an

AnyWareDesigns

production

M64 manual

Table of contents

1. Introduction

If you think that your c64 sounds fantastic and you make music with sequencers and synthesizers you will probably want to use your c64 as a synth module. *M64 will transform your c64 into a synth module*.

M64 features:

- Built in editor, compiler and player
- Mouse support
- Menu system
- Many modulation possibilites (General MIDI modulations)
- Assembly-like programming language for sound creation
- Mono- or polyphonic mode

2. Requirements

The following is required to use M64:

- C64 or C128 (in C64 mode)
- TV or Monitor
- MIDI interface (Passport-, Sequential- or DATEL/Seil/JMS-compatible)

The following is recommended:

- 1351 compatible mouse (or a joystick)
- Some kind of storage device. Disk drive or hard drive preferred

To get most out of this program:

- 1351 compatible mouse
- JiffyDOS or other ROM-based disk turbo
- Monitor
- SuperCPU

3. Installation

3.1 M64 files

The M64 package contains the following files:

M64.prg
: Standard executable version of the M64.

M64.msx
: MIDI SYSEX version of M64.

M64.txt
: This manual in ISO 8859-1 format.

M64.html
: This manual in HTML format.

3.2 MIDI transfer

To install M64 you need to transfer all executable files to a storage device on your c64. This manual contains a **midiloader** that will use your MIDI interface to receive executable files.

Midiloader usage:

Type in the midiloader from the listings below. You have to choose the correct one for your MIDI interface. Save the program (so you won't have to retype everything if something goes wrong). Run it. The program will tell if it was typed in correctly.

If everything is all right the program will wait for you to press return on a SYS49152 line. Do so.

The midiloader is now waiting for you to transfer any file from the M64 package with an .MSX extension. Start sending M64.MSX from your remote computer or synthesizer. The border flashes during transfer. If the file was correctly transferred just save it as you would do with any other BASIC program.

Below are three listings of the midiloader program. Choose the correct one. If you have absolutely no idea what type of interface you have you can type in the following program that will identify your MIDI interface.

3.3 Interface identifier

```
5 AD=56832
10 IFPEEK(AD+2)<>255THEN50
20 IFPEEK(AD+6)<>255THEN60
30 IFPEEK(AD+8)<>255THEN70
40 PRINT"NO INTERFACE CONNECTED":END
50 PRINT"SEQUENTIAL INTERFACE":END
60 PRINT"DATEL/SEIL/JMS INTERFACE":END
70 PRINT"PASSPORT INTERFACE"
```

3.4 Midiloader for PASSPORT-compatible interface (Sentech)

```
1 AD=49150:J=0:T=0:L=30
5 RT=0:READA$:IFA$=""THEN100
10 FORI=1TOLEN(A$)/2
15 GOSUB200:B=A:GOSUB200:A=B*16+A
25 POKEAD+J,A:J=J+1:RT=RT+A:NEXT:T=T+RT
27 READA:IFRT<>ATHENPRINT"DATA ERROR IN LINE "L:END
29 L=L+1:GOTO5
30 DATA "AAMAHIKJADCAKGMA",874
31 DATA "KJBFCAKGMAKJABKC",912
32 DATA "AIIFCNIGCOKNCANA",779
33 DATA "EIKJAAINCANAKCAI",792
34 DATA "CAEOMANNKKMANAPG",1339
35 DATA "MKBAPFKAAAOOCANA",1101
36 DATA "CAEOMADAEDAKAKAK",447
37 DATA "AKINECMACAEOMADA",759
38 DATA "CPCJAPAJAAJBCNOG",532
39 DATA "CNNAACOGCOEMCLMA",842
40 DATA "KJHPINAANMKNABNM",1051
41 DATA "BABEKNAINOINGDMA",871
42 DATA "CJHANAAKKJAACJAB",582
43 DATA "PAOLKNAJNOGAGIGI",1183
44 DATA "KCAACAJKMAEMHPMA",935
45 DATA "MJPHNAPEKCAICAJK",1256
46 DATA "MAKFCNKGCOIFKOIG",1055
47 DATA "KPGIINCANAFIGAEF",913
48 DATA "FCFCEPFCCBANAAEP",450
49 DATA "ELCBANAALNINMAPA",883
50 DATA "AGCANCPPOINAPFGA",1284
51 DATA "INAINOGAAIAAABAA",476
52 DATA "FAAAEAHPPA",511
53 DATA ""
100 IFT<>19827THENPRINT"DATA ERROR, CHECK PROGRAM":END
110 PRINTCHR$(147)"DATA OK, PRESS
RETURN!":PRINT:PRINT:PRINT"SYS49152"CHR$(19)
120 END
200 A=ASC(LEFT$(A$,1))-65:A$=RIGHT$(A$,LEN(A$)-1):RETURN
```

