

# SoftDin 7



**"Dream holiday"**

**Vice snapshot with Vice palette**

**Made with the GIMP from a photo  
and converted to C64 160x200  
Multicolor Mode Bitmap  
by Stefano Tognon  
in 2004**

**"The big of small"**

...



**Free Software Group**

OSin 7  
version 1.00  
9 January 2004

## General Index

Editorials.....	4
News.....	5
CGSC v1.11.....	5
SIDPlayer 4.4.....	5
SID PC64 Winamp plugin v2.0.....	6
Sidplay2/w.....	7
HVSC 5.7.....	7
Asterion Sid-Tracker v1.1.....	9
reSID 0.16.....	9
Sidplay2 + ReSID 0.16.....	10
2010 SID COMPO V1.....	10
Libsidplay1 1.36.59.....	10
SIDPLAY for Mac OS X 3.3.....	10
CGSC v1.12.....	11
HVSC 5.8.....	11
Catweasel MK4.....	12
XSidplay 1.6.5.2.....	14
The SID Compo IV.....	14
SIDBrowser V2.4.....	14
Dustbin (Stefano Palmonari) Interview!.....	17
Inside Modules.....	20
Structure.....	20
Instrument.....	21
Pattern.....	21
Code.....	23
Conclusion.....	47
xxlarge.....	48
Code.....	48
Analysis.....	50
In depth.....	50
Conclusion.....	52

## Editorials

Stefano Tognon <[ice00@libero.it](mailto:ice00@libero.it)>

Hi, again.

As you can see there is no a Marble Madness article. But I think that if you downloaded the new HVSC update you will enjoy a rip it contains. However I have not yet finish my rip, but now there is no pleasure to finish as we can listen to the available rip.

So this time I will present another engine from the inside: Modules tune by Ivan Del Duca. I choose this tune primary as I like it and secondary as this is probably made with the first music engine written by an Italian programmer.

The second article is about a very tiny sid: 256 bytes for a fantastic sound!

Well, why not making a competition about writing very small sid? A mix about programming and sid composition...

Finally I like to remember to all the sid people around that if you want to contribute to this magazine with article it's very simple: write something about your sid music activity and it will be a pleasure to make it available for all.

Bye  
S.T.

## News

Some various news of players, programs , competitions and hardware:

- CGSC v1.11
- SIDPlayer 4.4
- SID PC64 Winamp plugin v2.0
- Sidplay2/w
- HVSC 5.7
- Asterion Sid-Tracker v1.1
- reSID 0.16
- Sidplay2 + ReSID 0.16
- 2010 SID COMPO V1
- Libsidplay 1.36.59
- SIDPLAY for Mac OS X 3.3
- CGSC v1.12
- HVSC 5.8
- Catweasel MK4
- XSidplay 1.6.5.2
- The SID compo IV
- SIDBrowser V2.4

### CGSC v1.11

Version 1.11 of the Compute's Gazette Sid Collection was released on 8 May 2004.

The collection now contains 5913 MUS files, 1314 STR files and 1567 WDS files.

Download the update from <http://www.c64music.co.uk>

### SIDPlayer 4.4

Released on 8 May 2004 the new version of Christian Bauer SIDPlayer (BeOS, Linux).

It this version it is fixed a problem with the wrong replay routine being called when the IRQ vector was changed. Mastercomposer tunes (and probably others) not play correctly.

Download the player from here: <http://www.uni-mainz.de/~bauec002/SPMain.html>

## SID PC64 Winamp plugin v2.0

SID PC64 is a Winamp plugin which plays .sid files on real C64 through a PC64 cable (<http://sta.c64.org/pc64.html>) developed by Jakub 'Prezes' Wozniakowski.

Features and limitations:

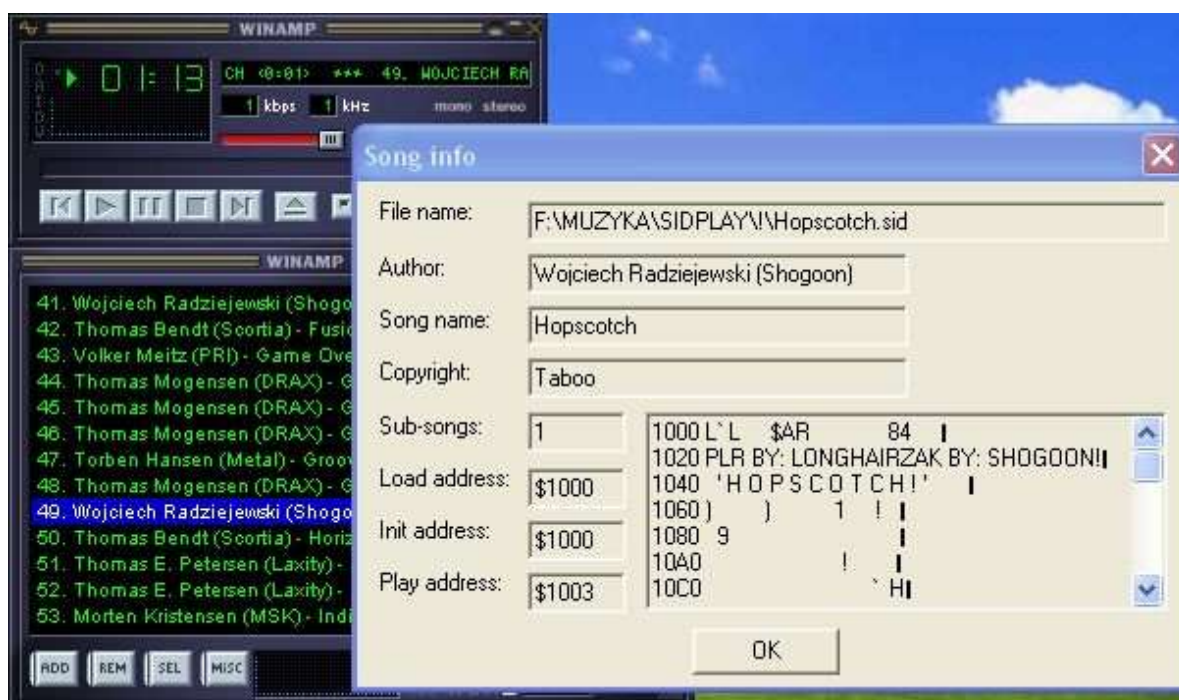
- Multispeed songs are supported
- Tunes which use samples are not supported
- Change playing sub-tune number moving Winamp's position bar (if disabled there is only one sub-song)
- The whole song must be transferred to C64 before playing, it can take few seconds - rasters on borders (c64), progress window (pc)
- Plugin and c64 can hang from time to time - not all tunes have been tested. This piece of player's code taken from HVSC will not work:

```
                                LDA #$00
                                STA kapusta
                                JSR $1000
kapusta                          NOP
                                ...
```

- Player on C64 is \$6a bytes long (including IRQ and simple transfer routines) so don't say it's lame ;-)
- Plugin was tested on Windows XP professional and Windows 2003 Server using Winamp 2.91, AMD Duron 1800

Version 2.0 (released on 16 may 2004) features:

- Windows XP/2000/2003 compatible
- detailed song info
- selectable parallel (LPT) port
- custom playing time
- improved stability
- transfer progress window



## Sidplay2/w

A new version of sidplay2/w was released on 27 May. These are the notable changes this time:

- support for multi-subtune BASIC tunes (libsidplay2)
- sprite stealing emulated (libsidplay2)
- p00 support (libsidplay2)
- stereo mus to sid conversion (libsidplay2)
- corrected instruction overlapping (libsidplay2)
- CPU Debug improvements - same memory model, progress meter, option to enter power-on delay value

Download it from <http://www.gsldata.se/c64/spw/>

## HVSC 5.7

HVSC Update #39 was released on 27 May and is available at [www.hvsc.c64.org](http://www.hvsc.c64.org)

After this update, the collection should contain 27,087 SID files!

This update features (all approximates):

- 1946 new SIDs
- 275 fixed/better rips
- 3 fixes of PlaySID/Sidplay1 specific SIDs
- 12 repeats/bad rips eliminated
- 230 SID credit fixes
- 470 SID model infos
- 41 tunes moved out of /DEMOS to their composers' directories
- 11 tunes from /DEMOS/UNKNOWN identified :-)
- 14 tunes moved out of /GAMES to their composers' directories

We're once more proud to present another monster update, consisting of overall over 2100 SID tunes, 275 of those being fixes. With this release we also finally managed to completely unload our huge queue of unreleased tunes to the public. There are still a couple of problem tunes left which we'll hopefully manage to fix for the next HVSC update, but basically that's all there is.

We've got the music compo entries from the following parties in this pack:

Breakpoint 2004, Sidwine 3, Floppy 2004, Forever 5 and 7D4

Further there are a lot of real, unmodified BASIC rips in this update:

They are basically RSID tunes, hence they need Sidplay2 or a real C64.  
To make them easier to spot we have added `_BASIC` to the filename.

Another bunch of classical tunes from the first minute have been added to HVSC:

- Peter Weighill's Master Composer collection /DEMOS/UNKNOWN/Master\_Composer/
- the Master Composer example tunes, most likely done by the author of the program, Paul Kleimeyer.
- the Brøderbund Music Shop tunes, uncredited, but with high likeliness also composed by the maker of the tool, Don Wilson.

Main Artists featured in this update:

- |   |                    |                      |
|---|--------------------|----------------------|
| • DRAX                                    | Yip                | Agemixer             |
| • Jan Albartus                            | Amadeus/Attic/Meka | Compod               |
| • Dwayne Bakewell & Kent Patfield (Pearl) | Richard Bayliss    | Glenn Rune Gallefoss |
| • Kristian Røstøen                        | Bzyk               | Comer                |
| • Cosowi                                  | Cycleburner        | Hein Holt            |
| • Jeroen Breebart                         | Jeff               | Klax                 |
| • Kordiaukis                              | Mitch & Dane       | Replay               |
| • Sad                                     | Shapie             | Surgeon              |
| • Taxim                                   |                    |                      |

NEW Artists featured in this update:

- |                        |                                 |                      |
|------------------------|---------------------------------|----------------------|
| • Android              | Commodore (BASIC classics)      | Paul Kleimeyer       |
| • Tonal Teapot         | 4wd-soft                        | Amadeus/Slash        |
| • David Bain           | Benno                           | Black Dove           |
| • Debby Cruz           | Freeze/The Force                | G-Fellow             |
| • Ewen Gillies         | David Green                     | Gummibeer            |
| • Krwiak               | Lio                             | Alexander Kern (Mac) |
| • Mr. Wig              | Stephan Parth (Nata)            | Rashka               |
| • Sam/SCCS             | Speedcracker                    | TheFatman            |
| • Steven Judd (Wyndex) | Tinnitus (Asterion & Trompkins) | Oskar Törnros        |
| • Steffen Wagner       | Nick Williams                   | Yoshi                |
| • Don Williams         |                                 |                      |

There are a couple of things worth mentioning in this update:

- The latest release of Sidplay2 not only contains some generic emulation improvements, but it also introduces the possibility to play back real BASIC tunes without the need to painfully convert them to machine code, which often resulted in rather doubtful playback quality. We ripped and re-ripped quite a lot of BASIC tunes for this update, most of which now reside in the /Commodore directory. There are the classic example tunes of the C64 users manual, some example tunes from "the bible", the C64 Programmer's Reference Guide. Also take a peek into the /GAMES dir of this update, there are a couple of classic BASIC tunes too, such as Sword of Fargoal, Telengard or the 1983 Stack Computer Services BASIC games! To make them easy to spot we added a \_BASIC suffix to all the filenames.
- After several requests, both from inside and outside the HVSC team, we decided to integrate Peter Weighill's old Master Composer Collection into HVSC. Many of these tunes bear a lot of nostalgic value, so it would be a shame not to have them. Be aware that due to a bug in the libsidplay1 libraries they might sometimes lock up your sidplay1-based player. They work fine on Sidplay2 and on a real C64, though.
- The HVSC crew got a new member: We all want to welcome back Wilfred Bos, the author of the fabulous ACID64 Player for users with hardware based SID cards such as the Hardsid or the Catweasel. Wilfred was a HVSC member of the first hour, but had to quit four years ago due to real life priorities. Welcome back, Wilfred!
- While talking about Wilfred, he just recently released a new version of ACID Player, upgrading the version number to 2.0. New features:
  - Cycle based emulation of 6510 CPU, 6526 CIA and 6569 VIC-II chip
  - Cycle based playback of SID data



- Support for sample playback of RSID and C64 Program files
- Support for playing PRG and PC64 files
- Simulation of space bar and joystick fire buttons to skip intros
- Support for restoring screen size
- Added 6510 CPU performance indicator
- Support for skipping silents of beginning of tunes
- Better resetting of SID chip
- Faster voice bar animation
- Fix for frequency adjustment to avoid warble sounds in some tunes
- Many other improvements

Download it here: <http://www.acid64.com>

## Asterion Sid-Tracker v1.1

Released 31 May 2004 the new version of Tinnitus editor.

major improvements:

- vibrating portamento implemented
- vibrating arpeggio implemented (experimental)
- independent detune level for each instrument
- multispeed up to 4x
- pulse step programming
- 11-bit filter step programming
- various bugs removed
- heavily refined packer

Download from <http://tinnitus.prv.pl>

## reSID 0.16

On 11 June reSID 0.16 was released.

reSID is the sid emulation library developed by Dag Lem.

This release fixes a few bugs, most notably a one cycle error in the ADSR delay bug emulation. This error only affected fast sampling.

