# Ice Din 14

**“Francesca”**

**Vice snapshot with Vice palette**

**Made with the GIMP from a FC photo**
**and converted to C64 320x200**
**Hires Mode Bitmap**
**by Stefano Tognon**
**in 2009**

**“Lighting night”**
**...**

Team Ice **Free Software Group**

*Sovereign 14*
version 1.00
25 January 2015

# SIDin Contents

## General Index

# Editorials

Stefano Tognon <ice00@libero.it>

Hi, again.

It is incredible, but the last issue of SIDin was released more than 4 years ago!

This long times did not means that I miss to works on SID related task:

- My SID tracker JITT64 has improved a lot in the developing version and only some little adjustment is needed before release it to the public.
- XSidPlay2 is completing the porting to QT4 by rewriting most of the graphics that is still in QT3 compatibility level.
- HVMEC (High Voltage Music Engine Collection) is still being ported from scratch to a dynamic site: hvmec.altervista.org from the old static site. The process is now at 73% of completion. When this phase will be completed, lot of new programs are in the queue to be added!
- Ice Team has now a new site that is being created: iceteam.altervista.org
  The old still remain active but it will be less updated when the new will be completed.

The news inserted in this number is not so fresh as many times has passed, but there is A NEW. Yes A NEW that over-class all the other: Matt Gray is again in the scene!!

This was the dream of all sid fans, but now it is true :)

In this number we will see the Matt Gray editor he has released for his Reformation project and then we will analyze the Martin Walker sound player used into Hunter's Moon game.

Bye
S.T.

# News

Some various news of players, programs, and competitions:

- CGSC v1.18
- JSIDPlay2 2.0
- HVSC #54
- CGSC v1.19
- CGSC v1.20
- SIDKnown v1.03
- CGSC v1.21
- CGSC v1.23
- HVSC #58
- HVSC #59
- HVSC #60
- CGSC v1.27
- CGSC v1.28
- CGSC v1.29
- The Piano Collection
- Goat Tracker 2SID Mono
- ACID 64 Player Pro v3.04
- SIDwinder 1.23 Enhanced!!
- HVSC #55
- HVSC #56
- SIDply/w 2.6
- CGSC v1.22
- HVSC #57
- CGSC v1.24
- CGSC v1.25
- CGSC v1.26
- HVSC #61
- HVSC #62
- Reformation

## CGSC v1.18

The Compute's Sid Collection has just been updated on October 2010.

The new release contains an additional 182 MUS, 176 STR & 106 WDS files and the grand totals are now 8580 MUS, 2142 STR & 2737 WDS files.

Download from www.c64music.co.uk

## Goat Tracker 2SID Mono

It is just a simple hack of Goatracker 2.70 made by Raf that output mono sound of the 2 SIDs.

Get it from: http://noname.c64.org/csdb/release/?id=94423

## JSIDPlay2 2.0

On September 2010, after 3 years from the last version, JSIDPlay2 was released.

Get it from: http://sourceforge.net/projects/jsidplay2/

## ACID 64 Player Pro v3.04

A new version of ACID 64 Player Pro has been released. It can now play SID tunes via JSid-play2 and JSidDevice.


Download from http://www.acid64.com

## HVSC #54

Released on Christmas 2010 the High Voltage Sid Collection update 54.

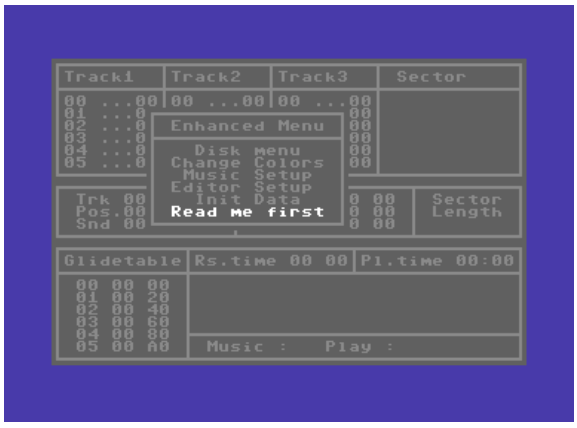
After this update, the collection should contain 39,626 SID files!

```
This update features (all approximates):
     933 new SIDs
     217 fixed/better rips
       5 PlaySID/Sidplay1 specific SIDs eliminated
      21 repeats/bad rips eliminated
     561 SID credit fixes
     258 SID model/clock infos
      28 tunes from /DEMOS/UNKNOWN/ identified
       6 tunes from /GAMES/ identified
      41 tunes moved out of /DEMOS/ to their composers' directories
       9 tunes moved out of /GAMES/ to their composers' directories
```

## SIDwinder 1.23 Enhanced!!

Released on 17 April 2011 and Enhanced version of SIDwinder 1.23 by PCH.