3.5 Midiloader for DATEL-compatible interface (Seil/JMS)

```
1 AD=49150:J=0:T=0:L=30
5 RT=0:READA$:IFA$=""THEN100
10 FORI=1TOLEN(A$)/2
15 GOSUB200:B=A:GOSUB200:A=B*16+A
25 POKEAD+J,A:J=J+1:RT=RT+A:NEXT:T=T+RT
27 READA:IFRT<>ATHENPRINT"DATA ERROR IN LINE "L:END
29 L=L+1:GOTO5
30 DATA "AAMAHIKJADCAKGMA",874
31 DATA "KJBGCAKGMAKJABKC",913
32 DATA "AIIFCNIGCOKNCANA",779
33 DATA "EIKJAAINCANAKCAI",792
34 DATA "CAEOMANNKKMANAPG",1339
35 DATA "MKBAPFKAAAOOCANA",1101
36 DATA "CAEOMADAEDAKAKAK",447
37 DATA "AKINECMACAEOMADA",759
38 DATA "CPCJAPAJAAJBCNOG",532
39 DATA "CNNAACOGCOEMCLMA",842
40 DATA "KJHPINAANMKNABNM",1051
41 DATA "BABEKNAGNOINGDMA",869
42 DATA "CJHANAAKKJAACJAB",582
```

```
43 DATA "PAOLKNAHNOGAGIGI",1181
44 DATA "KCAACAJKMAEMHPMA",935
45 DATA "MJPHNAPEKCAICAJK",1256
46 DATA "MAKFCNKGCOIFKOIG",1055
47 DATA "KPGIINCANAFIGAEF",913
48 DATA "FCFCEPFCCBANAAEP",450
49 DATA "ELCBANAALNINMAPA",883
50 DATA "AGCANCPPOINAPFGA",1284
51 DATA "INAENOGAAIAAABAA",472
52 DATA "FAAAEAHPPA",511
53 DATA ""
100 IFT<>19820THENPRINT"DATA ERROR, CHECK PROGRAM":END
110 PRINTCHR$(147)"DATA OK, PRESS
RETURN!":PRINT:PRINT:PRINT"SYS49152"CHR$(19)
120 END
200 A=ASC(LEFT$(A$,1))-65:A$=RIGHT$(A$,LEN(A$)-1):RETURN
```

3.6 Midiloader for SEQUENTIAL-compatible interface

```
1 AD=49150:J=0:T=0:L=30
5 RT=0:READA$:IFA$=""THEN100
10 FORI=1TOLEN(A$)/2
15 GOSUB200:B=A:GOSUB200:A=B*16+A
25 POKEAD+J,A:J=J+1:RT=RT+A:NEXT:T=T+RT
27 READA:IFRT<>ATHENPRINT"DATA ERROR IN LINE "L:END
29 L=L+1:GOTO5
30 DATA "AAMAHIKJADCAKGMA",874
31 DATA "KJBFCAKGMAKJABKC",912
32 DATA "AIIFCNIGCOKNCANA",779
33 DATA "EIKJAAINCANAKCAI",792
34 DATA "CAEOMANNKKMANAPG",1339
35 DATA "MKBAPFKAAAOOCANA",1101
36 DATA "CAEOMADAEDAKAKAK",447
37 DATA "AKINECMACAEOMADA",759
38 DATA "CPCJAPAJAAJBCNOG",532
39 DATA "CNNAACOGCOEMCLMA",842
40 DATA "KJHPINAANMKNABNM",1051
41 DATA "BABEKNACNOINGDMA",865
42 DATA "CJHANAAKKJAACJAB",582
43 DATA "PAOLKNADNOGAGIGI",1177
44 DATA "KCAACAJKMAEMHPMA",935
45 DATA "MJPHNAPEKCAICAJK",1256
46 DATA "MAKFCNKGCOIFKOIG",1055
47 DATA "KPGIINCANAFIGAEF",913
48 DATA "FCFCEPFCCBANAAEP",450
49 DATA "ELCBANAALNINMAPA",883
50 DATA "AGCANCPPOINAPFGA",1284
51 DATA "INAANOGAAIAAABAA",468
52 DATA "FAAAEAHPPA",511
53 DATA ""
100 IFT<>19807THENPRINT"DATA ERROR, CHECK PROGRAM":END
110 PRINTCHR$(147)"DATA OK, PRESS
RETURN!":PRINT:PRINT:PRINT"SYS49152"CHR$(19)
120 END
200 A=ASC(LEFT$(A$,1))-65:A$=RIGHT$(A$,LEN(A$)-1):RETURN
```

4. Concepts

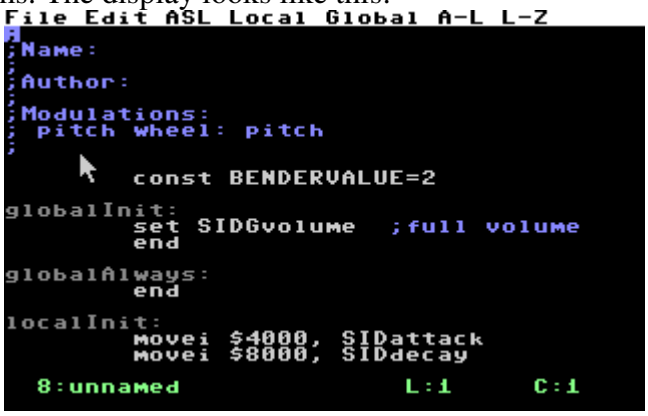
Sounds are created by writing programs that can read midi controllers, act on keypresses and control the sound-chip. Programs has to be compiled before they can be "run" for speed

reasons. When a sound program is running the C64 will act like a synth module to the outside world.

Sound programs are written, compiled and run in M64. Another program, The Performance Editor (under construction) will read precompiled sounds and store them in banks. Banks contain several sounds and the user can change sound via MIDI from a sequencer or synthesizer.

5. The Editor

When you start M64 you will see the edit screen. This is where you write, compile, run, load and save ASL programs. The display looks like this:



5.1 Editing

The editing is somewhat different from what you are used to if you use CBM BASIC.

First of all when you press keys you *insert* letters into the text instead of overwriting like CBM BASIC. Some keys have different meanings in the editor. Those are:

- 'Left-Arrow' is tabulator key.
- 'F1' is page up.
- 'F2' is home. (home of document)
- 'F3' is the compile key.
- 'F5' is the run key.
- 'F7' is page down.
- 'F8' is end. (end of document)
- Commodore graphic characters has been disabled (these are used to invoke commands).

5.2 The Menubar

The top row on the display is the menu bar. Use your mouse or joystick to access it (just like GEOS). Don't worry, everything can be controlled with the keyboard (you just have to learn all the keycodes).

Pull down the File menu and you will see what it contains. The letters to the right tells you that holding the Commodore key and pressing that letter is equivalent to accessing the menu bar.