Resampling is necessary to achieve the ultimate sound quality in SID emulation. In collaboration with Laurent Ganier, the resampling code has been highly optimized and streamlined for vectorizing compilers. High quality resampling can now be run in real time on off-the-shelf hardware. With vectorizing compilers (e.g. Intel's icc) real time resampling can be achieved even on somewhat older hardware.

For the full changelog and source code download reSID 0.16 at:

<ftp://ftp.funet.fi/pub/cbm/crossplatform/emulators/resid/resid-0.16.tar.gz>

## Sidplay2 + ReSID 0.16

On 15 June, Simon White had released the new version of libsidplay2 library modified for taking use of the new reSID 0.16 library and the changes made in the May released of sidplay2/w.

Download all the stuff from <http://www.gsldata.se/c64/spw/> and <http://sourceforge.net/projects/sidplay2>

## 2010 SID COMPO V1

Ended on 1 July 2004 the 2010 SID COMPO music competition organized by Rolemusic on [www.2010compo2010.tk](http://www.2010compo2010.tk)

Here the result:

1. Hein Design/Vision	Chill Addiction	7.16
2. Richard	High Speed	6.64
3. Surgeon/Vulture Design	NICE DREAM	6.58
4. hukka	Leego II	5.77
5. maktone / fairlight	mr freshness	5.66
6. Raf/Samar/Vulture design	God is a girl	4.92
7. Thomas Wurgler (AKA Trez	Happy Day	4.79
8. Anders Carlsson (Zapac)	Theme 1:22	4.64

## Libsidplay1 1.36.59

On 17 August 2004 was released an update to libsidplay1 (a low-cpu consuming sid emulation library) that fix security bug in debian.

Download from <http://www.geocities.com/SiliconValley/Lakes/5147/sidplay/linux.html>

## SIDPLAY for Mac OS X 3.3

This sidplayer for Mac OS X was released on 17 August 2004.

New to this version:

- Updated to latest libsidplay2 and reSID components
- Integrated HardSID support courtesy of Teli Sandor and Hard Software
- Redesigned the user interface using a separate info window
- Fixed visual glitches on Panther
- Added mixer panel with mute and solo controls
- Improved filter adjustment panel with options to add and remove filter spline points
- Added automatic download of composer photos from <http://composers.c64.org/>
- Added feature to play a random tune from the HVSC collection
- Added dock menu
- Improved search in playlists
- Updated to PSID64 0.6
- Fixed various major and minor bugs

Download from <http://www.sidmusic.org/sidplay/mac/>

## CGSC v1.12

Version 1.12 of the Compute's Gazette Sid Collection was released on 21 August 2004.

The collection now contains 5998 MUS files, 1314 STR files and 1567 WDS files.

Download the update from <http://www.c64music.co.uk>

## HVSC 5.8

HVSC Update #40 was released on 27 September and is available at [www.hvsc.c64.org](http://www.hvsc.c64.org)

After this update, the collection should contain 27,653 SID files!

This update features (all approximates):

- 563 new SIDs
- 724 fixed/better rips
- 2 fixes of PlaySID/Sidplay1 specific SIDs
- 10 repeats/bad rips eliminated
- 323 SID credit fixes
- 152 SID model/clock infos
- 27 tunes moved out of /DEMOS to their composers' directories
- 15 tunes from /DEMOS/UNKNOWN identified :-)  
(and two go back due to bad credits)
- 12 tunes moved out of /GAMES to their composers' directories  
(and one goes back due to bad credits)

Main Composers featured in this update:

(Artists marked with NEW! are either completely new to the HVSC or they get their own directory in this update)

Deek	James Hilthy (NEW!)
Kathy and Kidd Clark (NEW!)	Henning Rokling
John Vanderaart	Agemixer
Alien	Richard Bayliss
Glenn Rune Gallefoss	Ciaran
CMP (NEW!)	ELA
G-Fellow	Harmony Productions
Hein Holt	John Keding
Sascha Zeidler (Linus) (NEW!)	Lordnikon
Maktone	Murdock
Nadira	Ne7 (NEW!)
Richard Nygaard (NEW!)	Puterman
Reed (NEW!)	Scortia
Tango	Jeremy Thorne (NEW!)
Asterion	Zeus

So what is new about this update?

- We've got the music compo entries from the following parties in this pack:  
LCP 04, North Party 9, Assembly 04, Primary Star 04 and Solskogen 04
- The mega hyped Turrigan 3 soundtrack is of course also in this update, check out /VARIOUS/S-Z/Sonic/Turrigan\_3.sid. And head over to [www.smash-designs.de](http://www.smash-designs.de) and drop a mail to AEG and express your interest in Katakis 2, which, if all goes well, will be coproduced by ye ole master Manfred Trenz again!
- This update contains about 500 multi speed fixes. Due to an incorrect assumption regarding the

number of cycles per frame (the C64 has \$4CC7 and not \$4CF9 cycles per frame, due to running at 50.125Hz instead of 50Hz) most of the multi speed tunes that you will find in HVSC were played a little slower than intended after converting them to play via the CIA timer.

Also we removed some completely unnecessary cycle eating \$DC0E writes, which caused a larger slowdown than the mere \$22 cycles difference mentioned above.

Although a 0.2% speed difference may not sound much in audible terms, some tunes do require very precise timing to get the instruments to sound correct, and so we felt it would be best to fix them all in one big effort in this update.

Do note that we haven't fixed all the old game tunes yet, as some native CIA timed tunes were indeed made to be played at exactly 50 Hz instead of 50.125 Hz and those will be fixed as necessary.

- We divided the update in a /fix and /new section. This makes it easier for us to keep track of the update during its creation and it's at the same time more convenient for you to listen through the update pack, so you can avoid the fixes and concentrate on the new tunes only (we assume that's your prime reason you update your collection regularly, to get new SIDs to listen to). :-)  
Of course, listening to the /fix section might be interesting to some of you as well, as it almost exclusively contains multispeed tunes! :-)

## Catweasel MK4

Announced in July 2004, the new Catweasel MK4 pci card will be released as soon as some delay due to hardware components will be resolved.

Improvements to the SID audio part in this new card:

- **DC-DC converter eliminates noise.** On the Catweasel MK3, it was possible that noises from 3D-graphics cards or high-speed harddrives were coupled into the 12V-power supply of the SID audio part. This cannot happen any more on the Catweasel MK4, because a DC-DC converter is an insuperable obstacle for such noises.
- **Cycle-exact control.** In addition to the known programming that's compatible with the Catweasel MK3, the MK4 has a sophisticated script-language for SID control. This lets the programmer define the exact time for data to be written to the SID chips. To make sample playback sound exactly like on a real C64, the data rate to the SID chip must be kept at a constant rate. This is accomplished with a Fifo memory that is big enough to maintain the datarate even under high processor load conditions.
- **Digiboost for new SID versions.** As opposed to the 'classic SID' 6581, the newer SID-chips 8580 and 6582 cannot playback samples any more. This option, which is also called 'the fourth voice', is replaced by two sigma-delta converters on the Catweasel MK4, so the fourth channel is also audible with the newer SID versions. Since the filter properties and the sound of mixed waveforms of all SID versions have their supporters, this should make the decision for the right chip a little easier.
- **Filter capacitors selectable.** Commodore has defined three different capacitor values for the filters of the SIDs during the years that this chip has been produced. The result was that the same chip sounded differently if used in different computers. To bring the sound as close as possible to what you are used to, the filter capacitors can be chosen with a few jumpers.

- **Precise clocking.** The Catweasel MK3 used the commodore-chip 8701 to recreate the exact same clock. Since our stock of this chip is empty with the Catweasel MK3 being sold out, we have cloned it on the main logic chip of the Catweasel MK4: The exact base frequency is generated with crystals that have been made especially for us. By division and multiplication according to the specifications of the C64 schematics from 1982, we managed to replace the 8701, which is not made any more. Even the slight difference between PAL and NTSC computers is software-selectable!
- **Two SIDs for stereo sound.** You'll have twice the SID pleasure after installing a second SID chip. Every SID has it's own selection of filter capacitors, and SIDs of all versions can be mixed.

Technology improvements:

- **compatible with 3.3V and 5V PCI slots.** Even though PC boards with 3.3V PCI slots are not yet widely available, the Catweasel is prepared for it. The roadmap of the PCI special interest group plans to abandon 5V PCI slots within foreseeable time, and the Catweasel is perfectly suited for that date. Local generation of the 3.3V power also ensures proper function on early PCI motherboards that do not comply to the ATX standard.
- **Two DMA interfaces.** In addition to processor-based data transfer, the Catweasel MK4 can exchange data with the main system through two low-speed DMA channels: The first goes through the PCI slot, and it has a capacity of about 8K per second and direction. The second uses the direct connection to the onboard-floppy controller, and the speed is up to 100K per second.
- **Low power consumption.** The Catweasel MK4 makes use of the latest FPGA technology with 2.5V core voltage. This reduces the power consumption of the new controller to a fraction of what the Catweasel MK3 used. This also reduces heat generation a lot.
- **Re-configurable logic.** The FPGA on the Catweasel MK4 is completely re-configurable by the drivers. This means that a hardware update can be done through the internet! Should we find a disk format that cannot be handled with the current hardware, the core of the Catweasel can be 're-wired' to address the problem. The controller doesn't even have to be taken out of the computer for this update!
- **Drivers for many operating systems.** The Catweasel MK4 is delivered with drivers for Linux, Windows 98(se)/ME/XP/2000, Amiga OS4, and for Mac OS X at a later date.

Other improvement not related to sid:

- Kylwalda built in
- Support for auto-eject drives
- Hard-sectored disks supported
- Dual-ported memory
- More flexible read- and write operations
- Extensive timer-functions
- All events can trigger an interrupt (IRQ)
- Amiga mice supported in hardware
- Every signal can be programmed as output
- Compatible with CD32 pads

More information from [http://www.jschoenfeld.de/news/news\\_e.htm](http://www.jschoenfeld.de/news/news_e.htm)

## XSidplay 1.6.5.2

Released in October 2004 the new version of the linux sid player:

- fix Qt-mt detection
- fix streampos arithmetic cast in File.cpp
- fix tab order in filter dialog
- load config from global /etc/xsidplay.ini
- fix underquoted automake definitions

The player is available at: <http://www.geocities.com/SiliconValley/Lakes/5147>

## The SID Compo IV

The c64.sk music competition was taken from 22.november to 5.december 2004

There were 32 competing tunes plus 2 bonus tunes. Here the result, more info at the [www.c64.sk](http://www.c64.sk) site:

Place	PTS	Song Name / Author	ID
1	500	Natural Cause of Death / Thomas Mogensen (DRAX)	10
2	492	Natural Cause of Life / Alexander Rotzsch (Fanta/Oxyron/Plush)	31
3	467	Crush / Stellan Andersson (Dane/Booze Design)	24
4	457	Level Kamil Wolnikowski / (Jammer/EXON/MSL/Samar)	12
5	456	Vinyl Headz / Michal Hoffmann (Dat Nigga Randall / Ghettomuzik)	14
6	439	Unleas The F. Fury / Lasse rni (Cadaver/CoverBitOps)	21
7	438	Sherona Red / Hein Holt (Hein/Focus)	6
8	424	Focus On Food / Ronny Engmann (Dalezny)	15
9	423	Flight of Serpent Dragon / Rafal Kazimierski (Asterion)	11
10	417	Mutator / Mark Thomas Ross (MTR1975)	3
11	395	The Battle Of [illegible] / Maciej Stankiewicz (Trompkins/Tinnitus)	17
12	394	Are you satisfied ? (prw) / Marcin Kubica (Booker/Amorphis/Onslaught)	32
13	394	Garden of the Black Rose part 4 / Krzysztof Malczewski (Phobos/Samar)	4
14	387	Second Steps / Aleksy Eeben (aeb/Carillon)	7
15	387	Marvel / Gerard Hultink (GH/Toondichters)	20
16	385	Sunday Weirdness / Marc van den Bovenkamp (No-XS/World Wide Expressive)	29
17	375	Path Of Pain / Daniel M. Gartke (Turtle / The Demented)	25
18	375	Cosy Brains Pulsating / Mark Waldauskat (Heinmukk/Salva Mea)	19
19	354	Your Average Giallo / Sascha Zeidler (Linus)	16
20	339	Sidcompo IV / Martin Nordell (Maktone/Fairlight)	2
21	326	Nicca Pop / Anders Carlsson (Zapac)	13
22	312	Bonuslevel! / Tony Cav (Ferrara / phObos team)	8
23	306	Lost Again / Richard Bayliss (Richard / The New Dimension)	1
24	303	TRYITONlx2&4-first48sec / Przemek Mroczkowski (Surgeon/Vulture Design)	26
25	298	FarOff CyberSpace (short) / Robert Dfler (LordNikon)	9
26	291	Depleted Uranium / Linus Akerlund (Puterman/Fairlight)	18
27	286	Mididrums / Jaymz Julian (A Life in Hell)	27
28	267	Riffing On The Edge / Andrew Lemon (ne7/Creators/Napalm/Rebels)	22
29	252	COMPOSED \$c0de/ Rafal Szyja (Raf/Samar/Vulture)	30
30	229	ChipSlip / Marcus Jansson (CMP)	28
31	194	Into The Sunset / Oliver VieBrooks (Six of Dloc)	5
32	188	Distoken Disinterpreter / Langel Janson L. Bookbinder Esq. IX	23

## SIDBrowser V2.4

SIDBrowser is the Windows program that let your life to be more easy with sid:

- Browse easy in HVSC
- Define favourite paths
- See picture of musician (if available)
- Play subtune by button
- Read background-information of SID tune information list (STIL)
- Multilingual: Czech, English, German, Italian, Japanese, Polish

- Quick Search for filename, copyright, songlength, words inside STIL
- Show while browsing filelist sorted by Year
- Read documents of HVSC easy
- Browse inside Top100 of HVSC
- Create Play-List and edit them easy
- Toggle between 5 playlists, to create your own Bestest Best of Best ;-)
- Create SIDList (Mega-playlist)
- Import Playlists from sidplay (.spl), winamp (.m3u)
- Export Playlists to sidplay (.spl, Format 2)
- Listen Playlists random (shuffle)and repeat
- History of listened SIDs
- Get and put playlists online. Exchange them with your friends all the world!
- Chat with other SIDians ;-)
- Minimizable to Tray
- Taskbutton scrolls information about actual playing SID



The new version was released 2 January 2005 with this new features:

- little history-bug fixed
- photos updated
- updated splashscreen
- some values updated
- new Document for reading: Songlength-Database-Readme
- Language files updated a little bit
- include the new songlength-database 5.8 for HVSC 5.8

Download from <http://www.sidbrowser.com>



# Dustbin (Stefano Palmonari) Interview!

by Stefano Tognon

This time you can read an interview with Stefano Palmonari that occurs with some emails in December.