This version has an improved stay function and add many new functions .. as live piano and other next function in menu...




Get it from here:
http://noname.c64.org/csdb/release/?id=99574

## CGSC v1.19

The Compute's Sid Collection has just been updated

This update contains 94 MUS, 66 STR & 56 WDS new files and 17 MUS updated.

Download from www.c64music.co.uk

## HVSC #55

High Voltage SID Collection Update 55
Date: June 25, 2011

After this update, the collection should contain 40,400 SID files!

This update features (all approximates):

```
804 new SIDs (62 2SIDs)
113 fixed/better rips
  4 PlaySID/Sidplay1 specific SIDs eliminated
 30 repeats/bad rips eliminated
742 SID credit fixes
176 SID model/clock infos
 16 tunes from /DEMOS/UNKNOWN/ identified
  4 tunes from /GAMES/ identified
 36 tunes moved out of /DEMOS/ to their composers' directories
  2 tunes moved out of /GAMES/ to their composers' directories
```

Please note that from this update we've introduced PSID v3 format to allow SID files meant to be played in stereo, and the 31 character limit on the text fields AUTHOR, TITLE, RELEASED of SID files has been relaxed, allowing now all 32 chars to be used. Also filename length limit is completely gone, to allow filenames longer than 32 characters.

## CGSC v1.20

The Compute's Sid Collection has just been updated on 03 December 2011 and now contains 14% more files:

1363 MUS, 138 STR & 439 WDS new files and 23 files were updated.

The totals are now 10005 MUS files, 2346 STR files and 3218 WDS files.

Download from www.c64music.co.uk

## HVSC #56

High Voltage SID Collection: Update #56
Date: December 23, 2011

After this update, the collection should contain 41,250 SID files!

This update features (all approximates):

```
861 new SIDs
156 fixed/better rips
14  PlaySID/Sidplay1 specific SIDs eliminated
11  repeats/bad rips eliminated
786 SID credit fixes
88  SID model/clock infos
31  tunes from /DEMOS/UNKNOWN/ identified
12  tunes from /GAMES/ identified
51  tunes moved out of /DEMOS/ to their composers' directories
28  tunes moved out of /GAMES/ to their composers' directories
```

In this release we finally got rid of all PlaySID/Sidplay1 specific SIDs, by replacing all the remaining ones with proper RSID rips. Quite a milestone.

## SID Known v1.03

SID Known is a command line tool which you can use to identify SID tunes from SID and PRG files.

This tool can be used if e.g. you want to know which SID tune is used in a specific C64 demo or C64 game, or you have a SID tune found or ripped and you want to know if it is already in your SID collection.

Download: http://csdb.dk/release/?id=103744

## SIDplay/w 2.6

A new version of Sidplay/2 was released the same day of HVSC 56.

new features:

- PSIDv3/2SID support
- full 32 chars support for SID text fields
- 8580+digiboost selectable
- SidId support (latest sidid.cfg included, must be in the same dir of the exe), player routine detected is displayed in info page / Ctrl-P to show.

## CGSC v1.21

The Compute Sidplayer Collection has just been updated on 12 February 2012 and now contains an extra 592 MUS, 53 STR & 70 WDS files.

The totals are now 10593 MUS files, 2399 STR files and 3288 WDS files.

Download from www.c64music.co.uk

## CGSC v1.22

The Compute Sidplayer Collection has just been updated on 11 March 2012 and now contains 19% more, an extra 1635 MUS, 1999 STR & 1072 WDS files.

The totals are now 12228 MUS files, 3597 STR files and 4360 WDS files.

Download from www.c64music.co.uk

## CGSC v1.23

The Compute Sidplayer Collection has just been updated on November 2012.

The totals are now 12483 MUS files, 3705 STR files and 4461 WDS files.