5.3 The File menu

New

The "New" command can also be invoked by pressing CLR on the keyboard. The current ASL program will be erased and leave an empty one.

Open

The "Open" command will request for a file to open. The current ASL program will be erased and the selected one will be read.

Open as MIDI

The "Open as MIDI" command will wait for a MIDI SYSEX file to be sent. This way you can read programs from other devices outside the c64 world. The current ASL program will be erased and a new one will be read from MIDI.

Save

The "Save" command will rewrite the current ASL program to the current device without asking for a filename. If the file exists you will NOT be asked if you wish to remove that file.

Save as

The "Save as" command will request for a file to store. The current ASL program will be stored as the selected file. If the file exists you will be asked if you wish to remove that file. *You will NOT have to add "@:" in the filename.*

Save as MIDI

The "Save as MIDI" command will immediately send the current ASL program as MIDI SYSEX. Use "Open as MIDI" to read it back. This way you can save programs to other devices outside the c64 world.

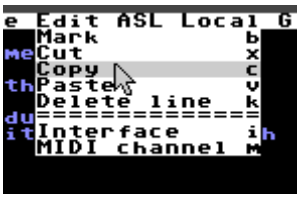
About

The "About" command will display version and registration information.

Quit

The "Quit" command will get you back to BASIC. As long as the ASL program hasn't been corrupted you can rerun M64 without erasing your work. This is pretty handy if you want to make some calculations or scratch some files without losing your work.

5.4 The Edit menu



Mark

The "Mark" command will toggle marking mode. You have to mark text in order to use "Cut" and "Copy".

Cut

The "Cut" command will move the marked text into the copy buffer. You might get confused because the display will be scrolled in order to place the cursor on the top line (for technical reasons). This little bug will hopefully be removed in future versions.

Copy

The "Copy" command will copy the marked text into the copy buffer.

Paste

The "Paste" command will paste the copy buffer into the current ASL program before the current line.

Delete line

The "Delete line" command will delete the current line.

Interface

The "Interface" command will let you manually select which type of MIDI interface you use. A dialog will open with three buttons marked "Sequential", "Datel", and "Passport". Use mouse or joystick to select or press "S", "D" or "P".

M64 will autodetect which interface you have but in case it fails (for some strange reason) you can change it manually here. Note that this information is just stored in memory and will be forgotten each time you restart the editor. If you have problems with the autodetect, contact the author.

MIDI channel

The "MIDI channel" command will let you select which MIDI channel M64 will receive messages on. A dialog will open with four buttons: use, cancel, up and down. Use mouse or joystick to select or press <cursor-up>, <cursor-down> to change channel, return to use the setting or stop to revert to the old setting.

5.5 The ASL menu

Compile

The "Compile" command will parse the ASL program and generate ASL code that can be executed with the "Run" command.

Run

The "Run" command will run the last successfully compiled ASL code.

5.6 The Local menu

The "Local" menu contains all default local register names. Selecting a name will paste it into the ASL program at the current cursor position like if you wrote it.

5.7 The Global menu

The "Global" menu contains all default global register names. Selecting a name will paste it into the ASL program at the current cursor position like if you wrote it.

5.8 The A-L menu

The "A-L" menu contains instruction names starting with letters from A to L. Selecting a name will paste it into the ASL program at the current cursor position like if you wrote it.

5.9 The L-Z menu

The "L-Z" menu contains instruction names starting with letters from L to Z. Selecting a name will paste it into the ASL program at the current cursor position like if you wrote it.

5.10 The status line

The status line at the bottom of the display shows the device and filename of the current ASL program. A '*' before the filename shows that the ASL program has been changed since it was last saved.

The status line also shows the current row and column of the cursor.

5.11 The error line

The error line is located just below the status line. It shows the last error from the compiler.

6. The File Requester

When opening or writing files from devices other than MIDI, M64 provides a file requester.

6.1 Editing

The tabulator key (arrow left, remember?) cycles through editing filename, filetype, device and drive.

To edit filetype press 'P' for PRG, 'S' for SEQ and 'U' for USR file.

To edit device, drive and filename just move cursor with cursor left/right and edit as usual. Pressing return while editing device or drive will reread the directory. Press return while editing filename or filetype when finished. Pressing STOP cancels the requester.

6.2 Select using keyboard

Use cursor keys to move up and down in the directory. Press return to select the highlighted file. Press STOP to cancel. Pressing 'F1' and 'F7' will scroll up and down respectively one page. 'F2' and 'F8' can be pressed to jump to first/last filename.

6.3 Select using mouse or joystick

Click on a filename to select it. Click on the arrow icons (at top and bottom of file list) to scroll one page up/down. Click on the OPEN/SAVE button when you are satisfied or CANCEL to cancel the requester.

7. Getting started

7.1 Opening a file

ASL programs can be loaded either from a standard Commodore device or from MIDI.

To load a file from Commodore device select File/Open from the menu or press <commodore-o>. A file requester will appear. Refer to the File Requester section for instructions how to use it.

To load a file from MIDI, select File/Open as MIDI from the menu or press <commodore-d>. A window will appear, telling you to start sending the file. Note that the sender's MIDI OUT must be connected to C64 MIDI IN in order to make this work. Send the file from the sequencer, synthesizer or whatever you are using. If all is well the border will flash and the file will be loaded.

7.2 Compiling

Select ASL/Compile from the menu or press F3 to compile the current ASL program. A window will appear that will inform you about any errors in the ASL program. When the program has been compiled press any key (except Restore) to return to the editor. If there was an error in the compile the cursor will be placed near the place where the error was detected.