***Hello, Stefano,***

***Can you introduce yourself to the readers: something about you and your real life.***

Hi everybody! My name is Stefano Palmonari, I'm 31 years old and I was known in the scene as Dustbin. I started composing computer music at 15 without any specific musical knowledge: just a lot of Rob Hubbard's tunes recorded on a tape... And tons of Jeroen Tel's tunes played 24:7 :-)) so I learned some tricks just listening to the musics... I was very lucky too, 'cause I met a guy from F4CG in Ferrara, wich I was born, that gave me some music editors like, for example, Future Composer. I started typing shit in it until I was able to produce something "ear-able":-))  
Actually I sing in a Queen-Tribute band, I'm married and I have a daughter: she's 2 years old and she already likes music of every kind: from Mozart to Coal Chamber...!

***You are one of the few Italian sid composers that worked into a game company that produce games in '90. Can you speak about those (old) days?***

Well... Italian musicians were talented (ie: Ivan Del Duca, Paolo Galimberti, Gianluca Gaiba, Nicola Tomljanovic...) and graphic artists too... Italian coders were quite good but not at the same level as Dan Phillips :-)) ... The only impressive Italian coders I remember are Paolo Galimberti and Andrea Pompili. Anyway, the main problem were the Software Houses themselves... They wanted talented people but they didn't wanted to pay them... :-|

Talking about those good old days, I remember a lot of hours passed on the C64 learning to use the editors. I was 17 and you can't imagine how happy I was when I finished my first shitty-tune done in Rockmonitor :-)) That shitty tune was sent to Simulmondo, more or less as a joke, but they called me to work for them... So I did.

90's were, computercally speaking, AMAZING...

***You have compose music for Amiga, PC, and even for Game Boy: what are the difference about composing music for the Sid chip and the other architectures?***

Everybody knows that .mod format was introduced with the Amiga but in some way we can consider Rockmonitor, on the glorious C64, as an ancestral of it.

The SID chip allows you to manage just 3 channels and you have to practice a lot on it to gain a balance between technique, full control of the tracks and musical skill!

On the Game Boy I did 3-tracks midi files (Rayman), testing it with a proprietary player: that was quite easy, 'cause the midi editor and my Roland makes the difference. On the C64, using the good old editors, we have to think in hex and write something like

```
SND.01  
DUR.0c  
C-5  
SND.02  
C-4  
DUR.18  
F#4  
...
```

Then we have to handle the arpeggio table, filter table, wave table... Every C64 editor, or at least

the editors I used, works keeping in mind the amount of memory, not a user friendly interface. I think the differences in composing music, between the platforms, are related to the interface... With PC-trackers you have a full control of every single note, at tick resolution. You can find some trackers for the GB, too, but the best seems to be Musicbox, a commercial product. In my opinion it's very difficult to use due to the hardware used: a real GB :-)) In order to compose in a "reasonable" way on the GB you have to use a pc-tracker in audio-emulation, like Paragon 5 Tracker.

Well, I found quite easy to compose on the Amiga/PC but tremendously difficult to compose with the C64. I found Renoise, on the PC, a fantastic editor/tracker... I hope they'll add a realtime sid-emulation plug-in/VST... something like that :-))

***I know that you are not a musician/composer, but your Sid music I'm listening now is what I call fantastic music! Maybe you can tell as how this is possible (some sid people could learn something from your experience).***

Oh... Thank you! But maybe "Fantastic" is too much. Anyway, I think the best feature invented is the arpeggio... You can use it "chopped" for funky tunes. I have to say one other thing: none of the workstations I used, neither a soundcard, gives you that typical WARM lead sound... Maybe just a minimoog! The SID chip is unique, and every SID chip is unique (someone said "filters" ? :-))

Here's a good snare ripped from zakazazam by Glenn Rune Gallefoss:

```
wavetable 81 41 40 80 80
arptable  CE AE AC CA DE
```

This data works fine with Sadotracker... :-)) (You can break it and shake it and re-compose it but... I think first Rob Hubbard music routines were studied well :-))

C64 produces some of the best sounds I ever heard...

Anyway, as I said, in my humble opinion, there are something to keep in mind when you compose on the SID:

- SID has just 3 channels (+1, when available :-)) , you can try to do miracles but only if...
- ... you reach a good balance between technique and the channels available.
- Learn to use PERFECTLY one or two editors... don't waste your time trying to learn all the editors around
- Listen to the best .sid around and try to remake it with your ears only (That's what I did 10 years ago!! With the difference I didn't have .sid files like today but just tapes live-recorded from TV :-)) (no audio out).

***At this point a question that I already know the answer :-)) : can we listen again to a your new composition for the C64 (hint: LSS2)?***

hehe, yes! :-)) Thanks to you, the coder, I did some jingles for Little Sara Sisters 2 :-))) Nothing special, but it was very exciting to type the good old commands in the pattern editor :-))

***A technical question: what do you find the actual editor/tracker compared with the ones you use 12 years ago? Some years are passed, the SID chip is the same, but what about the tools?***

I find Cybertracker and Odintracker user friendly and it's fantastic to see they've been developed in this years, after the death (just for the market!) of the C64.

Anyway, for some nostalgic reasons, but technical also, my preference goes to Sadotracker: the essential features are fully present and I don't have to remember a lot of keys to type and activate something.

***Now some quick final (standard) questions:  
Real machine vs emulator: what do you think of?***

Although the C64 is 99% emulated I will not sell my real C64 for any amount of money (I've already sold one of my two C64's but I won't sell the last one).

Above all the SID chip is critical to emulate: it's almost impossible to emulate it perfectly. They reached good results but emulating filters, for example, requires a lot of CPU and it doesn't work fine. This speaking about technical facts.

Emulation is fantastic because, personally, I re-discovered forgotten feelings when I played with Gyruss, for example... Or Cybernoid... That was the first Tel's tune I've listened to... And it's still beautiful.

***6581 vs 8580 chip: any (musical) preference?***

6581, without doubts. The new 8580 was not good compared to the old 6581!  
Sounds incredible but I think so.

***What is the worst sid that you compose and the better one?***

The worst one was the title music of Big Game Fishing... On the other hand, I think my best work was did for Basket Play Off.

***Who are your best sid authors?***

What the hell... :-) Rob Hubbard, who invented the profession, Jeroen Tel, whose technical skill was, and still is, incredible (he was already musician at age 0) and Martin Galway. These three persons has changed the story of the C64. Jeroen Tel is my music hero, but Rob Hubbard is the best and many musicians (and music-routines coders, too) have to say "thanx" to him.

***What are the best sids ever in your opinion?***

Hard question to answer... I think Sanxion's tape loader music...  
Then Cybernoid's title track and, last but not least, Wizball's soundtrack...  
I think Wizball is the best game ever made on the C64.

***Finally, many thanks for the time you give for this interview, and now you can say any things you want that the people will read from you!***

Thank you very much for interviewing me!

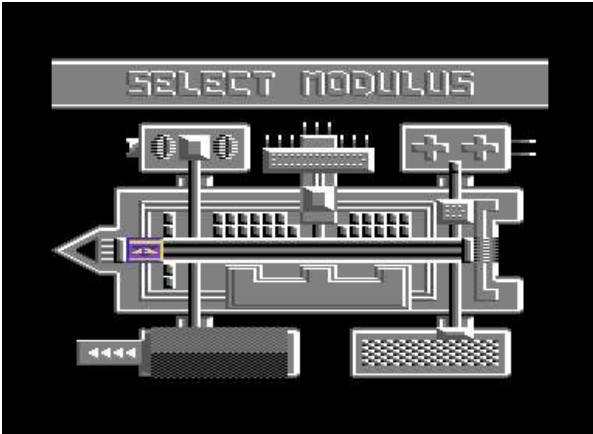
Well, first of all excuse my spaghetti-english, sooner or later I'll improve it :-)))

My dream is to turn back child at age 13 when my parents brought to me my first C64.

Then start again with all the things I did (maybe with some minor changes :-))

## Inside Modules

by Stefano Tognon <[ice00@libero.it](mailto:ice00@libero.it)>



Modules is one of the game programmed by Italian people in the '80 and in particular Ivan Del Duca has wrote his own music engine for creating his music.

Ivan has declared in some Italian interview that he was not a true musician (he is a game coder, even today), but that he probably wrote the first Italian music engine around, instead of used the one's that were available.

I would like to see how the engine works, and I choose to analyze Modules because I like the

music of this game.

## Structure

The player use some common structures:

- Tracks
- Patterns
- Music commands

A track is composed by a sequence of patterns to executes. There are 3 tables of tracks: one for each voices.

An entry in the track has this structure:

- high address of pattern to execute
- low address of pattern to execute
- max index in pattern to execute (FF=max, can be terminate by instruction)
- number of times to repeat the given pattern

This means that we have an 16 bit address of the pattern to execute, the max number of instruction in the pattern to execute (a FF values means that will be a special instruction into the pattern that will end the pattern execution) and the number of times this pattern will be repeated.

Some examples:

```
track1:  
  .byte >pat00, <pat00, $3D, $02  
  .byte >pat01, <pat01, $51, $00  
  .byte >pat03, <pat03, $FF, $00
```

Here *pat00* will be executed for \$3D instructions and it will be repeated 3 times, then *pat01* will be executed 1 time for \$51 instructions and finally *pat03* will be executed 1 time until the instruction flow reach the instruction that terminate the pattern.

## Instrument

This engine handles 16 instruments (even if it could be possible to use up to 256 if we extend the tables of attributes). In fact an instrument is built by 9 parameters putted into 9 tables of 16 entries.

Let we analyze one instrument definition:

- Attack-Decay of the voice
- Sustain-Release of the voice
- Low value of wave (for rectangular waveform)
- High value of wave (for rectangular waveform)
- Control of voice when it is on
- Control of voice when it is off (e.g. release)
- High byte of filter cut frequency
- Filter resonance
- Filter mode

As we can see, one instrument has very limited capability: ADSR, Wave duty cycle (fixed) and control values for the voice (two types). This means that all the effects that the player can reproduce are not associated with the instrument, with one exception: the filter.

In fact, the last 3 bytes of the instrument definition are for controlling the filter of the voice, by selecting the type of filter and resonance/cut frequency (however dynamically change onto filter are to be performed with instruction in the pattern)

## Pattern

The instructions used into the patterns are quite complicated, as they use low address pointer as an instruction (so, the code cannot be relocated easily).

But let we see how the instructions are made, by looking at the program (in pseudocode)

- Read a pattern value
- If it is 0, then the next value is a note duration
- If it is negative (e.g with value from \$80 to \$FF) it is a command
- Else it is a note; After a note, we read another value:
  - If it is above \$9F this is a note duration
  - Else it is another note and the duration is like the previous used one

As you can see this is not a so clear code, as note duration need even special command (0 byte) for be used correctly.

A declared note is an index for frequency values: a table of 12 frequencies is used (one octave) and the others are calculated each times by manipulating that little table. The note is so made in this manner:

octave-note

Where octave is the octave to use (from 0 to 7) and note goes from 1 to C.

Valid note duration values are from \$A0 to \$E0: this is an index that is used for read the real

duration from a table of duration. This table is of 16bit, so we can have very-very long sound.

Finally, we must see the commands available. Remember that a value of a command is the low pointer of the memory location that contains the code. The high value of the pointer is fixed.

<i>Value</i>	<i>Command</i>
\$80	End of pattern
\$83	Set control for release
\$86	Set a new instrument
\$89	Set the volume
\$8C	Portamento add
\$8F	Portamento down
\$92	Stop effect
\$95	Copy sid register values
\$98	Not used
\$9B	Various action: vibrato, gosub, return, play note

- The command \$80 will terminate the pattern. Remember that the pattern can be terminate by the length that is given into the track, or by this command.
- \$83 will make the release of the envelope (gate=0). The value used is the one declared into the instrument table.
- \$86 is the command for changing the instrument. It has two parameters:
  1. The instrument number to use
  2. A flag: <> 0 means to reload the filter parameters, otherwise currents will be used
- \$89 sets the volume, and so it has only one parameter:
  1. The volume (from \$0 to \$F) to use
- \$8C and \$8F will perform the portamento up and down, so it has those parameters:
  1. Frequency step to add/sub
  2. Delay between an add/sub
- \$92 is a command that stop all effects that are being done to this voice.
- \$95 is a command that is to be used for copy the values of sid registers. It has two parameters:
  1. Offset of source sid register
  2. Offset of destination sid registerYou can, for example, use this for copying the oscillator 3 value to one other register for putting a random value in it.
- \$9B is a command that performs many actions, according to the first parameter, so now we will see all the cases.