Download from www.c64music.co.uk

## HVSC #57

High Voltage SID Collection: Update #57
Date: June 24, 2012

After this update, the collection should contain 42,212 SID files!

This update features (all approximates):

```
   975 new SIDs
   239 fixed/better rips
    13 repeats/bad rips eliminated
  3136 SID credit fixes
   396 SID model/clock infos
    38 tunes from /DEMOS/UNKNOWN/ identified
    18 tunes from /GAMES/ identified
    72 tunes moved out of /DEMOS/ to their composers' directories
    24 tunes moved out of /GAMES/ to their composers' directories
```

## HVSC #58

High Voltage SID Collection: Update #58
Date: December 21, 2012

After this update, the collection should contain 43,116 SID files!

This update features (all approximates):

```
   906 new SIDs
   185 fixed/better rips
     2 repeats/bad rips eliminated
   971 SID credit fixes
   111 SID model/clock infos
    13 tunes from /DEMOS/UNKNOWN/ identified
    14 tunes from /GAMES/ identified
    33 tunes moved out of /DEMOS/ to their composers' directories
    14 tunes moved out of /GAMES/ to their composers' directories
```

## CGSC v1.24

The Compute Sidplayer Collection has just been updated on June 2013.

The totals are now 12511 MUS files, 3707 STR files and 4480 WDS files.

Download from [www.c64music.co.uk](http://www.c64music.co.uk)

## HVSC #59

High Voltage SID Collection: Update #59
Date: June 28, 2013

After this update, the collection should contain 43,856 SID files!

This update features (all approximates):

```
   747 new SIDs
   315 fixed/better rips
     7 repeats/bad rips eliminated
   629 SID credit fixes
   750 SID model/clock infos
     9 tunes from /DEMOS/UNKNOWN/ identified
     5 tunes from /GAMES/ identified
    27 tunes moved out of /DEMOS/ to their composers' directories
    15 tunes moved out of /GAMES/ to their composers' directories
```

## HVSC #60

High Voltage SID Collection: Update #60
Date: December 22, 2013

After this update, the collection should contain 44,670 SID files!

This update features (all approximates):
825 new SIDs
233 fixed/better rips
11 repeats/bad rips eliminated
910 SID credit fixes
120 SID model/clock infos
24 tunes from /DEMOS/UNKNOWN/ identified
8 tunes from /GAMES/ identified
60 tunes moved out of /DEMOS/ to their composers' directories
7 tunes moved out of /GAMES/ to their composers' directories

## CGSC v1.25

The Compute Sidplayer Collection has just been updated on February 2014.

The totals are now 13481 MUS files, 4019 STR files and 4658 WDS files.

Download from www.c64music.co.uk

## CGSC v1.26

The Compute Sidplayer Collection has just been updated on May 2014.

The totals are now 13532 MUS, 4026 STR and 4689 WDS files

Download from www.c64music.co.uk

## CGSC v1.27

The Compute Sidplayer Collection has just been updated on May 2014.

The totals are now 12483 MUS files, 3705 STR files and 4461 WDS files.

Download from www.c64music.co.uk

## HVSC #61

High Voltage SID Collection: Update #61
Date: June 28, 2014

After this update, the collection should contain 45,418 SID files!

This update features (all approximates):

```
   749 new SIDs
  1479 fixed/better rips
     1 repeats/bad rips eliminated
   727 SID credit fixes
   323 SID model/clock infos
    19 tunes from /DEMOS/UNKNOWN/ identified
     4 tunes from /GAMES/ identified
    45 tunes moved out of /DEMOS/ to their composers' directories
     5 tunes moved out of /GAMES/ to their composers' directories
```

## CGSC v1.28

The Compute Sidplayer Collection has just been updated on June 2014.

The totals are now 13644 MUS files, 4043 STR files and 4737 WDS files.

Download from www.c64music.co.uk

## HVSC #62

High Voltage SID Collection: Update #62
Date: December 21, 2014

After this update, the collection should contain 46,056 SID files!

This update features (all approximates):

```
   639 new SIDs
   168 fixed/better rips
     1 repeats/bad rips eliminated
   181 SID credit fixes
     7 SID model/clock infos
     7 tunes from /DEMOS/UNKNOWN/ identified
     5 tunes from /GAMES/ identified
    20 tunes moved out of /DEMOS/ to their composers' directories
     6 tunes moved out of /GAMES/ to their composers' directories
```

## CGSC v1.29

The Compute Sidplayer Collection has just been updated on January 2015.