7.3 Configuring

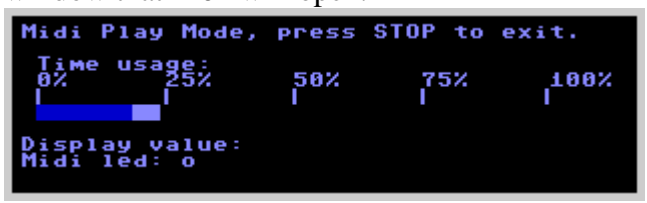
Make sure that C64 MIDI IN is connected to your master synthesizer's or computer's MIDI OUT.

Press <commodore-m> or select "Edit/MIDI channel" from the menu and make sure that M64 is configured to receive on the correct MIDI channel.

Press <commodore-i> or select "Edit/Interface" from the menu and make sure that your interface has been selected.

7.4 Running

Select ASL/Run from the menu or press F5 to run the last compiled program. The picture below shows the play window that M64 will open.



When the ASL code is running M64 is ready to receive MIDI messages. Pressing keys on your master keyboard should produce sound. The Midi led indicator will be lit if M64 receives MIDI messages on *any* channel. M64 will ignore all messages except those sent on the previously selected MIDI channel.

The time usage indicator shows the CPU-usage. If time usage exceeds 100% the sound will slow down (that is, always-routines will not run at the correct frequency) and the word "OVERFLOW" will appear.

Pressing space will toggle screen blanking on/off. Screen blanking reduces noise. When the screen is blanked, a red screen color will indicate overflow. A C128 running in C64 mode will operate at 2MHz when the screen is blanked!

8. Programming Abstract SID Language

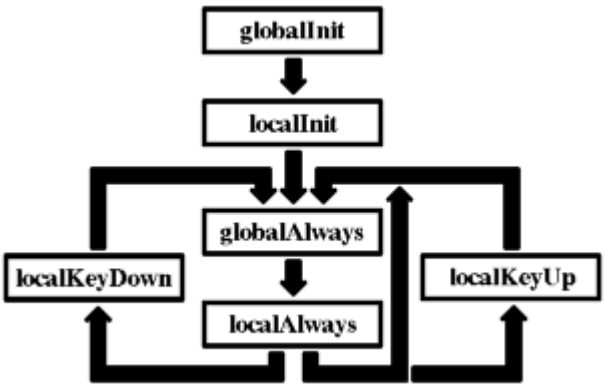
When programming ASL some understanding of how the SID chip works is recommended. Consult your c64 user manual if necessary.

An ASL program consists of 6 main sections. These are called:

- globalInit
- globalAlways

- localInit
- localAlways
- localKeyDown
- localKeyUp

M64 runs ASL code as follows.



First, the globalInit routine is executed followed by one call to localInit for each channel of the SID. Now globalAlways and localAlways (for each channel) are executed frequently (50Hz if not specified). If a keydown message is sent from the master via MIDI then localKeyDown for the unused channel is executed. If a keyup message is sent then localKeyUp for the allocated channel is executed.

As we will see later, it is possible to specify exactly which code that should be executed for each channel.

To sum up, the init-routines are ideal for setting up. Always-routines should be used for handling effects like vibrato, slides and other things that need to update frequently. Keydown-routines should contain code needed to start the sound. Keyup-routines should contain code that stops the sound or fades it. Local-routines affect the channel specific things while global-routines affect everything that is not (the filter for example affects the whole SID chip, not just a channel and is therefore something that should be handled by globalInit and globalAlways).

8.1 Numeric values

Numeric values can be either decimal, hexadecimal or binary. If you have programmed 6502 assembler you will be happy because it is the same syntax in ASL.

- Decimal values are just digits. Example: 56 or 987
- Hexadecimal values are a dollar character followed by digits. Example: $20 or $fa3b
- Binary values are a procent character followed by digits. Example: %1100110100 or %11

8.2 Constants

A constant declaration has the form:

```
const <identifier>=<expression>[,<identifier>=<expression>,[...]]
```

A constant declaration looks like this:

```
    const TIME=50
```

This will assign the value 50 to the identifier "time". It is possible to declare several constants on one line like this:

```
    const TIME=50, DELAY=14
```

After a constant declaration it is possible to use that identifier in expressions. An example:

```
    const LENGTH=50
    const POSTLUDE=LENGTH-5
```

You should use upper case characters for constants to make it easy to differ them from labels and registers.

8.3 Expressions

Expressions may contain constants, numeric values. Possible operations are plus, minus and unary minus.

Here are some examples of a valid expressions (if TIME previously has been declared as a constant):

```
    -TIME+100
    TIME--$ff
    %100-$100
    -$1+-%0
```

8.4 Labels

Labels are names on lines in the ASL program. Labels are declared by writing letters at the beginning of a line (optionally followed by a colon). An example:

```
    set SIDgate
foo addi 1,a
    cmp a,b
    bne foo
    end
```

In this example "foo" is a label. Here is another example of valid labels:

```
    set SIDgate
foo addi 1,a
q:  cmp a,b
    bne foo
yeah
    end
```

Here "foo", "q" and "yeah" are valid labels.

The ASL compiler needs to know where certain routines are located and therefore there are some labels that just has to exist in every ASL program. We get back to those labels later.

8.5 Registers

Registers are used to hold values. Registers are 16 bit, which means that each register can hold values from 0 to 65535. Registers wrap around if you try to exceed the limits (if you add 1 to a register containing 65535 the result will be 0).

> *Note: Due to the fact that registers wrap around, you can also say that registers can hold values from -32768 to 32767. The values from -32768 to -1 are actually exactly the same as 32768 to 65535. You can thus add -1 to 50 and you get 49 (the same for 65535+50).*

There are two types of registers, local and global. In order to use registers they must be declared. Some registers are predeclared.

8.6 Global registers

Global registers can be declared with the global directive as follows.

```
global [<registername>[,<registername>[,...]]]
```

These are the predeclared global registers:

MIDGexpression
: Expression controller value.

MIDGmodWheel
: Modulation wheel controller value.