<i>Values</i>	<i>Command</i>
\$9B \$02 indType limit step delay(speed)	This command is for making effect like vibrato. There is the <i>limit</i> that will invert the adding sequence to became descendant. <i>Step</i> is the amount of value to add/subtract. <i>Delay</i> is the ticks between two add/sub, and so it is the speed. Finally <i>indType</i> is an index to a table of effect to use: 0: Manage actual filter cut frequency high value 1: Manage the wave high value of a voice 2: Manage the frequency high value of a voice (vibrato)
\$9B \$03	Return instruction: the flow of execution is passed back to the caller
\$9B \$05 note	This instruction will put <i>note</i> as output frequency. No action is taken about duration of this note.
\$9B val	Gosub intruction: the flow of execution is passed to the pattern that has \$C9 as high address and <i>val</i> as low address

As I have commented a lot the first patterns into the code, you can see how those commands are used directly by looking into the code.

## Code

Here the reverse engineering source code of the player. Note that all he right remain to Ivan.

```

;PSID file version 2
;Load Address: 0
;Init Address: c57b
;Play Address: cfe8
;name:      Modulus
;author:    Ivan Del Duca
;copyright: 1988 System
;songs: 1 (startsong: 1)
processor 6502
.org $C000

    ldx #$00
    stx vTemp

loopNext:
    lda disVoice,x          ; voice 1 disable?
    bne nextVoice

    lda flagMus,x          ; is to process music flow?
    beq nextVoice

    dec flagMus,x          ; no more to process
    stx vIndex             ; voice index
    jsr processMusic

nextVoice:
    inc vTemp               ; next voice
    ldx vTemp
    cpx #$03
    bne loopNext

    jsr makeDurEffect
    lda disMusic           ; is to disable music?

```

```

        bne  disPlayer
        rts

vTemp:
        .byte $03                ; voice temp index

disPlayer:
        jsr  disablePlayer
        lda  #$00
        sta  actFiltCut
        rts

processMusic:
        lda  maxIndex,x          ; max index of pattern to execute
        cmp  patIndex,x          ; actual pattern index
        beq  readTrack

makePatFlow:
        jsr  setPatAddr
        lda  patIndex,x          ; actual pattern index
        inc  patIndex,x          ; actual pattern index
        tay
        lda  ($14),y             ; read from pattern
        beq  jmpSetDuration
        bmi  execInstr           ; is an instruction?
        tay                               ; it's a packet note
        jsr  makeControlOff

setFreq:
        jsr  setFrequency        ; set frequency of this note
        lda  relDelay,x          ; delay to reload
        sta  actDelay,x          ; actual delay
        jsr  makeControlOn

setDuration:
        jsr  readNextValue       ; read note duration
        cmp  #$9F
        bcs  readDuration
        jsr  goBackPIndex        ; go back with index
        lda  packDuration,x      ; use previous pack duration

readDuration:
        sta  packDuration,x
        sec
        sbc  #$A0
        tay
        lda  durationHiTab,y
        sta  durationHi,x
        lda  durationTab,y
        sta  duration,x
        rts

jmpSetDuration:
        jmp  setDuration

execInstr:
        sta  instAddr
        jmp  (instAddr)          ; execute the instruction

;=====
; Read next attern value
; in case it reads a new pattern
; from track
;=====
readNextValue:
        lda  maxIndex,x          ; max index of pattern to execute
        cmp  patIndex,x          ; actual pattern index
        beq  readTrack

        jsr  setPatAddr
        lda  patIndex,x          ; actual pattern index
        inc  patIndex,x          ; actual pattern index
        tay
        lda  ($14),y
        rts

readTrack:
        lda  trackIndex,x        ; actual track index
        tay
        cpx  #$00
        beq  readTrack1
        cpx  #$01
        beq  readTrack2

readTrack3:
        jsr  testForRepeat
        lda  track3,y
        sta  hiPatAddr,x
        iny

```



```

    lda track3,y
    sta loPatAddr,x
    iny
    lda track3,y
    sta maxIndex,x           ; max index of pattern to execute
    lda #$00                 ; reset pattern index
    sta patIndex,x          ; actual pattern index
    jsr readNRepeat3
    sty trackIndex+2        ; actual track index
    jmp makePatFlow

readTrack2:
    jsr testForRepeat
    lda track2,y
    sta hiPatAddr,x
    iny
    lda track2,y
    sta loPatAddr,x
    iny
    lda track2,y
    sta maxIndex,x           ; max index of pattern to execute
    lda #$00                 ; reset pattern index
    sta patIndex,x          ; actual pattern index
    jsr readNRepeat2
    sty trackIndex+1        ; actual track index
    jmp makePatFlow

readTrack1:
    jsr testForRepeat
    lda track1,y
    sta hiPatAddr,x
    iny
    lda track1,y
    sta loPatAddr,x
    iny
    lda track1,y
    sta maxIndex,x           ; max index of pattern to execute
    lda #$00                 ; reset pattern index
    sta patIndex,x          ; actual pattern index
    iny
    lda track1,y
    sta nRepeat,x           ; number ot times to repeat
    iny
    sty trackIndex          ; actual track index
    jmp makePatFlow

;=====
; Set pattern index address
;=====
setPatAddr:
    lda loPatAddr,x
    sta $14
    lda hiPatAddr,x
    sta $15
    rts

;=====
; Make the voice control off
;=====
makeControlOff:
    lda vCntOff,x
    jsr indexToOffset
    sta $D404,x             ; Voice 1: Control registers
    ldx vIndex              ; voice index
    rts

;=====
; Make the voice control on
;=====
makeControlOn:
    lda vCntOn,x
    jsr indexToOffset
    sta $D404,x             ; Voice 1: Control registers
    ldx vIndex              ; voice index
    rts

;=====
; go back pattern index
;=====
goBackPIndex:
    dec patIndex,x         ; actual pattern index
    rts

;=====
; set instrument to use
;=====
setInstrument:
    jsr readNextValue
    tay                    ; instrument number

```

```

setInstrumentY:
    jsr  indexToOffset
    lda  tbAD,y
    sta  $D405,x                ; Generator 1: Attack/Decay

    lda  tbSR,y
    sta  $D406,x                ; Generator 1: Sustain/Release

    lda  tbWLo,y
    sta  $D402,x                ; Voice 1: Wave form pulsation amplitude (lo byte)

    lda  tbWHi,y
    sta  $D403,x                ; Voice 1: Wave form pulsation amplitude (hi byte)

    ldx  vIndex                  ; voice index
    sta  waveHi,x
    sta  waveHiRel,x

    lda  tbCntOn,y              ; voice control (on)
    sta  vCntOn,x

    lda  tbCntOff,y            ; voice control (off)
    sta  vCntOff,x

    tya
    sta  actInstr,x            ; actual instrument number

modCode:
    sty  temp
    jsr  readNextValue          ; filter flag state (<>0=reload filter parameters)
    bne  reloadFilter
    lda  bitWiseTab,x
    eor  #$FF
    and  actFiltRes
    sta  actFiltRes
    sta  $D417                  ; Filter resonance control/voice input control
    cpx  #$00
    bne  noResAFC

    lda  #$00                  ; reset for voice 0
    sta  actFiltCut
noResAFC:
    jmp  jmpProcessMusic

reloadFilter:
    ldy  temp

reloadFilterY:
    lda  filtCutFH,y
    sta  $D416                  ; Filter cut frequency: hi byte
    cpx  #$00
    bne  skipActFC
    sta  actFiltCut

skipActFC:
    sta  actFiltCutFH
    lda  actFiltRes
    and  #$0F
    ora  filterResCtr,y
    sta  actFiltRes
    lda  actFiltVol
    and  #$0F
    ora  filterMode,y
    sta  actFiltVol
    sta  $D418                  ; Select volume and filter mode
    lda  bitWiseTab,x
    ora  actFiltRes
    sta  actFiltRes
    sta  $D417                  ; Filter resonance control/voice input control
    jmp  jmpProcessMusic

;=====
; Disable the passed x voices
;=====
disableVoice:
    lda  #$01
    sta  disVoice,x            ; disable this voice
    rts

;=====
; Set the volume of voices
;=====
setVolume:
    jsr  readNextValue          ; read the volume
    sta  temp
    lda  actFiltVol
    and  #$F0
    ora  temp

```

```

    sta actFiltVol
    sta $D418          ; Select volume and filter mode
    jmp jmpProcessMusic

;=====
; Change control of voice for
; release phase
;=====
makeRelease:
    jsr makeControlOff
    jmp jmpProcessMusic

;=====
; portamento up effect (can be used
; also for vibrato in conjunction
; with portamento down)
; par1: freq. step to add
; par2: delay to reload
;=====
portUp:
    jsr readNextValue      ; freq. step to add
    sta freqStep,x
    jsr readNextValue      ; delay to apply
    sta relDelay,x         ; delay to reload
    sta actDelay,x         ; actual delay
    inx
    txa
    dex
    ldy #$01               ; freq. add
    jsr onEffect
    jmp jmpProcessMusic

;=====
; portamento down effect (can be used
; also for vibrato in conjunction
; with portamento up)
; par1: freq. step to sub
; par2: delay to reload
;=====
portDown:
    jsr readNextValue      ; freq. step to sub
    sta freqStep,x
    jsr readNextValue      ; delay to reload
    sta relDelay,x         ; delay to reload
    sta actDelay,x         ; actual delay
    ldy #$02               ; freq. sub
    jsr onEffect
    jmp jmpProcessMusic

;=====
; Stop effect in the passed x voice
;=====
stopEffect:
    lda #$00
    sta flagEff,x          ; flag effect for each voices
    lda flagEff
    bne goProcessMusic
    lda flagEff+1
    bne goProcessMusic
    lda flagEff+2
    bne goProcessMusic

    lda #$00
    sta abilEffect        ; stop all effects

goProcessMusic:
    jmp jmpProcessMusic

;=====
; copy a sid register to another
; par1: SID index source
; par2: SID index destination
;=====
sidCopy:
    jsr readNextValue
    sta fromIndex,x
    jsr readNextValue
    sta toIndex,x
    ldy #$03
    jsr onEffect
    jmp jmpProcessMusic

;=====
; 02 indType limit step delay(speed)
; 03
; 05 packet_note
; <>
;=====
varyAction:

```

```

    jsr  readNextValue
    cmp  #$02
    beq  setVibLikeEf
    cmp  #$03                ; return (restore state)
    beq  jmpInstReturn
    cmp  #$05                ; set packet note
    beq  setPacketNote
    jmp  instGosub          ; gosub (store state)

jmpInstReturn:
    jmp  instReturn

setPacketNote:
    jsr  readNextValue      ; read packet note
    tay
    jmp  setFreq

setVibLikeEf:
    jsr  readNextValue      ; index type
    sta  indType,x
    jsr  readNextValue      ; limit value
    sta  limit,x
    lsr
    lsr
    sta  lowLimit,x
    lda  #$00
    sta  currValue,x
    jsr  readNextValue      ; read the step
    sta  step,x
    jsr  readNextValue      ; read delay (speed)
    sta  delayRelod,x
    sta  delay,x
    ldy  #$05                ; vibrato like effect
    jsr  onEffect
    jmp  jmpProcessMusic

;=====
; Performe a gosub instruction
;=====
instGosub:
    sta  temp                ; low address of gosub routine
    lda  loPatAddr,x
    sta  storedLoPatAddr
    lda  hiPatAddr,x
    sta  storedHiPatAddr
    lda  temp
    sta  loPatAddr,x        ; set low address
    lda  patIndex,x        ; actual pattern index
    sta  storedPatIndex
    lda  maxIndex,x        ; max index of pattern to execute
    sta  storedMaxIndex
    lda  actInstr          ; act instrument voice 1
    sta  storedInstr
    lda  #$C9
    sta  hiPatAddr,x      ; set high address
    lda  #$00
    sta  patIndex,x       ; actual pattern index
    lda  #$FF
    sta  maxIndex,x       ; max index of pattern to execute
    jsr  setPatAddr
    jmp  jmpProcessMusic

;=====
; Performe a return instruction
;=====
instReturn:
    lda  storedLoPatAddr
    sta  loPatAddr,x
    lda  storedHiPatAddr
    sta  hiPatAddr,x
    lda  storedPatIndex
    sta  patIndex,x       ; actual pattern index
    lda  storedMaxIndex
    sta  maxIndex,x       ; max index of pattern to execute
    jsr  setPatAddr
    ldy  storedInstr
    lda  #$60
    sta  modCode          ; put rts
    jsr  setInstrumentY    ; set only instrument paremeter
    lda  #$8C             ; put sty absolute
    sta  modCode
    lda  actFiltCut        ; actual filter cut frequency
    bne  toReload
    jmp  jmpProcessMusic

toReload:
    jmp  reloadFilterY

;=====

```

```

; Test for repeat the current pattern
;=====
testForRepeat:
    lda cRepeat,x          ; repeat counter
    cmp nRepeat,x          ; number of times to repeat
    bne incTestRepeat
    rts

incTestRepeat:
    inc cRepeat,x          ; inc repeat counter
    lda cRepeat,x          ; repeat counter
    cmp nRepeat,x          ; number of times to repeat
    beq noRepeat

    dey                    ; go back to previous row in track
    dey
    dey
    dey
    rts

noRepeat:
    lda #$00
    sta cRepeat,x          ; reset repeat counter
    rts

;=====
; Read the number of time to
; repeat for track 3
;=====
readNRepeat3:
    iny
    lda track3,y
    sta nRepeat,x          ; number of times to repeat
    iny
    rts

;=====
; Read the number of time to
; repeat for track 2
;=====
readNRepeat2:
    iny
    lda track2,y
    sta nRepeat,x          ; number of times to repeat
    iny
    rts

;=====
; Set the frequency of note
; Y=packet note: octave/note
;=====
setFrequency:
    tya
    tax
    and #$0F                ; take the note
    tay
    dey
    lda freqTableHi,y
    sta actFreqHigh
    lda freqTableLo,y
    sta actFreqLow
    txa
    and #$F0                ; take the octave of the note
    lsr
    lsr
    lsr
    lsr
    tay                      ; octave
    cpy #$00
    beq freqOk

calcFreq:
    lsr actFreqHigh
    ror actFreqLow
    dey
    bne calcFreq

freqOk:
    jsr indexToOffset
    lda actFreqLow
    sta $D400,x              ; Voice 1: Frequency control (lo byte)
    ldx vIndex                ; voice index
    sta actFreqLo,x
    jsr indexToOffset
    lda actFreqHigh
    sta $D401,x              ; Voice 1: Frequency control (hi byte)
    ldx vIndex                ; voice index
    sta actFreqHi,x
    lda #$00
    sta currValue,x