The totals are now 13896 MUS files, 4135 STR files and 4791 WDS files.

Download from www.c64music.co.uk

## Reformation

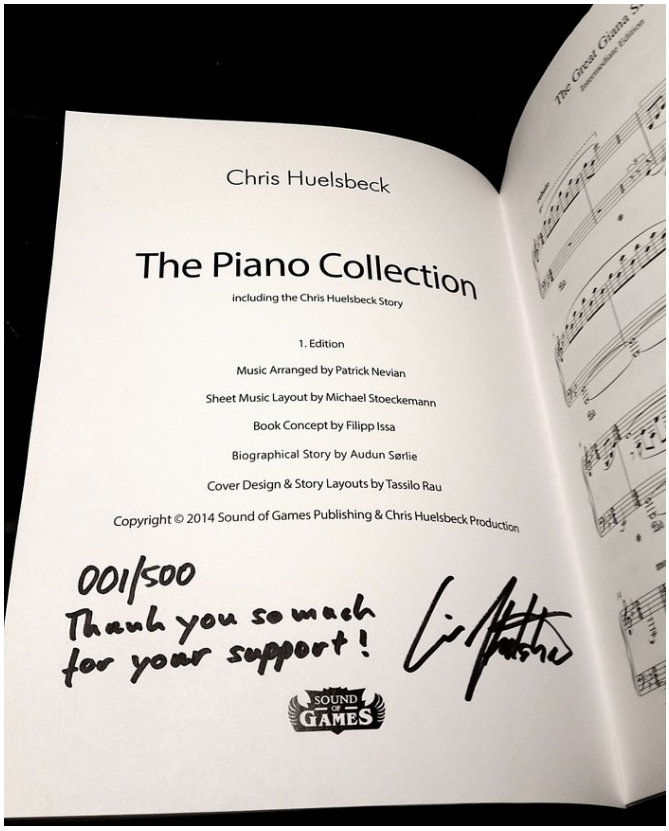
With a come back to the scene after more that 20 years, in end of 2014 Matt Gray realize a Kickstarter projects full of remixes and new sids:

http://www.kickstarter.com/projects/1289191009/reformation-c64-track-remakes-by-matt-gray-last-ni

# The Piano Collection

Chris Huelsbeck has launched another musical project onto Kickstarter at the end of 2014: The piano collection.

http://www.kickstarter.com/projects/chris-huelsbeck/chris-huelsbeck-the-piano-collection-and-limited-s

# Scarzix Interview!

by Stefano Tognon

This time I go to interview Scarzix, a very active musician that is helping into publicize the CheeseCutter editor and helping the musician that what to use it.

***Hello Carsten, could you introducing yourself and what you do in real life?***

Yes of cause, my name is Carsten Berggreen (Scarzix) - I'm from 1972, am a SID-composer and 6510 coder too, but in real-life I have been programming since I was 13-14 years old. I've been running my own web development company since 2002, but this year I have decided to put it on hold and apply for a "normal job" in a great company with great colleges as I have become father to a son who is now 4 years old and had some starting issues. Things has costed a lot of time and energy.. not to mention money, so the only logical thing was therefore to leave the "roller coaster of freedom" and apply for something steady. Priority has changed, you could say + I want to have spare time where I can focus on composing music again AND hopefully get some of my games ready for AppStore (I have some projects that has been moving forward too slow).

***Can you describe how you start your C64 activity in the early days and why you stop then it, before a came-back two decades after?***

I began touching BASIC on a friends VIC-20, then got my own C64 still coding BASIC and we were some boys in our school-class that all had these C64/C128 machines so we arranged to make competitions in who could come up with the most cool game-idea/effect for a meeting 2-3 weekends after. Then we gathered, showed our works and voted to see who made the best... so I guess I was "into demo competing" before entering the demoscene. I actually won a few times, even though my friends had a C128 with more powerful BASIC commands, but that just made me stubborn and work harder - and it paid out I guess.

At some point my games couldn't grow more, as I started to get out-of-memory errors or rather, my saved programs didn't look the same nor could they run anymore, and we all thought our programs was way too slow for those "really cool effects" we saw in crack intro's and games. So slowly we began to investigate what a monitor was and then trying to disassemble and write new code ourselfs. I can still recall doing an IRQ and making a raster-split for the first time.