MIDGpan
: Panoration controller value.

MIDGpitch
: Pitch wheel value.

MIDGsustain
: Sustain pedal controller value.

MIDGvolume
: Volume controller value.

SIDGdisable3
: If set, channel 3 of the SID will be disabled (used for testing).

SIDGfilterBP
: Selects band-pass filter type.

SIDGfilterExt
: If set, SID will filter an external signal (connected to pin 5 of the monitor jack).

SIDGfilterFreq
: Filter cutoff frequency. The exact frequency can be calculated with the following formula: FREQUENCY = (SIDfreq * 0.183) + 30Hz.

SIDGfilterHP
: Selects high-pass filter type.

SIDGfilterLP
: Selects low-pass filter type.

SIDGfilterReso
: Filter resonance. Setting this register to a large value will peak the volume of frequencies nearest the cutoff.

SIDGmonophonic
: If set, M64 will execute localKeyDown for all channels at once when a key is pressed. If cleared (the default), M64 will only execute localKeyDown for an unused channel.

SIDGvolume
: SID volume.

Registers starting with 'SIDG' are registers that will affect the SID chip.

Registers starting with 'MIDG' contain values of the controllers that has been sent via MIDI from the master synthesizer or computer.

All global registers contain values relevant to the whole SID chip.

8.7 Local registers

Local registers can be declared with the local directive as follows.

```
local [<registername>[,<registername>[,...]]]
```

These are the predeclared local registers:

MIDaftertouch
: Aftertouch (key/channel pressure) controller value.

MIDfreq
: Frequency of the last pressed key.

MIDkey
: The actual key pressed (0..127).

MIDvelocity
: Set to the key down velocity when a key is pressed and to key up velocity when key is released.

SIDattack
: Attack time.

SIDdecay
: Decay time.

SIDdisable
: If set, the channel will be disabled (mainly for testing).

SIDfilter
: If set, the filter will be applied to the channel.

SIDfreq
: Frequency control. The exact frequency can be calculated with the following formula: FREQUENCY = (SIDfreq * CLOCK / 16777216) Hz where CLOCK equals the system clock frequency, 1022730 for American (NTSC) systems, 985250 for European (PAL).

SIDgate
: Gate on/off. Set to start attack cycle and clear to start release.

SIDmodulate
: Ring modulation on/off. Set to enable modulation.

SIDnoise
: Noise waveform on/off. Set to enable.

SIDpulse
: Pulse waveform on/off. Set to enable.

SIDpulseWidth
: Pulse width.

SIDrelease
: Release time.

SIDsaw
: Sawtooth waveform on/off. Set to enable.

SIDsustain

Sustain level.

SIDsync

Syncronize on/off. Set to enable.

SIDtraingle

Triangle waveform on/off. Set to enable.

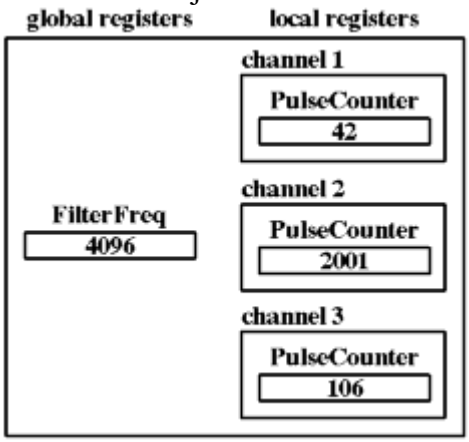
The registers starting with 'SID' are registers that will affect the SID chip. Note that all SID registers are 16 bit, even the "boolean" ones! This means that all registers are treated equally and you don't need to know how many bits each register oThe registers starting with 'SID' are registers that will affect the SID chip. Note that all SID registers are 16 bit, even the "boolean" ones! This means that all registers are treated equally and you don't need to know how many bits each register occupy. 65535 is always maximum and 0 is always minimum.

Registers starting with 'MID' contain values that has been sent via MIDI from the master synthesizer or computer.

> *All local registers contain values relevant to each of the three channels in the SID chip.*

8.8 Register handling

Assume that we have written an ASL program that declares one local register, PulseCounter, and a global register, FilterFreq. As you can see from the diagram below, there are actually three versions of the local register, one for each channel. Each version can hold a unique value. The diagram also shows that there is just one version of the global register.



Let's say that the routine localAlways2 displays the PulseCounter, then "$07D1" will be shown in the run window (2001 hexadecimal, that is).

Take a look at the following code:

```
localInit:
        movei 666, PulseCounter
        end
```

The **movei** instruction moves a constant to a register. Since it is executed in the localInit routine the second argument is assumed to be a local register. If we replace PulseCounter with a global register, then the program will not work as expected. *NOTE: The current version of the ASL compiler will not report this type of error.* To read and write to global registers from local routines you should use the **glob2reg** and **reg2glob** instructions.

Another example:

```
globalAlways:
        move FilterFreq, SIDGfilterFreq
        end
```

Here, the **move** instruction requires two global registers as arguments since this is a global routine. Any local registers as arguments would yield unpredictable results.

8.9 Instructions

Instructions have the form:

```
    <instruction> [<argument>[,<argument>[...]]]
```

There are three types of arguments: registers (reg), expressions (expr) and labels (lbl).

Below is a list of all instructions and their arguments.

Debug instructions:

display <reg>
: Display value of reg on screen (hexadecimal). This is for debugging ASL programs only. This instruction does nothing in The Performance Player.