```

```

    lda waveHiRel,x          ; wave high to reload
    sta waveHi,X
    rts

jmpProcessMusic:
    lda vIndex              ; voice index
    jmp processMusic

;=====
; Convert voice index to sid offset
;=====
indexToOffset:
    sta temp
    ldx vIndex              ; voice index
    lda vOffset,x          ; read voice offset of sid
    tax
    lda temp
    rts

onEffect:
    inx
    txa
    dex
    sta flagEff,x          ; store voice+1
    tya
    sta freqEffect,x       ; store frequency effect
    sta abilEffect         ; abilitate all effects
    rts

;=====
; activate effects
;=====
activateEffects:
    lda flagEff
    beq testV2
    lda #$00
    sta vIndex              ; voice index
    jsr makeEffect

testV2:
    lda flagEff+1
    beq testV3
    lda #$01
    sta vIndex              ; voice index
    jsr makeEffect

testV3:
    lda flagEff+2
    beq skipEff
    lda #$02
    sta vIndex              ; voice index
    jsr makeEffect
skipEff:
    rts

;=====
; Make freq. effect
;=====
makeEffect:
    ldx vIndex              ; voice index
    lda freqEffect,x
    cmp #$01                ; freq. add
    beq addFreq
    cmp #$02                ; freq. sub
    beq subFreq
    cmp #$03                ; sid copy
    beq efSidCopy
    cmp #$05                ; vibrato like effect (even for wave and filter)
    beq jsrDecDelay
    rts

efSidCopy:
    jsr sidCopyEffect
    rts

jsrDecDelay:
    jsr decDelay
    rts

addFreq:
    lda actDelay,x          ; actual delay
    bne decADelay
    clc
    lda actFreqLo,x
    adc freqStep,x
    sta actFreqLo,x
    jsr indexToOffset
    sta $D400,x             ; Voice 1: Frequency control (lo byte)

```

```

    ldx vIndex                ; voice index
    lda actFreqHi,x
    adc #$00
    sta actFreqHi,x
    jsr indexToOffset
    sta $D401,x              ; Voice 1: Frequency control (hi byte)

    ldx vIndex                ; voice index
    lda relDelay,x           ; delay to reload
    sta actDelay,x          ; actual delay
    rts

decADelay:
    dec actDelay,x          ; dec actual delay
    rts

subFreq:
    lda actDelay,x          ; actual delay
    bne decSDelay

    sec
    lda actFreqLo,x
    sbc freqStep,x
    sta actFreqLo,x
    jsr indexToOffset
    sta $D400,x              ; Voice 1: Frequency control (lo byte)

    ldx vIndex                ; voice index
    lda actFreqHi,x
    sbc #$00
    sta actFreqHi,x
    jsr indexToOffset
    sta $D401,x              ; Voice 1: Frequency control (hi byte)

    ldx vIndex                ; voice index
    lda relDelay,x           ; delay to reload
    sta actDelay,x          ; actual delay
    rts

decSDelay:
    dec actDelay,x          ; dec actual delay
    rts

;=====
; copy sid register values effect
;=====
sidCopyEffect:
    lda fromIndex,x
    tay
    lda $D400,y              ; Voice 1: Frequency control (lo byte)
    tay
    lda toIndex,x
    tax
    tya
    sta $D400,x              ; Voice 1: Frequency control (lo byte)
    ldx vIndex                ; voice index
    rts

decDelay:
    dec delay,x
    beq delayIs0
    rts

delayIs0:
    lda delayRelod,x
    sta delay,x              ; reload the delay
    jsr setAddrForEff
    lda lowLimit,x
    sta temp
    lda currValue,x
    cmp temp
    bcc effectAdd
    jsr incByLowlimit

    cmp temp
    bcc effectSub
    jsr incByLowlimit
    cmp temp
    bcc effectSub
    jmp effectAdd

;=====
; set the address for effect
;=====
setAddrForEff:
    lda indType,x           ; index type of effect
    tax
    lda loIndType,x
    sta $8C

```

```

    lda hiIndType,x
    sta $8B
    lda #$D4
    sta $67
    lda sidIntType,x
    sta $66
    ldx vIndex          ; voice index
    rts

effectAdd:
    ldy #$00
    jsr adapt2ForVoice
    lda ($8B),y        ; read actual value for the effect
    clc
    adc step,x
    sta ($8B),y        ; store the new value
    jsr adaptForVoice
    sta ($66),y        ; put sid effect
    ldx vIndex         ; voice index
    jmp testLimit

effectSub:
    ldy #$00
    jsr adapt2ForVoice
    lda ($8B),y
    sec
    sbc step,x
    sta ($8B),y
    jsr adaptForVoice
    sta ($66),y        ; put sid effect
    ldx vIndex         ; voice index
    jmp testLimit

;=====
; Test limit for curr value and
; reset if necessary
;=====
testLimit:
    inc currValue,x
    lda currValue,x
    cmp limit,x
    bne skipResetCV
    lda #$00
    sta currValue,x
skipResetCV:
    rts

loIndType:
    .byte >actFiltCutFH, >waveHi, >freqHi

hiIndType:
    .byte <actFiltCutFH, <waveHi, <freqHi

sidIntType:
    ; filter cut f high, wave high, freq. hi
    .byte $16, $03, $01

;=====
; Adapt the register for the voice
;=====
adaptForVoice:
    sta temp
    lda $66
    cmp #$16          ; this is a value not depending of voice
    beq skipAsFix     ; so skip
    lda vOffset,x     ; read offset for the voice
    tay
skipAsFix:
    lda temp
    rts

;=====
; Adapt2 the register for the voice
;=====
adapt2ForVoice:
    lda $66
    cmp #$16          ; this is a value not depending of voice
    beq skipAsFix2    ; so skip
    ldy vIndex        ; voice index
skipAsFix2:
    rts

;=====
; Increment temp by low limit
;=====
incByLowlimit:
    lda temp
    clc
    adc lowLimit,x
    sta temp

```



```

        lda  currValue,x
        rts

;=====
; Make note duration and effects
;=====
makeDurEffect:
        ldx  #$00

testDVoice:
        lda  disVoice,x          ; is voice disable?
        bne  nextVoice_

        dec  duration,x          ; dec low of note duration
        bne  nextVoice_
        dec  durationHi,x        ; dec high of note duration
        lda  durationHi,x
        cmp  #$FF
        bne  nextVoice_

        lda  #$01                ; music flow is to process
        sta  flagMus,x

nextVoice_:
        inx
        cpx  #$03
        bne  testDVoice

        lda  abilEffect          ; abilitate effects
        beq  skipAbil
        jsr  activateEffects
skipAbil:
        lda  disVoice
        beq  skipDisable
        lda  disVoice+1
        beq  skipDisable
        lda  disVoice+2
        beq  skipDisable

        lda  #$01
        sta  disMusic            ; disable music

skipDisable:
        rts

;=====
; Init music
;=====
initMusic:
        lda  #$00
        ldx  #$00

repClear:
        sta  disVoice,x
        inx
        cpx  #$5F
        bne  repClear

        lda  #$01
        sta  flagMus
        sta  flagMus+1
        sta  flagMus+2
        sta  bitWiseTab
        lda  #$C6
        sta  instAddr+1
        lda  #$07
        sta  vOffset+1
        lda  #$0E
        sta  vOffset+2
        lda  #$02
        sta  bitWiseTab+1
        lda  #$04
        sta  bitWiseTab+2
        rts

;=====
; disable the player
;=====
disablePlayer:
        lda  #$60                ; put rts
        sta  $C000
        rts

;C5B5
        .byte $C0, $60, $00, $00
        .byte $FF, $FF, $FF, $FB
        .byte $00, $00, $02, $FF
        .byte $FF, $FF, $FF, $FF
        .byte $FF, $FF, $FF, $FF

```

```

.byte $FF, $FF, $FF, $FF
.byte $FF, $FF, $FF, $FF
.byte $FF, $FF, $FF, $FF
.byte $FF, $FF, $FF, $FF
.byte $FF, $FF, $FF, $FF
.byte $FF, $FF, $FF, $FF
.byte $FF, $FF, $FF, $FF

actFreqHigh:
.byte $54

actFreqLow:
.byte $7D

freqTableLo:
.byte $1E, $18, $8B, $7E
.byte $FA, $06, $AC, $F3
.byte $E6, $8F, $F8, $2E

freqTableHi:
.byte $86, $8E, $96, $9F
.byte $A8, $B3, $BD, $C8
.byte $D4, $E1, $EE, $FD

durationHiTab: ; table of not duration (high value)
.byte $03, $01, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $02, $01, $00, $00
.byte $00, $00, $00, $00

durationTab: ; table of note duration
.byte $C0, $E0, $F0, $A0
.byte $78, $60, $50, $45
.byte $3C, $35, $30, $2C
.byte $28, $25, $22, $20
.byte $1E, $1C, $1B, $19
.byte $18, $17, $16, $15
.byte $14, $13, $13, $12
.byte $11, $11, $10, $0F
.byte $0F, $0F, $0E, $0E
.byte $0D, $0D, $0D, $0C
.byte $0C, $0C, $0B, $0B
.byte $0B, $0B, $0A, $0A
.byte $0A, $08, $06, $05
.byte $04, $03, $02, $01
.byte $D0, $68, $B4, $5A
.byte $2D, $A4, $D2, $69

.byte $FF

.org $C680
JMP disableVoice ; 80: end of pattern
JMP makeRelease ; 83: set control for release
JMP setInstrument ; 86: set a new instrument
JMP setVolume ; 89: set the volume
JMP portUp ; 8C: portamento add
JMP portDown ; 8F: portamento down
JMP stopEffect ; 92: stop effect
JMP sidCopy ; 95: copy sid register values
JMP $0000 ; 98: not used
JMP varyAction ; 9B: various action: vibrato, gosub, return, play note

.byte $00, $00

disVoice: ; l=disable voice
.byte $00, $00, $00

flagMus: ; l=process music flow
.byte $00, $00, $00

vIndex: ; voice index
.byte $02

disMusic:
.byte $00

maxIndex: ; max index of pattern to execute

```

```

    .byte $3D, $F5, $0E
patIndex:                ; actual pattern index
    .byte $3A, $13, $0E
durationHi:              ; high part of duration of note
    .byte $00, $00, $00
duration:                 ; duration of note
    .byte $08, $80, $80
vCntOn:                  ; voice control on
    .byte $11, $85, $41
vOffset:                  ; voice offset
    .byte $00, $07, $0E
instAddr:                 ; instruction address to execute
    .byte $9B, $C6
    .byte $00
nRepeat:                  ; number of times to repeat
    .byte $02, $00, $02
temp:                     ;
    .byte $48
trackIndex:               ; actual track index
    .byte $04, $04, $04
loPatAddr:                ; low pattern index
    .byte $00, $00, $00
hiPatAddr:                ; high pattern index
    .byte $CA, $CB, $CC
actFiltVol:               ; actual filter and volume of sid
    .byte $3F
bitWiseTab:               ; a table for bitwise operation
    .byte $01, $02, $04
actFiltRes:               ; actual filter res
    .byte $F2
freqStep:                 ;
    .byte $00, $00, $00
relDelay:                 ; delay to reload
    .byte $00, $00, $00
actDelay:                 ; actual delay
    .byte $00, $00, $00
flagEff:                  ; flag effect for each voices
    .byte $01, $02, $03
freqEffect:               ;
    .byte $05, $05, $05
abilEffect:               ; flag for abilitate effects
    .byte $05
freqHi:                   ;
actFreqHi:                ; actual frequency high value
    .byte $54, $3F, $05
actFreqLo:                ; actual value of frequency low
    .byte $7D, $4B, $47
;C6E5
    .byte $00, $00, $00
cRepeat:                  ; repeat counter
    .byte $00, $00, $00
actInstr:                 ; actual instrument number
    .byte $00, $02, $01
fromIndex:                ; sid index for from
    .byte $00, $00, $00
toIndex:                  ; sid index for to
    .byte $00, $00, $00
actFiltCutFH:             ;
    .byte $48, $00, $00
storedLoPatAddr:

```

```

    .byte $00
storedHiPatAddr:
    .byte $00
storedPatIndex:
    .byte $00
storedMaxIndex:
    .byte $00
storedInstr:
    .byte $00
actFiltCut:                ; actual filter cut frequency
    .byte $00
packDuration:             ; pack duration
    .byte $A8, $A0, $A0
tbAD:                    ; attack decay table
    .byte $2A, $58, $82, $0C
    .byte $FA, $08, $09, $0C
    .byte $08, $08, $09, $0C
    .byte $3A, $09, $09, $00
tbSR:                    ; sustain release table
    .byte $3C, $5C, $50, $00
    .byte $00, $00, $4C, $50
    .byte $00, $50, $00, $5A
    .byte $4A, $00, $00, $00
tbWLo:                   ; wave low table
    .byte $0A, $7A, $00, $7A
    .byte $7A, $00, $7A, $00
    .byte $00, $50, $00, $40
    .byte $00, $00, $00, $00
tbWHi:                   ; wave high table
    .byte $06, $07, $00, $07
    .byte $08, $08, $08, $0C
    .byte $00, $08, $00, $08
    .byte $00, $00, $00, $00
tbCntOn:                 ; control voice (on) table
    .byte $11, $41, $85, $41
    .byte $41, $15, $41, $15
    .byte $11, $41, $11, $15
    .byte $81, $81, $11, $00
tbCntOff:                ; control voice (off) table
    .byte $10, $40, $84, $40
    .byte $40, $14, $40, $14
    .byte $10, $40, $10, $14
    .byte $80, $80, $10, $00
filtCutFH:               ; filter cut frequency high byte table (one values for each
instrument)
    .byte $80, $30, $70, $90
    .byte $5D, $00, $A0, $00
    .byte $00, $00, $00, $80
    .byte $90, $00, $00, $00
filterResCtr:            ; filter resonance control table for the instrument
    .byte $F0, $F0, $F0, $F0
    .byte $F0, $00, $F0, $00
    .byte $00, $00, $00, $F0
    .byte $F0, $00, $00, $00
filterMode:              ; filter mode (type) table for the instrument
    .byte $10, $10, $30, $30
    .byte $30, $00, $10, $00
    .byte $00, $00, $00, $10
    .byte $10, $00, $00, $00
;C790
    .byte $00, $00, $00, $00
    .byte $00, $00, $00, $00
    .byte $00, $00, $00, $00
    .byte $00, $00, $00, $00
    .byte $00, $00, $00, $00
    .byte $00, $00, $00, $00
    .byte $00, $00, $00, $00
    .byte $00, $00, $00, $00
currValue:                ; current value for effect
    .byte $00, $68, $13
delay:

```

```

    .byte $02, $08, $08
vCntOff:                ; voice control off
    .byte $10, $84, $40
waveHi:
    .byte $06, $00, $04
indType:                ; index type of effect
    .byte $02, $00, $01
step:
    .byte $01, $01, $01
limit:                  ; limit value
    .byte $04, $80, $20
delayRelod:
    .byte $06, $08, $0A
lowLimit:               ; low limit value
    .byte $01, $20, $08
waveHiRel:              ; wave high to reload
    .byte $06, $00, $07

    .byte $00, $00
    .byte $00, $00, $00, $00
    .byte $00, $00, $00, $00
    .byte $00, $00, $00, $00
    .byte $00, $00, $00, $00
    .byte $00, $00, $00, $00
    .byte $00, $00, $00, $00
    .byte $00, $00, $00, $00
    .byte $00, $00, $00, $00
    .byte $00, $00, $00, $00
    .byte $00, $00, $00, $00
    .byte $00, $00, $00, $00
    .byte $00, $00, $00, $00

; track format:
; high address of pattern to execute
; low address of pattern to execute
; max index in pattern to execute (FF=max, can be terminate by instruction)
; number of times to repeat the given pattern

track1:
    .byte >pat00, <pat00, $3D, $02
    .byte >pat00, <pat00, $51, $00
    .byte >pat01, <pat01, $0A, $04
    .byte >pat02, <pat02, $16, $08
    .byte >pat03, <pat03, $10, $16
    .byte >pat03, <pat03, $FF, $00
    .byte >pat0b, <pat0b, $FF, $00
    .byte >pat0c, <pat0c, $FF, $00
    .byte >pat0d, <pat0d, $FF, $00
    .byte >pat0e, <pat0e, $FF, $00
    .byte >pat0f, <pat0f, $FF, $00
    .byte >pat10, <pat10, $FF, $00
    .byte >pat11, <pat11, $FF, $00
    .byte >pat12, <pat12, $FF, $00
    .byte >pat13, <pat13, $FF, $00
    .byte >pat14, <pat14, $FF, $00
    .byte >pat15, <pat15, $FF, $00
    .byte >pat16, <pat16, $FF, $00
    .byte >pat17, <pat17, $FF, $00
    .byte >pat18, <pat18, $FF, $00
    .byte >pat26, <pat26, $FF, $00
    .byte >pat37, <pat37, $FF, $00
    .byte >pat27, <pat27, $FF, $00
    .byte $00, $00, $00, $00

track2:
    .byte >pat05, <pat05, $F5, $00
    .byte >pat0a, <pat0a, $5B, $00
    .byte >pat04, <pat04, $FF, $00
    .byte >pat19, <pat19, $FF, $00
    .byte >pat1a, <pat1a, $FF, $00
    .byte >pat1b, <pat1b, $FF, $00
    .byte >pat1c, <pat1c, $FF, $00
    .byte >pat1d, <pat1d, $FF, $00
    .byte >pat1e, <pat1e, $FF, $00
    .byte >pat1f, <pat1f, $FF, $00
    .byte >pat20, <pat20, $FF, $00
    .byte >pat21, <pat21, $FF, $00
    .byte >pat22, <pat22, $FF, $00
    .byte >pat23, <pat23, $FF, $00
    .byte >pat24, <pat24, $FF, $00
    .byte >pat25, <pat25, $FF, $00

```

```

;C8A0
track3:
.byte >pat06, <pat06, $0E, $02
.byte >pat06, <pat06, $3C, $00
.byte >pat07, <pat07, $24, $05
.byte >pat08, <pat08, $39, $05
.byte >pat07, <pat07, $24, $02
.byte >pat09, <pat09, $FF, $00
.byte >pat28, <pat28, $FF, $00
.byte >pat29, <pat29, $FF, $00
.byte >pat2a, <pat2a, $FF, $00
.byte >pat2b, <pat2b, $FF, $00
.byte >pat2c, <pat2c, $FF, $00
.byte >pat2d, <pat2d, $FF, $00
.byte >pat2e, <pat2e, $FF, $00
.byte >pat2f, <pat2f, $FF, $00
.byte >pat30, <pat30, $FF, $00
.byte >pat31, <pat31, $FF, $00
.byte >pat32, <pat32, $FF, $00
.byte >pat33, <pat33, $FF, $00
.byte >pat34, <pat34, $FF, $00
.byte >pat35, <pat35, $FF, $00
.byte >pat36, <pat36, $FF, $00

```

```

;C8F4
.byte $00, $00, $00, $00
.byte >pat36, <pat36, $FF, $00
.byte $00, $00, $00, $00

```

```

;C900
pat38:
.byte $86, $0D, $00 ; set instrument
.byte $23, $D5
.byte $9B, $05, $23 ; play note
.byte $86, $0E, $00 ; set instrument
.byte $9B, $05, $5A ; play note
.byte $D6
.byte $9B, $05, $5A ; play note
.byte $86, $0D, $00 ; set instrument
.byte $9B, $05, $23
.byte $D5
.byte $9B, $05, $23 ; play note
.byte $86, $0E, $00 ; set instrument
.byte $9B, $05, $5A ; play note
.byte $D6
.byte $9B, $05, $5A ; play note
.byte $86, $0F, $00 ; set instrument
.byte $00, $D0
.byte $9B, $03 ; restore state
.byte $00
.byte $00, $00

```

```

;C930
pat39:
.byte $86, $0D, $00 ; set instrument
.byte $26, $D5
.byte $9B, $05, $23 ; play note
.byte $86, $0E, $00 ; set instrument
.byte $9B, $05, $5A ; play note
.byte $D6
.byte $9B, $05, $5A ; play note
.byte $86, $0D, $00 ; set instrument
.byte $9B, $05, $26 ; play note
.byte $D5
.byte $9B, $05, $23 ; play note
.byte $86, $0E, $00 ; set instrument
.byte $9B, $05, $5A ; play note
.byte $D6
.byte $9B, $05, $5A ; play note
.byte $86, $0F, $00 ; set instrument
.byte $00, $D0
.byte $9B, $03 ; restore state

.byte $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00

```

```

.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00

```

```

pat00:
;CA00
.byte $86, $00, $00           ; set instrument
.byte $9B, $02, $02, $04, $01, $06
.byte $83                     ; set control for release
.byte $83                     ; set control for release
.byte $2A
.byte $A8                     ; note duration
.byte $15, $11, $13
.byte $2A, $15, $11, $13
.byte $2A, $15, $11, $13
.byte $2A, $15, $11, $13
.byte $26, $15, $11, $13
.byte $26, $15, $11, $13
.byte $26, $15, $11, $13
.byte $26, $15, $11, $13
.byte $25, $15, $11, $13
.byte $25, $15, $11, $13
.byte $25, $15, $11, $13
.byte $25, $15, $11, $13
.byte $25, $15, $11, $13
.byte $A8                     ; note duration
.byte $2A
.byte $83                     ; set control for release
.byte $00, $D8               ; note duration
.byte $00, $DA               ; note duration
.byte $00, $A0               ; note duration
.byte $00, $A0               ; note duration

```

```

pat01:
;CA47
.byte $00, $A8               ; note duration
.byte $9B, <pat39           ; gosub instruction
.byte $00, $DB               ; note duration
.byte $9B, <pat39           ; gosub instruction
.byte $9B, <pat38           ; gosub instruction
.byte $92                   ; stop effect
.byte $92                   ; stop effect

```

```

pat02:
;CA53
.byte $86, $05, $00         ; set instrument
.byte $9B, $02, $02, $04, $01, $06
.byte $1A, $B0, $18         ; note + note duration + note
.byte $9B, <pat39           ; gosub instruction
.byte $1A, $A8               ; note + note duration
.byte $00, $B0               ; note duration
.byte $9B, <pat39           ; gosub instruction
.byte $9B, <pat38           ; gosub instruction

```

```

pat03:
;CA69
.byte $86, $05, $00         ; set instrument
.byte $1A, $B0, $18         ; note + note duration + note
.byte $9B, <pat39           ; gosub instruction
.byte $1A, $A8               ; note + note duration
.byte $9B, <pat38           ; gosub instruction
.byte $9B, <pat39           ; gosub instruction
.byte $9B, <pat38           ; gosub instruction
.byte $86, $06, $00         ; set instrument
.byte $3A, $A1               ; note + note duration
.byte $83                   ; set control for release

```

```

.byte $9B, $05, $36 ; play note
.byte $9B, $05, $4A ; play note
.byte $9B, $05, $46 ; play note
.byte $3A, $A2 ; note + note duration
.byte $83 ; set control for release
.byte $92 ; stop effect
;CAB8
.byte $80, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
;CAB8
.byte $16, $26, $21, $26
.byte $28, $11, $16, $26
.byte $C0, $80, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
;pat04:
;CAE0
.byte $86, $02, $01 ; set instrument
.byte $9B, $02, $00, $80, $01, $08
.byte $25, $A0 ; note + note duration
.byte $9B, $05, $21 ; play note
.byte $9B, $05, $2A ; play note
.byte $83 ; set control for release
.byte $92 ; stop effect
.byte $80 ; end of pattern
;CAF4
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
;pat05:
;CB00
.byte $89, $0F ; set volume
.byte $86, $02, $01 ; set instrument
.byte $9B, $02, $00, $80, $01, $08
.byte $25, $A0 ; note + note duration
.byte $9B, $05, $21 ; play note
.byte $9B, $05, $2C ; play note
.byte $83 ; set control for release
.byte $86, $03, $01 ; set instrument
.byte $9B, $02, $01, $20, $01, $01
.byte $31, $A1, $4A
.byte $A8, $3A, $A4, $38
.byte $A8, $35, $A2, $46
.byte $A1, $4A, $A8, $48
.byte $31, $4A, $A2, $4A
.byte $A8, $45, $A1, $41
.byte $C0, $43, $45, $43
.byte $45, $46, $48, $46
.byte $48, $4A, $4C, $4A
.byte $4C, $31, $33, $35
.byte $33, $35, $36, $38
.byte $3A, $38, $3A, $A4
.byte $00, $B0 ; note duration
.byte $4A, $A1 ; note + note duration
;CB50
.byte $86, $02, $01 ; set instrument
.byte $9B, $02, $00, $40, $01, $08
.byte $83 ; set control for release
.byte $83 ; set control for release
.byte $25, $A1 ; note + note duration
.byte $9B, $05, $21 ; play note
.byte $A0
.byte $9B, $05, $2C ; play note
.byte $A0
.byte $9B, $05, $25 ; play note
.byte $A0
.byte $9B, $05, $2C ; play note
.byte $9B, $05, $11 ; play note
.byte $9B, $02, $02, $04, $01, $04
.byte $86, $03, $00 ; set instrument
.byte $2A, $A2, $23, $A8
.byte $21, $25, $A4, $2A
.byte $A4, $26, $D9, $00