I was very interested in coding and I had no skills nor any keyboard for playing music at that time, this came later when I began to pay visits to a local youth club where they had a music room and a synth. I was crazy already back then about Jarre music as I have an older brother (9 years older) who introduced me to his music and he was also the one buying a C64 before I got one myself. So I played a LOT of games on his computer.

At the youth club I began to practice playing simple things like Axel Foley, Depeche Mode bass'es etc. slowly I got the hang of it, and luckily my dad has always been playing harmon organ, but I hated that crappy sound and his cheeky tunes, so he never got a change to teach me. The only thing I ever learned from him was to play a "flute harmonica" - cant recall the instrument name, but one of those with a small keyboard and a mouth peace and he wrote down the letters for an "old tune" which he used to play for adults when he was out playing keyboard for parties (paid jobs).

So I knew where the key "C" was on a keyboard, but had no clue about chords or how to build a good lead or scales or how to read nodes.

***In early days you compose using JCH and Future Composer editors and now with Cheese-Cutter. What the motivation about those choices? Which is the best editor?***

I played with both BASIC and machine-code in order to do my own sound-player somehow. I was just aiming for sound-effects for my games and perhaps some simple bass kinda thing.. and maybe some drums as I also play a little drums (learned that at the club too).

Then Soundmonitor came... and suddenly I could open Chris' tunes and see how he managed to make them.

But Soundmonitor tunes was to heavy regarding memory as they weren't compressed and way too slow as I recall it for rastertiming and I still was only doing VERY simple "test tunes".

Then came FutureComposer with a whole new sound and much better drums than I had ever been able to make in Soundmonitor, so I tried that out. After some time, can't recall how long, I got contacted/called by a guy called Martin Lassen (handle: Stranger). He was coding on a new kinda Babarian/sideways scroller game and had already a tilemap and editor going, but he needed a musician and asked if I was up for that. I said, well - that would be fun to try, not really sure if I was able to do something that was good enough, but he had met his guy called JCH who had agreed to let him use his music-editor/player for the game, so I could have that as a tool. I had never heard about neither the editor nor JCH, but thought it sounded as a great idea as I had been designing and programming on my own player/editor which I had named "SOUNDIX" - oohh the hours I had spend on designing a UI for that.

Anyways; I got the "magic disk" delivered from JCH by mail-service and that was the initial start on a long and fun time with HOURS of discussions on phone about his new players and its features and feedback on how I found them etc. Great fun and quite expensive so my mum asked me to start paying half our phone-bill myself... bugger.. but hey, it was worth it. + I often posted my latest tunes to him and got new players and other worktunes back as feedback.

Unfortunately, we didn't manage to get much further with that first prototype of his game. So we met at Martin's place with a 3rd guy (who could do graphics) and decided that we wanted to make a game that we all three could play together. So the first goal would be to make a fun 3 player game as there was always one person observing the other two playing when you were three guys drinking before a party... we spotted a need for a new game.

First we tried to make "BOING!" which I have published a work-edition of my title music on my Soundcloud account: https://soundcloud.com/scarzix/boing-year-1989

But we ran into a, at that time, unsolvable problem regarding complex collisions between three players and the walls, so we decided to cancel the project and start all over with a less "fast paced" game and that was the tank game called BattleZone... or that's what we thought it would be called, because when we tried to sell it to a publisher, they told us that there was also a game called that, so we changed the title to CombatZone... which didn't exist... or so we thought. Because there is ALSO a previous CombatZone that has nothing to do with our game.

Anyways, this game was the first big assignment I have ever completed regarding composing. While practicing JCH editor and having a lot of fun, I was also able to compose some "out of context" tunes of my own. Including Galactix which I composed for the Triangle trackmo demo called "Road of Excess" http://csdb.dk/release/?id=2493

A lot of tunes came to reality in those few years around 1989-1990 and visiting demo-/copyparties was also a part of all this, as I really focused on the composing more than programming. I had found a new toy that was brilliant. I even bought a small keyboard so I could find my leads and tunes easier - and yes, I can play Galactix on a keyboard. :-)

I was very proud to be asked by Triangle at that time. This was my "big entry" in that precious demoscene which I really wanted to join, but didn't really know anyone inside the big groups. But getting one of my tunes into a great demo would be awesome - and it was a great feeling.

