n)
```

Returns a string containing one character with ASCII code *n*.

=ASC

```
ASC(a$)
```

Supplies you with the ASCII code of the first character of *a$*.

=LEN

```
LEN(a$)
```

Returns the number of characters in *a$*.

=VAL

```
VAL(a$)
```

Converts a number written in *a$* into a numeric value.

=STR$

```
STR$(n)
```

Converts the number *n* into a string.

=BIN$

```
BIN$(n[,len])
```

Converts the number *n* into a binary string with at least *len* digits.

=HEX$

```
HEX$(n[,len])
```

Converts the number *n* into a hexadecimal string with at least *len* digits.

ADVANCED TOPICS

CPU Cycles

LowRes NX has a simplified simulation of CPU cycles. There is a fixed limit of cycles per frame. This assures the same program execution speed on all devices, so if you optimize your program on your device to run smoothly, it will run the same on all other devices.

Each execution of a command, function or operator, as well as access to a variable or a constant count 1 cycle. Some operations have additional costs:

- String creation and modification count 1 cycle per letter.
- Array initialization counts 1 cycle per element.
- Memory area modification counts 1 cycle per byte (not single byte modifications like POKE).
- BG area modification and text output count 2 cycles per cell (not single cell modifications like CELL).

Total cycles per frame	17556
Cycles per VBL interrupt	1140
Cycles per raster interrupt	51

The main program may spend any number of cycles, but when the limit is reached before a WAIT VBL or WAIT command, the execution continues in the next frame. If interrupts exceed their limit, you will see black scanlines on the screen.

Hardware Reference

Memory Map

```
$0000 - Cartridge ROM (32 KB)

$8000 - Character Data (4 KB)
$9000 - BG0 Data (2 KB)
$9800 - BG1 Data (2 KB)

$A000 - Working RAM (16 KB)

$E000 - Persistent RAM (4 KB)

$FE00 - Sprite Registers (256 B)
$FF00 - Color Registers (32 B)
$FF20 - Video Registers
```

```
$FF40 - Audio Registers
$FF70 - I/O Registers
```

Character Data

A character is an 8x8-pixel image with 2 bits per pixel, with a resulting size of 16 bytes. The video RAM has space for 256 characters.

The first 8 bytes of a character contain the low bits of all its pixels, followed by 8 more bytes containing the high bits of all pixels.

BG Data

A background is a map of 32x32 character cells. Each cell occupies two bytes:

```
- Character number
- Attributes:
    Bit  Purpose
    0-2  Palette number
    3    Flip X
    4    Flip Y
    5    Priority
    6-7  Unused
```

Persistent RAM

Imagine it as a battery buffered RAM on the game cartridge. Use it for data like game positions or high score tables. The content of the persistent RAM will be saved automatically when you exit the program and loaded when you run it. Each program saves its persistent RAM separately.

Sprite Registers

There are 64 sprites available, each occupies 4 bytes:

```
- X position
- Y position
- Character number
- Attributes:
    Bit  Purpose
    0-2  Palette number
    3    Flip X
    4    Flip Y
    5    Priority
    6-7  Size:
         0: 1 character (8x8 px)
         1: 2x2 characters (16x16 px)
         2: 3x3 characters (24x24 px)
         3: 4x4 characters (32x32 px)
```

Note: X and Y sprite position registers have an offset of 32, so they can move out of the top/left screen borders without using negative numbers. Using the BASIC commands, this offset is removed for convenience.

Color Registers

There are 8 palettes of each 4 colors. One color is one byte:

```
Bits  Component
0-1   Blue
2-3   Green
4-5   Red
```

Video Registers

```
$FF20 - Attributes:
    Bit  Purpose
    0    Sprites enabled
    1    BG0 enabled
    2    BG1 enabled
    3    BG0 cell size,
    4    BG1 cell size:
         0: 1 character (8x8 px)
            (BG 256x256 px)
         1: 2x2 characters (16x16 px)
            (BG 512x512 px)