```



```

.byte $A2, $2A, $A4, $25
.byte $A8, $11, $B0, $2C
.byte $2A, $A4, $21, $A8
.byte $11, $B0, $2C, $2A
.byte $A2, $11, $B0, $2C
.byte $11, $2C, $11, $2C
.byte $28, $26, $2A, $A4
.byte $11, $B0, $15, $1A
.byte $A2, $2A, $A8, $25
.byte $A8, $21, $C0, $15
.byte $21, $16, $21, $18
.byte $21, $1A, $21, $01
.byte $A8, $01, $C0, $1A
.byte $01, $18, $01, $16
.byte $01, $15, $01, $13
.byte $01, $11, $DA, $2A
.byte $A8, $13, $B0, $2A
.byte $15, $2A, $18, $2A
.byte $A4, $25, $C0, $23
.byte $21, $23, $25, $28
.byte $2A, $B0, $11, $13
.byte $15, $18, $1A, $A0
.byte $00, $B0                ; note duration
.byte $86, $01, $00           ; set instrument
.byte $11, $A4, $11
.byte $A8, $2A, $25, $B0
.byte $21, $A8, $11, $B0
.byte $2A, $25, $2A, $11
.byte $2A, $25, $23, $21
.byte $A4
.byte $80                        ; end of pattern

.byte $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00

pat06:
;CC00
.byte $86, $01, $00           ; set instrument
.byte $9B, $02, $01, $20, $01, $0A
.byte $5A, $A0, $56
.byte $55, $A0, $5A, $A8
.byte $83                        ; set control for release
.byte $00, $D8
.byte $86, $04, $00           ; set instrument
.byte $00
.byte $DA

pat07:
;CC18
.byte $4A, $A8, $3A, $B0
.byte $4A, $DB, $3A, $A8
.byte $4A, $A8, $3A, $B0
.byte $4A, $DB, $3A, $A8
.byte $83                        ; set control for release
.byte $86, $01, $00           ; set instrument
.byte $46, $A8, $36, $B0
.byte $46, $DB, $36, $A8
.byte $46, $A8, $36, $B0
.byte $46, $DB, $36, $A8

pat08:
;CC3C
.byte $4A, $A8, $3A, $C0
.byte $3A, $38, $B0, $31
.byte $4A, $3A, $C0, $4A
.byte $3A, $4A, $4A, $A8
.byte $3A, $C0, $3A, $38
.byte $B0, $31, $4A, $3A
.byte $C0, $4A, $3A, $4A
.byte $46, $A8, $36, $C0
.byte $36, $31, $B0, $4A
.byte $46, $36, $C0, $46
.byte $36, $46, $46, $A8
.byte $36, $C0, $36, $31
.byte $B0, $4A, $46, $36
.byte $C0, $46, $36, $46
.byte $C0

pat09:
;CC75
.byte $5A, $A2
.byte $83                        ; set control for release
.byte $92                        ; stop effect
.byte $80                        ; end of pattern

.byte $00, $00
.byte $00, $80, $00, $00
.byte $00, $00, $00, $00

```

```

.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00

pat0a:
;CCA0
.byte $2A, $B0, $2C, $C0
.byte $11, $13, $15, $1A
.byte $B0, $01, $03, $05
.byte $08, $A8
.byte $86, $07, $00 ; set instrument
.byte $0A, $A0
.byte $86, $03, $01 ; set instrument
.byte $9B, $02, $01, $20, $01, $01
.byte $1A, $A4, $15, $B0
.byte $18, $A4, $2A, $15
.byte $B0, $18, $1A, $A4
.byte $15, $A8, $11, $A4
.byte $01, $A8, $1A, $A4
.byte $13, $B0, $11, $A8
.byte $1A, $B0, $15, $A8
.byte $2A, $C0, $2B, $2C
.byte $11, $12, $13, $14
.byte $15, $16, $17, $18
.byte $19, $1A, $A4, $18
.byte $A8, $15, $13, $B0
.byte $11, $A8, $2A, $B0
.byte $25, $2A, $A8, $25
.byte $B0, $21, $25, $A8
.byte $2A, $A8
.byte $83 ; set control for release
.byte $80 ; end of pattern

.byte $00, $00, $00, $00

pat0b:
;CD00
.byte $89, $0F ; set the volume
.byte $86, $08, $00 ; set instrument
.byte $2A, $D5, $1A
.byte $11, $01
.byte $83 ; set control for release
.byte $80 ; end of pattern

pat0c:
;CD0C
.byte $89, $0F ; set the volume
.byte $86, $07, $00 ; set instrument
.byte $8F, $FF, $01
.byte $31, $C0
.byte $83 ; set control for release
.byte $92 ; stop effect
.byte $80 ; end of pattern

pat0d:
;CD19
.byte $89, $0F ; set the volume
.byte $86, $09, $00 ; set instrument
.byte $8C, $80 ; portamento add
.byte $01, $41, $B4
.byte $92 ; stop effect
.byte $00, $B0
.byte $8F, $80 ; portamento down
.byte $01, $00, $B4
.byte $92 ; stop effect
.byte $83 ; set control for release
.byte $80 ; end of pattern

pat0e:
;CD2E
.byte $89, $0F ; set the volume
.byte $86, $03, $00 ; set instrument
.byte $9B, $02, $01, $10, $01, $05
.byte $31, $A8, $83
.byte $92 ; stop effect
.byte $80 ; end of pattern

pat0f:
;CD3E
.byte $89, $0F ; set the volume
.byte $86, $03, $00 ; set instrument
.byte $9B, $02, $01, $10, $01, $05
.byte $33, $A8
.byte $83 ; set control for release
.byte $92 ; stop effect

```

```

    .byte $80                ; end of pattern

pat10:
;CD4E
    .byte $89, $0F          ; set the volume
    .byte $86, $03, $00     ; set instrument
    .byte $9B, $02, $01, $10, $01, $05
    .byte $35, $A8
    .byte $83                ; set control for release
    .byte $92                ; stop effect
    .byte $80                ; end of pattern

pat11:
;CD5E
    .byte $89, $0F          ; set the volume
    .byte $86, $03, $00     ; set instrument
    .byte $9B, $02, $01, $10, $01, $05
    .byte $36, $A8
    .byte $83                ; set control for release
    .byte $92                ; stop effect
    .byte $80                ; end of pattern

pat12:
;CD6E
    .byte $89, $0F          ; set the volume
    .byte $86, $0A, $00     ; set instrument
    .byte $9B, $02, $02, $10, $01, $03
    .byte $21, $B0
    .byte $83                ; set control for release
    .byte $92                ; stop effect
    .byte $80                ; end of pattern

pat13:
;CD7E
    .byte $89, $0F          ; set the volume
    .byte $86, $0A, $00     ; set instrument
    .byte $2A
    .byte $D3, $1A, $2A, $26
    .byte $2A, $1A
    .byte $83                ; set control for release
    .byte $80                ; end of pattern

pat14:
;CD8C
    .byte $89, $0F          ; set the volume
    .byte $86, $08, $00     ; set instrument
    .byte $11, $D0, $1A
    .byte $83                ; set control for release
    .byte $80                ; end of pattern

pat15:
;CD96
    .byte $89, $0F          ; set the volume
    .byte $9B, $02, $02, $10, $05, $02
    .byte $86, $05, $00     ; set instrument
    .byte $21, $C0, $25
    .byte $2A
    .byte $83                ; set control for release
    .byte $92                ; stop effect
    .byte $80                ; end of pattern

pat16:
;CDA8
    .byte $89, $0A          ; set the volume
    .byte $86, $0B, $00     ; set instrument
    .byte $21, $A2
    .byte $9B, $05, $21
    .byte $A8
    .byte $83                ; set control for release
    .byte $92                ; stop effect
    .byte $80                ; end of pattern

pat17:
;CDB6
    .byte $89, $0F          ; set the volume
    .byte $86, $0C, $00     ; set instrument
    .byte $9B, $02, $02, $08, $0A, $01
    .byte $21, $DB
    .byte $83                ; set control for release
    .byte $00, $A4
    .byte $86, $0F, $00     ; set instrument
    .byte $80                ; end of pattern

pat18:
;CDCA
    .byte $89, $0F          ; set the volume
    .byte $86, $03, $00     ; set instrument
    .byte $9B, $02, $01, $10, $01, $05
    .byte $3A, $C0, $35

```

```

.byte $33, $3A, $35, $33
.byte $3A, $35, $33, $3A
.byte $B0, $3A
.byte $83 ; set control for release
.byte $00
.byte $A8
.byte $80 ; end of pattern

.byte $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00
.byte $00, $00, $00, $00

pat19:
;CE00
.byte $86, $08, $00 ; set instrument
.byte $21
.byte $D5, $11, $25, $15
.byte $83 ; set control for release
.byte $80 ; end of pattern

pat1a:
;CE0A
.byte $86, $03 ; set the volume
.byte $00, $41, $C0
.byte $83 ; set control for release
.byte $80 ; end of pattern

pat1b:
;CE11
.byte $86, $02, $00 ; set instrument
.byte $8C, $80, $01
.byte $41
.byte $B4
.byte $92 ; stop effect
.byte $00, $B0
.byte $8F, $80, $01
.byte $00
.byte $B4
.byte $92 ; stop effect
.byte $83 ; set control for release
.byte $80 ; end of pattern

pat1c:
;CE24
.byte $86, $03, $00 ; set instrument
.byte $9B, $02, $01, $10, $01, $05
.byte $31, $A8
.byte $83 ; set control for release
.byte $80 ; end of pattern

pat1d:
;CE31
.byte $86, $03, $00 ; set instrument
.byte $9B, $02, $01, $10, $01, $05
.byte $33, $A8
.byte $83 ; set control for release
.byte $80 ; end of pattern

pat1e:
;CE3E
.byte $86, $03, $00 ; set instrument
.byte $9B, $02, $01, $10, $01, $05
.byte $35
.byte $A8
.byte $83 ; set control for release
.byte $80 ; end of pattern

pat1f:
;CE4B
.byte $86, $03, $00 ; set instrument
.byte $9B, $02, $01, $10, $01, $05
.byte $36, $A8
.byte $83 ; set control for release
.byte $80 ; end of pattern

pat20:
;CE58
.byte $86, $05, $00 ; set instrument
.byte $11
.byte $C0, $15
.byte $83 ; set control for release
.byte $80 ; end of pattern

pat21:
;CE60

```

```

    .byte $86, $0A, $00          ; set instrument
    .byte $9B, $02, $02, $10, $05, $02
    .byte $21, $B0
    .byte $83                    ; set control for release
    .byte $80                    ; end of pattern

pat22:
;CE6D
    .byte $86, $08, $00          ; set instrument
    .byte $21, $D0, $2A
    .byte $83                    ; set control for release
    .byte $80                    ; end of pattern

pat23:
;CE75
    .byte $86, $05, $00          ; set instrument
    .byte $11, $B0, $00, $C0
    .byte $83                    ; set control for release
    .byte $80                    ; end of pattern

pat24:
;CE7E
    .byte $86, $0B, $00          ; set instrument
    .byte $9B, $02, $02, $08, $03, $04
    .byte $11
    .byte $A2
    .byte $9B, $05, $11          ; play note
    .byte $A8
    .byte $83                    ; set control for release
    .byte $00, $B0
    .byte $92                    ; stop effect
    .byte $80                    ; end of pattern

pat25:
    .byte $86, $0C, $00          ; set instrument
    .byte $9B, $02, $02, $10, $0A, $02
    .byte $21
    .byte $DB
    .byte $83                    ; set control for release
    .byte $00, $A4
    .byte $86, $0F, $00          ; set instrument
    .byte $80                    ; end of pattern

pat26:
CEA4
    .byte $86, $03, $00          ; set instrument
    .byte $9B, $02, $01, $20, $01, $05
    .byte $2A, $C0, $25
    .byte $23, $2A, $25, $23
    .byte $2A, $25, $23, $2A
    .byte $B0, $2A
    .byte $83                    ; set control for release
    .byte $00
    .byte $A8
    .byte $80                    ; end of pattern

pat27:
;CEBE
    .byte $86, $00, $00          ; set instrument
    .byte $9B, $02, $02, $04, $01, $04
    .byte $2A
    .byte $A8, $21, $B0, $25
    .byte $A4, $25, $A8, $21
    .byte $B0, $3A, $A8
    .byte $83                    ; set control for release
    .byte $00, $A2
    .byte $80                    ; end of pattern
    .byte $00
    .byte $00, $00, $00, $00
    .byte $00, $00, $00, $00
    .byte $00, $00, $00, $00
    .byte $00, $11, $21, $11
    .byte $83                    ; set control for release
    .byte $80                    ; end of pattern
    .byte $00, $00
    .byte $00, $00, $00, $00
    .byte $00, $00, $00, $00
    .byte $00, $00, $00, $00
    .byte $00, $00, $00, $00
    .byte $00, $00, $00, $00

pat28:
;CF00
    .byte $86, $08, $00          ; set instrument
    .byte $21
    .byte $D5, $11, $25, $15
    .byte $83                    ; set control for release
    .byte $80                    ; end of pattern

```

```

pat29:
;CF0A
.byte $86, $08, $00           ; set instrument
.byte $25, $C0
.byte $80                     ; end of pattern

pat2a:
;CF10
.byte $86, $07, $00           ; set instrument
.byte $9B, $02, $02, $04, $0A, $04
.byte $31, $AA, $00
.byte $B0
.byte $83                     ; set control for release
.byte $80
.byte $00                     ; end of pattern

pat2b:
;CF20
.byte $86, $03, $00           ; set instrument
.byte $9B, $02, $01, $10, $01, $05
.byte $31, $A8
.byte $83                     ; set control for release
.byte $80                     ; end of pattern

pat2c:
;CF2D
.byte $86, $03, $00           ; set instrument
.byte $9B, $02, $01, $10, $01, $05
.byte $33, $A8
.byte $83                     ; set control for release
.byte $80                     ; end of pattern

pat2d:
;CF3A
.byte $86, $03, $00           ; set instrument
.byte $9B, $02, $01, $10, $01, $05
.byte $35
.byte $A8
.byte $83                     ; set control for release
.byte $80                     ; end of pattern

pat2e:
;CF47
.byte $86, $03, $00           ; set instrument
.byte $9B, $02, $01, $10, $01, $05
.byte $36, $A8
.byte $83                     ; set control for release
.byte $80                     ; end of pattern

pat2f:
;CF54
.byte $86, $0A, $00           ; set instrument
.byte $9B, $02, $02, $04, $0A, $02
.byte $21, $B0
.byte $83                     ; set control for release
.byte $92                     ; stop effect
.byte $80                     ; end of pattern

pat30:
;CF62
.byte $86, $05, $00           ; set instrument
.byte $35, $D0, $31
.byte $4A
.byte $83                     ; set control for release
.byte $80                     ; end of pattern

pat31:
;CF6B
.byte $86, $05, $00           ; set instrument
.byte $11, $D3
.byte $01, $1A, $0A
.byte $83                     ; set control for release
.byte $80                     ; end of pattern

pat32:
.byte $86, $05, $00           ; set instrument
.byte $9B, $02, $02, $10, $05, $02
.byte $21, $B0, $2A, $C0
.byte $83                     ; set control for release
.byte $80                     ; end of pattern

pat33:
;CF84
.byte $86, $03, $00           ; set instrument
.byte $8C                     ; portamento add
.byte $10, $00, $21, $A2
.byte $92                     ; stop effect
.byte $00, $A8
.byte $83                     ; set control for release