**What happen after completing the game music for Combat Zone?**
Our publisher screwed us to say the least, we made a bad deal and in the end we each got 1 Amiga 500 each as payment... and I had promised JCH a % of my final fee for the game, but suddenly I had nothing. I was SO ashamed for years, but JCH was a great friend and said that's okay - you can always pay me some other time. Which I never did, because I never got into any more projects as my second 1541 drive all of a sudden ALSO died. I was young, had no money, had to decide.. save up for Amiga monitor and drives or try to buy my 3rd 1541 drive? The choice was hard but had to be made. I left C64 and spend my savings on an extra floppy + some memory expansion for my A500 as I couldn't afford a harddrive. Bad bad bad...had a lot of fun, but using floppies was crap regarding composing as it took ages to go through ST disks for "a usable sample" + I could hold a tone for a lead nor make my own good sounds. I bought a sampler, and that didn't help me a lot as my keyboard was too cheap... dang... the trackers didn't work as JCH editor, I couldn't program sounds. I couldn't afford anything. I tried to manipulate sound in wave-editors, but it all sounded so noisy and crappy compared to the clean SID sound I was used to.

So I decided to focus on the assembler language of MC68000 and BOY was that a fun new tool compared to 6510... it was like coding BASIC again. 16 long registers... oooh yes baby! I began coding on a turnbased sci-fi game, but after some time I discovered that my game made errors when I pushed the mapplotter and blitter to the max for a smooth scrolling... I spend 1½ year on trying to figure out how or why it failed on my A500 and not on my friends machines.

After getting older and beginning my IT education as systemdeveloper, I learned about cold soldering and RAM hardware failures.

So that's how and why I left C64 and then Amiga... with bleeding hearts both times... on top of this I met this wonderful girl around 1993-94, but she turned out to have a deep depression so I focused on her and had little or no time for scene related stuff anymore. It was a crazy and chaotic youth of my twenties. I learned a lot about human brain and mind by observing what happen to her etc. A unbelievable rough time, but I tried the best I could to stand up to it - and to top it all, some years later, after more girlfriends, in 1999 I met this other girl and fell in love. Only to discover some months later, that she ALSO had a depression and was already on medicine... crap.. but this time I had the experience from the previous 3½ years, so I stood by her for 6½ and this was while I even tried to start up my company on my own. So it was really uphill for those first years of my company time.

Now forward till today, I slowly began to listen to more and more "old stuff" through SlayRadio, then began to download Remix.Kwed.Org tunes and I wondered how my old tunes might sound, like my Combatzone tunes (the loader "Waiting" and the title tune) would be fun to hear in a remake I thought, so I asked around, but nobody seemed to know those tunes nor being willing to remake them when I asked the composers.

Hmmmm.. maybe my tunes weren't that great? so I decided to take the "punch" directly and get the truth. Posted my Combatzone tunes in the Facebook group for sceners and asked on a scale from 0 to 10 .. where 10 is Galway, JT, Hubbard, Follin etc. where are my tunes? I can't recall the precise numbers, but I think they were like 8-9'ish from those responding. I was so happy. YES! people liked my tunes, but had never heard of me.

Then I posted a link to my Galactix and suddenly people "knew my tune" and said : "wooow? was that yours??? I have been listening to your tune for hours when I was 18 years old" etc. I was SOOOO happy to hear this. This was the lost energy that all of a sudden came back in my face. My tunes were loved, my love to the music was appreciated. The hours and weeks of hard work and months of practice had actually paid off back then. But I had no knowledge, because I left the

scene so suddenly. JCH had even contacted me back then and asked if I would join him on PC with Edlib, but I hated the AdLib PC sound even more than samples on Amiga, so I had rejected that cool offer. So stupid and stubborn I was back then.

Now, the fun started on Facebook, as a Norwegian guy I had never heard of called Pål said I should join his group, because they made demos and he liked my music and I should start making SIDs again. I said, I am not sure I will ever make SIDs again, but he still said that I should join his team of friends as they had a lot of fun with the old C64 and IF I ever decided to compose again, he would be proud to have me on his team.