$FF21 - BG0 scroll offset X
$FF22 - BG0 scroll offset Y
$FF23 - BG1 scroll offset X
$FF24 - BG1 scroll offset Y
$FF25 - Scroll offset MSB
```

```
    (most significant bits)
    used for big cell size only:
    Bit  Purpose
    0    BG0 X+256
    1    BG0 Y+256
    2    BG1 X+256
    3    BG1 Y+256

$FF26 - Raster line
```

Audio Registers

There are registers for 4 voices:

```
$FF40 - Voice 0
$FF4C - Voice 1
$FF58 - Voice 2
$FF64 - Voice 3
```

Each voice occupies 12 bytes:

```
- Frequency low-byte
- Frequency high-byte
- Status:
    Bit  Purpose
    0-3  Volume
    4    Mix to left
    5    Mix to right
    6    Init
    7    Gate
- Peak meter (read only)
- Attributes:
    Bit  Purpose
    0-3  Pulse width
    4-5  Wave:
         0: Sawtooth
         1: Triangle
         2: Pulse
         3: Noise
    6    Timeout enabled
- Length (timeout)
- Envelope byte 1:
    Bit  Purpose
    0-3  Attack
    4-7  Decay
- Envelope byte 2:
    Bit  Purpose
    0-3  Sustain
    4-7  Release
- LFO attributes:
    Bit  Purpose
    0-1  Wave:
         0: Triangle
         1: Sawtooth
         2: Square
         3: Random
    2    Invert
    3    Env mode enabled
    4    Trigger enabled
- LFO settings byte 1:
    Bit  Purpose
    0-3  LFO Rate
    4-7  Frequency amount
- LFO settings byte 2:
    Bit  Purpose
    0-3  Volume amount
    4-7  Pulse width amount
- Reserved
```

Note: The frequency is a 16-bit value: f = hertz * 16

I/O Registers

```
$FF70 - Gamepad 0 status
$FF71 - Gamepad 1 status

Gamepad status:
    Bit  Purpose
    0    Up
    1    Down
    2    Left
    3    Right
    4    Button A
    5    Button B

$FF72 - Last touch X position
$FF73 - Last touch Y position
$FF74 - Last pressed key (ASCII code)
$FF75 - Status:
    Bit  Purpose
    0    Pause button
    1    Touch

$FF76 - Attributes:
    Bit  Purpose
    0-1  Gamepads enabled:
        0: off
        1: 1 player
        2: 2 players
    2   Keyboard enabled
    3   Touchscreen enabled
```

Sound Data Format

This is the format used for the PLAY, MUSIC and TRACK commands. It's valid to store only the sound presets, if no MUSIC or TRACK commands are used with this data. If any tracks are available, all patterns must be stored. Empty tracks after the last used one don't need to be stored.

```
Offset - Content
0      - 16 sound presets
128    - 64 patterns
384    - 64 tracks
```

Each sound preset occupies 8 bytes and matches the format of the audio registers of one voice, but without the first 4 bytes.

Each pattern occupies 4 bytes:

```
- Voice 0:
    Bit  Purpose
    0-6  Track index
         (64 = voice unused)
    7    Flag loop start
- Voice 1:
    Bit  Purpose
    0-6  Track index
         (64 = voice unused)
    7    Flag loop end
- Voice 2:
    Bit  Purpose
    0-6  Track index
         (64 = voice unused)
    7    Flag song stop
- Voice 3:
    Bit  Purpose
    0-6  Track index
         (64 = voice unused)
```

Each track occupies 96 bytes and consists of 32 entries with each 3 bytes:

```
- Note pitch (0 = empty)
- Data (ignored if note is 0):
    Bit  Purpose
    0-3  Volume
    4-7  Sound
- Control:
    Bit  Purpose
    0-3  Parameter
    4-7  Command
```