Control instructions:

execute <expr>,<reg>
: Execute 6502 machine language code at address expr and set 6502 register x and a to high/low value of register reg. The value of reg will be changed to what x and a contains after the execution of the ML routine. This instruction should not be used by anyone who doesn't know exectly what they are doing. Memory area $CE00-$D000 is reserved for user defined ML programs to be executed with this command. Typical use could be to trigger other audio hardware or calculate complex functions.

settimer <reg>
: Changes the interval by which the always-routines are executed. This is the value written to the CIA timer. The value can be calculated as follows:
SPEED_VALUE=TIME/CLOCK_SPEED
where CLOCK_SPEED is 1022730 for American (NTSC) monitors and 985250 for European (PAL) monitors. TIME is the interval between always-updates in seconds. If your ASL-programs always get OVERFLOW, then increase this value.

settimeri <expr>
: Same as settimer but takes an expression as first argument.

end
: Execution stops here. This instruction ends an ASL routine.

mhz <reg>
: The value of reg is set to the current processor speed in MHz. Currently, only the SuperCPU unit will be detected.

palntsc <reg>
: The value of reg is set to 1 if the current system is a European (PAL) system, else 0 (American NTSC).

Compare and branch instructions:

cmp <reg1>,<reg2>
Compares reg1 with reg2. To be used before most branch instructions.
cmpi <expr>,<reg>
Compares expr with reg. To be used before most branch instructions.
beqz <reg>,<lbl>
Jumps to lbl if value of reg is zero. *NOTE: Abbrevation for Branch-if-EQual-to-Zero.*
bnez <reg>,<lbl>
Jumps to lbl if value of reg is not zero. *NOTE: Abbrevation for Branch-if-Not-Equal-to-Zero.*
bra <lbl>
Jumps to lbl unconditionally. *NOTE: Abbrevation for BRanch-Always.*
bhi <lbl>
Jumps to lbl if the value of the right operand was higher than the value of the left operand in the latest compare. *NOTE: Abbrevation for Branch-if-Higher-than.*
beq <lbl>
Jumps to lbl if the value of the right operand was equal to the value of the left operand in the latest compare. *NOTE: Abbrevation for Branch-if-EQual.*
bne <lbl>
Jumps to lbl if the value of the right operand was not equal to the value of the left operand in the latest compare. *NOTE: Abbrevation for Branch-if-Not-Equal.*
blo <lbl>
Jumps to lbl if the value of the right operand was lower than the value of the left operand in the latest compare. *NOTE: Abbrevation for Branch-if-LOwer-than.*
bhs <lbl>
Jumps to lbl if the value of the right operand was higher than or equal to the value of the left operand in the latest compare. *NOTE: Abbrevation for Branch-if-Higher-or-Same.*
bls <lbl>
Jumps to lbl if the value of the right operand was lower than or equal to the value of the left operand in the latest compare. *NOTE: Abbrevation for Branch-if-Lower-or-Same.*

Register manipulation:

move <reg1>,<reg2>
Move the value of reg1 to reg2.
movei <expr>,<reg>
Move the value of expr to reg.
set <reg>
sets all bits in reg. Equivalent to: movei 65535,reg
clr <reg>
clears all bits in reg. Equivalent to movei 0,reg
reg2glob <lreg>,<greg>
moves result of local register lreg to global register greg. Used in local routines to change global registers. *WARNING: Do not use this instruction in global routines!!!*
glob2reg <greg>,<lreg>
moves result of global register greg to local register lreg. Used in local routines to access global registers (normally reading MIDI controllers). *WARNING: Do not use this instruction in global routines!!!*

Arithmetic instructions:

add <reg1>,<reg2>
Adds the value of reg1 to reg2 and stores the result in reg2.
addi <expr>,<reg>
Adds the value of expr to reg and stores the result in reg.
sub <reg1>,<reg2>
Subtracts the value of reg1 from reg2 and stores the result in reg2.
subi <expr>,<reg>
Subtracts the value of expr from reg and stores the result in reg.
scale <reg1>,<reg2>
scales the value of reg2 so that max is the value of reg1 and min is zero. That is: reg2=(reg2/65535)*reg1. Very useful if you want to scale down a sinus value for example.
scalei <expr>,<reg>
scales the value of reg so that max is the value of expr and min is zero. See scale.
shiftli <expr>,<reg>
shifts reg expr times to the left. That is: reg=reg*(2^expr).
shiftri <expr>,<reg>
shifts reg expr times to the right. That is: reg=int(reg/(2^expr)).
ashiftri <expr>,<reg>
shifts reg expr times to the right. The difference between ashiftri and shiftri is that ashiftri shifts negative numbers correctly.
max <reg1>,<reg2>
reg2 is set to the largest value of reg1 and reg2. Good for limiting counters.
maxi <expr>,<reg>
reg is set to the largest value of reg and expr.
min <reg1>,<reg2>
reg2 is set to the smallest value of reg1 and reg2. Good for limiting counters.
mini <expr>,<reg>
reg is set to the smallest value of reg and expr.

Other instructions:

sinus <reg1>,<reg2>
calculates a sinus value from value of reg1 and stores it in reg2. That is: reg2=sinus(reg1). Sinus does not use radians or degrees. To get a full cycle just change reg1 from 0 to 65535! The amplitude is 32767 and the offset is 32767 so the real formula is (ignore it):
reg2=32767+32767*sin(reg2/65535*2*pi)
triangle <reg1>,<reg2>
calculates a triangle value (similar to sinus) from value of reg1 and stores it in reg2. To get a full cycle just change reg1 from 0 to 65535!
random <reg>
reg is set to a random number.
note2freq <reg1>,<reg2>
reg2 is set to the frequency value of the note value in reg1. Useful if you want to play other notes than the expected. Note values are between 0 and 127.
pitchbendi <expr>,<reg>
This is a special instruction that is made specially to handle pitchbend. It was created for speed reasons. Pitchbendi uses the MIDkey register and <reg> to bend the frequency expr semitones. <reg> holds the bender value ($8000 is no bend, $ffff is full bend forward and 0 is full bend backwards). The result is copied directly into the

MIDfreq register, so all you have to do is a "move MIDfreq,SIDfreq" to hear the results. If you wish to use this instruction for other purposes you can change the MIDkey and <reg> value before the call and then use MIDfreq to whatever you want. *WARNING: This instruction should not be used in global-routines! The result is undefined!*

8.10 Comments

Comments are everything to the right of a semicolon just like 6502 assembler.