Well, that was so sweet and kind, so why not? what's to loose? let's see how that goes, it thought without further research I accepted the friendship and joined OFFENCE... people were like: woaaaah cool! I was like, what? why? who? so I started to Google Offence and check CSDB.... *yikes!*

I had suddenly joined one of the biggest and most active groups on the C64 scene here in 2013 and I was like, okay if they have shown me this kinda "accept" I better try to deliver something in return and it's better be good, because looking and listening to their previous productions kinda set the bar rather high.

I knew I still had all my old equipment in boxes including old disks, but I couldn't recall my "ooh so secret encryption key" for the JCH editor + the drive was broken. So I searched for his editor on CSDB and tried it in VICE. Next I was trying to figure out if I could somehow access my old disks for worktunes to see if I could recall ANYTHING from how I did use that editor. Nothing. No options. No one in my near network with a way to load my disks. Then I searched around again and tried GOAT tracker and others but none of them had that stacked trackview and overview as I had in JCH editor. I asked here and there and someone told me that there was this CheeseCutter editor that was able to open old JCH editor work files and the player was also NewPlayer. I began to check it out and contacted the developer, Timo (Abaddon/Triad) who was the developer AND a musician himself.

We became friends rather quickly on Facebook and he was really great. He told me ALL the tricks and shortcuts I could imagine. He had thought this editor so much through, because he really missed having JCH editor on a modern PC platform, but with SID sound - and here it was.

I was thrilled - indeed here it was - JCH editor anno 2013.. Amazing!

He told me that the new 2.x generation broke with the old NewPlayer standards as he wanted to evolve the player further in his own directions, and I decided that the latest player would probably be beneficial to use as I would get updates when he made something new.

I started composing and my first "real tune" since 1990 was Back2Basic. I thought the name was rather saying for what I did. I was returning to the scene. Going back, to where it all started for me... BASIC in more than one terms.

One of the things that I didn't understand was why this editor was so "hidden" and no-one really knew about it, and his answer was so polite and sweet: "well, I am not much of a talker, I prefer to focus on my coding" - and since I came from 12 years of talking/communicating on web etc. I thought - okay, if he needs someone to advertise this awesome tool to keep him going, i will take that assignment myself.

So I created the CheeseCutter Community - thinking, since I got all that great help to learn it very fast from him, I should use my own skills of teaching others what I know. I looked at CSDB, but it seemed clunky and hard to put tutorials/images into without a lot of fuzz and weirdness, so I decided to go for Facebook until I have more time for a better community which I could program myself.

If we look at editors and players as two things, I agree that there might be more user-friendly designed editors than CheeseCutter. But CheeseCutter is opensourced and anyone willing can begin implementing a better UI if thats the 1st priority. For me it was great to have 800x600 instead of 320x200 and access to save on my harddrive instead of clunky D64 images in VICE.

Looking from a player perspective, I like to use CheeseCutter, because it works like I recall it to work with pulse/filter programs and with all the cool add-ons from Timo, its just a joyride. But the real killer feature that will make me pick CheeseCutter anytime, is that stacked track-sequencer that I believe JCH was the first to invent. Instead of a pattern/sequence flipping once you reach the end of it, it shows/scrolls the next pattern in below or above it - and you can always scroll both directions in your tune from top to bottom and back.

Being 100% self-taught musician and being very visual, this means that I can press "PLAY WITH FOLLOW" (SHIFT+F2) and then I can SEE where the tones OUGHT to be instead of having to know the theory of music. Secondly I often make a pattern and then scroll up and down while it loops so I can move notes/keys up and down until I am satisfied with the timing and sound/key. The more I do it, the faster I am at finding the right distance and tone. I usually can find a melody if you can humm it. Of-cause, having no "background" for this makes my journey a little longer, but I love this process and making covers is a true challenge that I think most good musicians will go through to learn how other compose their music.

Eg. when I did my INXS cover for Vandalism, I sampled the whole tune and then I started with the tempo of the rhythm, then I began breaking down the bass and chords etc. one step at a time.. and in the end, the result speaks for itself. I mean, when the audience at the presentation begins to sing the lyrics to my SID, I will call it a successful cover.

So why is CheeseCutter THE tool for me? Because I can scroll through my tune and don't loose contact with the flow. I have seen other players where transpose between sequences can break the visual flow of the sequence numbers. That's not good. On the other side, the instrument part of CheeseCutter is a really nasty killer if you don't know anything about SID programming. But coming from JCH's editor that's not a problem at all. That's why I wanna teach more new composers how to do it + I made the videotutorials on Youtube too. Because we never know when we have to leave again, so I don't want my knowledge of how to use this editor to get lost again. Aka, we don't know if I was suddenly in a plane-crash or car-accident. Not that I think much about that, but why not share and document this knowledge + a lot of people asked me many times if I could do some videos instead of pictures and text explanations.