```

```

    .byte $80                ; end of pattern

pat34:
;CF91
    .byte $86, $0B, $00      ; set instrument
    .byte $95, $1B, $0F      ; put osc3 to freq. hi
    .byte $83
    .byte $83                ; set control for release
    .byte $83                ; set control for release
    .byte $51, $DB
    .byte $83                ; set control for release
    .byte $00, $A4
    .byte $86, $0F, $00      ; set instrument
    .byte $80                ; end of pattern

pat35:
;CFA3
    .byte $86, $03, $00      ; set instrument
    .byte $9B, $02, $00, $40, $05, $02
    .byte $83                ; set control for release
    .byte $5A, $B0, $51, $C0
    .byte $58, $B0, $51, $C0
    .byte $5A, $B0, $55, $C0
    .byte $5A, $B0, $5A
    .byte $83                ; set control for release
    .byte $80                ; end of pattern

pat36:
;CFBE
    .byte $86, $00, $00      ; set instrument
    .byte $95, $1C, $16      ; put asdr3 to filter freq hi
    .byte $4A, $A8, $41, $A8
    .byte $5A, $A4, $4A, $B0
    .byte $3A
    .byte $83                ; set control for release
    .byte $00, $A2
    .byte $80                ; end of pattern
    .byte $00, $00, $00
    .byte $00, $00, $00, $00

pat37:
;CFD8
    .byte $89, $0F          ; set volume
    .byte $86, $00, $00      ; set instrument
    .byte $2A, $A8, $21
    .byte $B0, $25, $A2
    .byte $83                ; set control for release
    .byte $80                ; end of pattern
    .byte $00, $00, $00

playMusic:
    jsr $C000
    jsr $C000
    rts

    .byte $00, $00, $00, $00
    .byte $00, $00, $00, $00
    .byte $00, $00, $00, $00
    .byte $00, $00, $00, $00
    .byte $00

```

## Conclusion

The engine is not so powerful compared to some player of the same year, but probably only the features needed by Ivan were implemented into the player.

However we know that wonderful sid music can be produced even with simple player (look at "Lazy Jones" in SIDin #2) and so I think that with this player, Modulus is resulted very-very good.

# xxlarge

by Stefano Tognon <[ice00@libero.it](mailto:ice00@libero.it)>

xxlarge is a tune created by Ninja/theDream is less than 256 byte. Before reading this article, goes to download and listen to it! (<http://noname.c64.org/csdb/release/?id=14337>)

The tune is a cover of Jeff's "X large" and you will note that it is played a little slower that the original.

Now I think you are wondering how music and code can be fitted in so little space, as Jeff original is large about 800 bytes!

Here I go to analyze the code to show how this can be made and finally I hope that one of you would like to take the challenge to produce a 256/512b tune (but this will be explained later).

## Code

Here the complete code that were released together with the music.

```
include standard.c64

mus_pnt      = $17      ; $16
mus_filtercnt = $39      ; $05 (basic line lo)
mus_vlpos    = $3b
mus_v2pos    = mus_vlpos+7
mus_v3pos    = mus_vlpos+14
pattern_idx  = $3d      ; $0b
mus_bufferv23 = $b3      ; $03
mus_pulsecnt = $cd      ; Cursor Blink Cnt :)

org $0801

adr $080c, $05
byt $97, "789,8"           ; POKE

pattern_list:              ; $08 = 8
    byt pattern3, pattern2
    byt pattern3, pattern2
    byt pattern1, pattern1
    byt pattern5, pattern4

pulse_tab:                 ; $14 = 20
    byt $07,$47,$87,$B7,$08,$38,$88,$B8
    byt $09,$39,$89,$49,$09,$B8,$88,$38
    byt $08,$B7,$87,$37

SR_tab:                    ; 3 + 2*6 = 15
    byt $d7
filter_cut:                byt $60,$78,$90,$a8,$c0,$DB
filter_res:                byt $77
                            byt $01,$41,$71,$a1,$e1,$1f
                            byt $38

mus_irq:
    ldx mus_pulsecnt
    lda pulse_tab - 1,x
    STA $D403
    STA $D402
    asl
    bpl mus_exit

    LDA mus_bufferv23
    STA mus_v3pos
    LDA mus_v2pos
    STA mus_bufferv23
    LDA mus_vlpos
    STA mus_v2pos
    byt $2c

mus_filterchange:         ; c=0
    ror mus_filtercnt     ; inc <-> ror
```

IRQ activation

BIT absolute instruction



```

        ldy mus_filtercnt
        LDA filter_res,y
        STA $D418 - 5,y
        STA $D417
get_nybble:
        lda mus_pnt
        inc mus_pnt
        lsr
        tax
        lda patterns,x
        bne do_note

        dec pattern_idx
        lda pattern_idx
        lsr
        and #7
        tax
        lda #$66<<1
        ror
        sta mus_filterchange
        lda pattern_list,x
        sta mus_pnt
        txa
        ; fall through to get_nybble

do_note:
        bcs mus_getlo
        lsr
        lsr
        lsr
        lsr

mus_getlo:
        and #$0f
        STA mus_vlpos
        asl
        beq mus_filterchange
        adc filter_cut,y
        STA $D416

loop_base = $100 - 3*7
        ldy #loop_base
        lda #$41
mus_11:
        sta $d404 - loop_base,y
        lda SR_tab - loop_base,y
        sta $d406 - loop_base,y
        ldx lo(mus_vlpos - loop_base),y
        LDA notes_lo - 2,X
        STA $D400 - loop_base,y
        LDA notes_hi - 2,X
        STA $D401 - loop_base,y
        tya
        adc #7
        tay
        lda #$11
        bcc mus_11

mus_exit:
        if 0
        lda mus_filtercnt
        sta $0400
        lda mus_pnt
        sta $0402
        lda pattern_idx
        and #15
        ldx mus_pulsecnt
        sta $0404,x
        endif
        jmp $ea31

notes_lo:
        ; $0e = 14
        byt $5a,$ce,$c1,$b4,$9c,$09,$d0,$82
        byt $68,$88,$af,$39,$13,$a1
notes_hi:
        ; $0e = 14
        byt $04,$05,$07,$08,$0b,$0d,$0d,$0f
        byt $11,$13,$14,$17,$1a,$1b

patterns = *
pattern3 = (* - patterns) * 2 ; $0b = 11
pattern5 = pattern3 + 6
        byt $03*$10+$05,$06*$10+$07,$00*$10+$08
        byt $03*$10+$0a,$06*$10+$00,$0d*$10+$03,$0c*$10+$0b
        byt $06*$10+$09,$03*$10+$0a
        byt 0
pattern1 = (* - patterns) * 2 ; $0a = 10
        byt $02*$10+$00,$05*$10+$04,$02*$10+$00,$05*$10+$02
        byt $0a*$10+$05,$00*$10+$09,$02*$10+$05,$0a*$10+$05
        byt $04*$10+$02,$0a*$10+$02
        byt 0
pattern2 = (* - patterns) * 2 ; $09 = 9
        byt $02*$10+$05,$07*$10+$0b,$0c*$10+$02,$0d*$10+$05

```

cycle throw patterns

Insert DEC (ROR)

Voice 1 rectangular

Voice 2/3 triangular

```

        byt $0f*$10+$02,$0e*$10+$0c,$05*$10+$0b,$02*$10+$0a
        byt 0
pattern4 = (* - patterns) * 2          ; $0a = 10
        byt $02*$10+$05,$00*$10+$04,$02*$10+$00,$05*$10+$02
        byt 0
pattern_end = (* - patterns) * 2
        byt "NINJA"
end $0801

```

## Analysis

One of the first things that a player must do is to add a IRQ handler after it has initialize the engine. The IRQ has to call the player routine at each programmed interval (like by CIA or VIC).

This is a task that require many bytes, but we have to use the less we can, so look at this heavy optimized peaces of code:

- The player did not initialize itself, or in other words, it did not use an init routine, as all that is needed is done inside the play routine (and with variables already initialized).
- The IRQ initialization use the BASIC instruction: POKE 789,8

The last is a very optimized way to initialize IRQ.

The address 788/789 contains the Hardware Interrupt vector that standard Kernal IRQ routine will give control during an IRQ call. It is very common to change this address to assign an self made IRQ routine. In this case, the high byte of the IRQ routine is assigned to 08.

What this means? Simple, as normally the IRQ vector is at \$EA31, with this instruction it will be located to \$0831. In other words, the IRQ routine *mus\_irq* must be starting at \$0831 and we don't need anymore BASIC instructions to go away. If you look at the source, before the \$0831 address, we have lot of bytes used by the player for patterns and tables.

Before going into further analysis of the source we have to say that the player as these features:

- The player use a track for the music: *pattern\_list*
- The track is composed by a sequences of patterns
- Each pattern contains sequence of note and filter command (packed in one byte)
- All 3 voices are used
- The player has a wave and filter table for dynamic effects onto the voices

## In depth

Now we start to look at the end of the source from *loop\_base = \$100 - 3\*7*

This is a cycle that:

- Initialize voice control register to rectangular for voice 1 and triangular for voice 2 and 3
- Set the Sustain/Release of each voice
- Set the high/low byte of frequency for the current note to play

All is similar to *sta \$d404 - loop\_base,y* with y equals to *loop\_base* at beginning and then it is increased by 7 (for next voices), so we have:

$\$D404 - \text{loop\_base} + \text{loop\_base} = \$D404$

The loop will end when adding 7 to *y* will make the carry positive, and as *y* starts from  $\$100-3*7$ , this will occur after we finish to pass the 3<sup>o</sup> voice.

As we see, voice 1 is rectangular, so it must be set his wave pulse values. This is done at the beginning of the routine. High and low value of wave are the same, but values are taken from a table, so they are not fixed. The nice idea for saving space, is to use the value in *\$cd* as index for the table. This is the values used for making the cursor blinking animation by the Kernal. Its value goes from  $\$14$  to  $\$1$ .

The high bit of the wave value read from the table is then used for skipping or not the rest of the player: this give some delay in calling the player.

What is missing into the player is that it not set the Attack/Decay value of each voices: according to Ninja there where no more space to add this features.

Now goes to see from *get\_nibble*: here current pointer to music notes (*mus\_pnt*) is incremented and then a pattern value is read using the pointer. A 0 values means that the pattern is ended and so the *pattern\_idx* is decremented and a new pattern is taken from the track (*mus\_pnt* is also initialized to the right place. Note that *pattern\_idx* is always decremented as the track is composed by 8 patterns, only the 3 lower bits are used.

Each pattern values are composed by two nibbles: each nibble has this meaning:

- $\$00$  change *mus\_filtercnt*
- $\$01$  not used
- $\$02-\$0f$  notes index (from tables)

So, goes to the *mus\_filterchange* label: before this there is a BIT instruction (opcode  $\$2C$ ): this means that each time the player is executed, no action is taken to the *current mus\_filtercnt*, and so the same value from table is putted to the filter.

Instead, as soon as a nibble is zero, (look after *mus\_getlo*) the flow is passed to the *mus\_filterchange* position and so the current filter position is changed.

In fact at *mus\_filterchange* we can have these instructions:

- *inc mus\_filtercnt*
- *ror mus\_filtercnt*

and so filter index table is incremented and decremented (*ror* is shorted than *dec*, but according to Ninja it is still good enough).

The instructions that made *inc* to *ror* are:

```
lda # $\$66<<1$ 
ror
sta mus_filterchange
```

If Carry is 0 a  $\$66$  (*ror*) is putted into *mus\_filterchange*, while if Carry is 1,  $\$E6$  (*inc*) is putted.

Finally, each patterns are played twice, so during the first time the filter table is incremented, in the second it is decremented.

In the source if you made compile the “if 0” statement, you will have a look of how *mus\_filtercnt* behaves.

## Conclusion

I hope that you have found interesting the analysis of this wonderful piece of code, so why not try yourself?

Well, this is why I'm near to organize the "Tiny Sid Competition": see what coder/musician are able to produce with less space.

"Tiny Sid Competition" will be executed from 15 January 2005 to 21 April 2005 instead of the "SidWine compo" (sorry, but SidWine compo take me too many months to organize, so I go for something less time consuming this year).

Allowed sizes will be:

- 256 bytes
- 512 bytes

Maybe a size of 1K could be another chance if required, but I think that 1KB are big enough, even if it could be interesting too.

Allowed will be all tunes that fit in that categories as prg executable (e.g. they must be runnable with run after loading) and they can be even cover/remix of existing tunes (cover with less byte is hard in every case).

Reverse engineering source code (or better original source) may be allowed by the author to be published in an article like this in SIDin.

I don't know how many people could be interesting in such compo, but if we did not try...

Check: <http://digilander.iol.it/ice00/tsid/tinysid>

Finally I like to thank Ninja for allowing me to write this article about his work and to have give some in-depth description of the engine.

*QED*  $\square$  *end*