Example:

```
;This is a comment
    local hello
    clr hello  ;This is also a comment
;   end        This line is a comment (no end here)
    end        ;but here!
```

8.11 Examples

Let's look at a simple ASL program example:

```
;
;Name: Dummy
;
;Author: Jonas Hulten
;
;Modulations:
; pitch wheel: pitch
;

    const BENDERVALUE=2

globalInit:
    set SIDGvolume          ;full volume
    end

globalAlways:
    end

localInit:
    movei $4000, SIDattack
    movei $8000, SIDdecay
    movei $D000, SIDsustain
    movei $9000, SIDrelease

    set SIDsaw
    end

localKeyDown:
    move MIDfreq,SIDfreq    ;play correct frequency
    set SIDgate             ;start attack
    end

localKeyUp:
    clr SIDgate             ;start release
    end

    local pitch
```

```
localAlways:
    ;pitchbend
    glob2reg MIDGpitch, pitch
    pitchbendi BENDERVALUE, pitch
    move MIDfreq, SIDfreq
    end
```

This program is the default ASL program that will show up in the editor when you start M64. It will produce a simple sawtooth sound that will sound like the bass in Spelunker or the melody in Burnin' Rubber. The pitch wheel can be used to change the pitch of the sound two semitones up or down.

The program runs like this: First of all it will initialize. The globalInit routine will set volume to maximum and localInit will set attack, decay and release time. It will also set sustain level and enable the sawtooth waveform. The values for attack, decay, sustain and release are deliberately written in hexadecimal form since only the first nibble really matters (if you don't know what this means, just forget it, it's not important).

When keys are pressed the localKeyDown routine will copy the correct frequency to the SID chip. The SID gate will also be set to start the sound. When keys are released the localKeyUp routine will clear the SID gate so that the release of the sound starts.

As the sound plays, the globalAlways routine will do nothing but the localAlways routine will make the pitchbend. This is done by first copying the global register MIDGpitch to a local one (pitch is declared just above the routine). Then the special instruction 'pitchbendi' does all the magic and updates local register MIDfreq with the correct frequency value. The last thing to do is to copy the frequency value to the SID chip.

To change the character of the sound we change the waveform to pulse instead of sawtooth. To make the sound constantly change we let the pulse width wary over time. To do so we place a counter in a register that will constantly count up and then use the 'triangle' or 'sinus' instruction to get the actual values to put into the SIDpulseWidth register. The program now look like this (all changes in italics).

```
;
;Name: Not so Dummy
;
;Author: Jonas Hulten
;
;Modulations:
; pitch wheel: pitch
;

    const BENDERVALUE=2
    const PULSESPEED=200

globalInit:
    set SIDGvolume          ;full volume
    end

globalAlways:
    end

    local pulseCount
```

```
localInit:
    movei $4000, SIDattack
    movei $8000, SIDdecay
    movei $D000, SIDsustain
    movei $9000, SIDrelease

    set SIDpulse
    clr pulseCount
    end

localKeyDown:
    move MIDfreq,SIDfreq    ;play correct frequency
    set SIDgate             ;start attack
    end

localKeyUp:
    clr SIDgate             ;start release
    end

    local pitch

localAlways:
    ;pitchbend
    glob2reg MIDGpitch, pitch
    pitchbendi BENDERVALUE, pitch
    move MIDfreq, SIDfreq

    addi PULSESPEED, pulseCount
    triangle pulseCount, SIDpulseWidth
    end
```

Note that there are one 'pulseCount' register for each channel. All three are syncronized so if you press three keys at the same time you can hear that the pulse width is the same for all three channels. We will now make them different from each other. We do so by making three different localInit routines, one for each channel! Since just the pulseCount will be different we will reuse our code. The localInit routine now looks like this:

```
localInit1:
    clr pulseCount
    bra init
localInit2:
    movei 20000, pulseCount
    bra init
localInit3:
    movei 40000, pulseCount
init:
    movei $4000, SIDattack
    movei $8000, SIDdecay
    movei $D000, SIDsustain
    movei $9000, SIDrelease

    set SIDpulse
    end
```

Note that the original localInit label has been removed. That is because otherwise it would override the other localInit labels.

Ok, let's look at another example. This time it's a monophonic sound that uses all three channels for one sound. This sound uses the syncronize feature of the SID chip to make an

odd sound. We will enable sync on channel 1 which will syncronize the fundamental frequency of channel 1 with the fundamental frequency of channel 3, producing "hard sync" effects. We modulate the frequency of channel 1 with the pitch wheel (the resolution of the pitch wheel is much higher that the modulation wheel). Channel 3's frequency will be the frequency of the key we press. Here is the program:

```
;
;Name: Monophonic sync
;
;Author: Jonas Hulten
;
;Modulations:
; modulation wheel: channel 1 frequency
;

globalInit:
    set SIDGvolume          ;full volume
    set MIDGmonophonic
    end

globalAlways:
    end

localInit1:
    movei $4000, SIDattack
    movei $8000, SIDdecay
    movei $D000, SIDsustain
    movei $9000, SIDrelease
    set SIDsaw
    set SIDsync
    end
localInit2:
localInit3:
    end

localKeyDown1:
    set SIDgate             ;start attack
    end
localKeyDown2:
    end
localKeyDown3:
    move MIDfreq,SIDfreq    ;play correct frequency
    end

localKeyUp1:
    clr SIDgate             ;start release
    end
localKeyUp2:
localKeyUp3:
    end

    local mod

localAlways1:
    glob2reg MIDGpitch, mod
    move mod, SIDfreq
    end
localAlways2:
localAlways3:
    end
```

To make monophonic sounds we simply added a line in the globalInit that says '**set** MIDGmonophonic'.

We can try the ring modulation by changing the waveform to triangle and setting SIDmodulate instead of SIDsync. localInit1 will look like this:

```
localInit1:
    movei $4000, SIDattack
    movei $8000, SIDdecay
    movei $D000, SIDsustain
    movei $9000, SIDrelease
    set SIDtriangle
    set SIDmodulate
    end
```

You might just feel like Jeff Minter while playing around with this sound.

8.12 Trix and Tips

There are a couple of techniques that are very useful when programming ASL.

Constants

Use constants instead of numbers so that people can adjust the sound parameters without reading the whole code.

```
    const VIBRATOSPEED=50
;     ...
    addi VIBRATOSPEED, Counter
```

is MUCH better than:

```
    addi 50, Counter
```

Limiting

The following example shows how to use the 'cmp' instruction in combination with branch instructions to make a counter count from 0 to 4 and then stop counting.

```
    cmpi 4, Counter  ;compare Counter with 4.
    bhs noCount      ;branch to noCount if Counter was
                           ;higher or same as 4.
    addi 1, Counter  ;increment Counter.
noCount:
```

This can be done in a more efficient way:

```
    addi 1, Counter  ;increment Counter
    mini 4, Counter  ;let Counter be the lowest of 4 and Counter
```

Vibrato

The following code shows how to add vibrato to a register.

```
    const SPEED=200, AMPLITUDE=40 ;first some declarations
    local Counter, SinValue

;... some code here ...
```

```
    addi SPEED,Counter            ;update Counter
    sinus Counter, SinValue       ;get sinus value (0..65535)
    scalei AMPLITUDE+AMPLITUDE, SinValue ;scale down sinus value (0..80)
    subi AMPLITUDE, SinValue      ;change offset (-40..40)
    add SinValue, SIDfreq         ;add vibrato to SID frequency!

;... and some code here too ...
```

Pitchbend

Making pitchbend with ASL can be done without the special 'pitchbend' instruction but that is not recommended. This instruction does more than it's operands tell you:

The following code will demonstrate the typical use.

```
    const BENDERVALUE=12         ;pitchbend a full octave (12 semitones)
    local pitch                  ;first some declarations

;... some code here ...

localAlways:
    glob2reg MIDGpitch, pitch    ;copy global register to local
    pitchbendi BENDERVALUE, pitch
                                  ;the result is an updated MIDfreq
register
    move MIDfreq, SIDfreq        ;so just copy it to the SID!

;... and some code here too ...
```

8.13 Troubleshooting

Finding errors that are not compile errors are generally hard (especially if you are inexperienced). The most common errors when programming ASL are:

M64 stops completely when I run my script!

Probably because you forgot to end some routine with the 'end' instruction. It can also be an 'execute' instruction that expects some files to be loaded before starting M64.

I try to modulate pulse width with the modulation wheel but I can't get it to work!

The SIDGmodWheel is a global register. Use glob2reg in local routines to copy it to a local register (like SIDpulseWidth).

Nothing happens when I tap on the MIDI keyboard!

Make sure that MIDI OUT on the keyboard or sequencer is connected to MIDI IN on the c64 MIDI interface.

Check that the keyboard is sending on the correct channel. The receive channel can be changed by pressing <commodore-m> or selecting "Edit/MIDI channel" from the menu (the default is channel 1).

Also, check that the correct MIDI interface is selected. Press <commodore-i> in the editor or select Edit/Interface in the menu.

When running, I get OVERFLOW whenever I press a key!

Your sound is too complex to be updated at the current frequency. Use the 'settimeri' command to increase the update interval time.

Registers suddenly get strange values!

Make sure that there are no references to global registers in local routines (except in 'glob2reg' and 'reg2glob' instructions). Also make sure that there are no local registers at all in global routines.

9. Bugs

M64 should be pretty bug-free. However, if M64 suddenly jams you should hold Run/Stop and tap the Restore key. If that didn't got you back to a BASIC prompt you have to make a soft reset by hand (this is not possible with a standard C64 without a reset switch).

Now, at the BASIC prompt, type SYS2061 and press return. You will hopefully be back in M64 with your previous ASL program. If M64 didn't start then repeat the Run/Stop-Restore procedure and instead of starting it with SYS2061, you reload M64 from your storage device and run it. If your ASL program has not been destroyed M64 will let you continue editing it.

Send bug reports directly to me, the author, at the following address:
bjonte@hem2.passagen.se.

10. Future

Future versions of M64 will (hopefully) feature:

- Multiple SID-chip support (SuperCPU will be recommended)
- Performance Player (create banks with precompiled ASL programs and change sounds via MIDI)

11. Registration

This program was shareware but since I haven't updated the program for half a year I have decided to release it as freeware! Those who have registered will get the Performance Player for free when it is released and others will have to pay a small sum of money for it.

12. Copyright

13. AnyWare Designs

AnyWare Designs are Jonas Hultén (author of this program) and Petrus Hyvönen.

14. Credits

M64 was written by Jonas Hultén.

Thanks to

- Ola Andersson for beta-testing.
- Petrus Hyvönen and Åsa Nyberg for ideas and comments.
- Daniel Sevo for raytracing the MIDI cable on his much faster Amiga.

M64 was developed using the following equipment.

Hardware:

- Commodore 64
- 1541-II disk drive
- Action Replay MK IV
- SuperCPU-64
- Sentech MIDI-interface
- Amiga500

Software:

- DAsm crossassembler
- PRLink transfer utility
- FrexxED for Amiga

$VER: M64_manual 1.1.2 (10.1.98)
Created by Jonas Hultén.
Email: bjonte@hem2.passagen.se

URL: http://www.algonet.se/~bjonte/AnyWare/M64/M64.html