Coming from a webcompany where I have touched e-learning etc. I know about multiple ways of learning, so this made absolute good sense to me - and I am not done doing them, things just takes their time.

***You are the fresh winner of the Ambient Sid Csdb music competition with a stereo SID tune. What was your filling into composing a tune that has to run for 12 minutes? Not an easy and common task?***

First of all, I had decided that I didn't want to participate in music compos' in general, because the vote system on CSDb is rather useless with its anonymous downvoters + everyone can see the current result before the voting has ended. That's not good. Too much strategy involved for a fair competition.
But - if we forget about the competition part itself and just see it as a fun assignment, this it becomes fun and interesting again.

I heard some of the entries and none of them sounded like I had expected. I thought when the rules stated we could be inspired by games like Delta, Lightforce etc. Parallax was the first that

came into my mind and NONE of these tunes had any of that Galway/Hubbard sound I adore so much. The original SID. There was a lot of experimental stuff, which is great, but I wouldn't want to have ANY of those on loop while I work. To much interference and noise for my mind. I know, its about taste - and I don't call it bad, it was just not what I had expected for this music compo.

Two days before I decided that I would go full-in, but I wanted to try multi SID as Timo had produced this 2SID beta edition of CheeseCutter (maybe because I have asked him so many times?) and I wanted to see how that would work since it was allowed in the rules.

Regarding the length, my longest tune until then was my title tune of Combat Zone, it's 6 minutes and I made it back then and it still fitted into the memory with the game. okay it could have been shorter, but I have always been a huge fan of Galway's Parallax + Hubbards Lightforce and Delta and they long tunes.

My Electrosphere Part 1 is also around 7 minutes, but my next part of Electrosphere isn't ready yet, so I couldn't push that through in two days.

Only choice was to go ambient chill down and then try to use the 6 channels in a way that would fill the soundscape much more than previous 6 channel stereo attempts - and I was rather surprised to see how much easier my ideas suddenly came to life, because I didn't have to sacrifice the "ground" or "pitch" of my tune to replace it with another part. Of-cause 6 channels isn't unlimited, but I could make a drum rythm with echo/delay effect, I could make chords that faded slowly, I could use both sides for the huge deep bass etc. All in all. FREEDOM beyond what I have tried before, and still it was the REAL SID sound not some noisy sample or semi synth on Amiga.

On top of this, I hadn't been able to compose 6 channeled until I got this new Logitech G35 headset with build-in surround where it can real-time mix left and right so you don't get pure raw 3+3 stereo, but mixed .. sort of reverb/hall effect, but not as much as I put on the version on Soundcloud.

The 12 minutes was a bit of a struggle in the end as it became very late the last night, but Cruzer was so close to get the new Zynaps inspired musicplayer ready for me as requested, so I really wanted to push forward and make that player go live too, so I didn't disappoint him for his work.

If I am ever going to open the workfiles again, there might be some parts I might wanna adjust slightly, won't tell where, but if I succeed with my plan for my first "SID album" in 2015/16 then Singularity in stereo would be a logic choice to put into it.

***What is the best feature of the SID chip and the missing one in your opinion?***

Best feature? I like it all, it's default, it's quirks. I love its pulse waveform and the things you can do with sync of a second channel. Used that a lot in Forgotten Times and Electrosphere Part 1.

Missing? well, that's obvious, more channels AND a separate filter pr. channel. In fact a filter pr. channel would have helped a LOT, but we must also remember that multispeed tunes or multichannel tunes are often using too much CPU cycles to play and if we hadn't had these limits, who knows how long we might have been waiting for a person like Rob Hubbard to invent a clever way of playing chords with arpeggios or drums like him and Jeroen Tel/Charles Deenen

So if we scaled the SID, we would also need to scale the CPU, memory and VIC. I love the fact that we can put in a SID-fixer in short time and plugin two physical SID chips, but it hurts my heart a little to know that ONE C64 has to be sacrificed to make another go dual SID.

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

Thinks should be able to run on real hardware, if you make demos. But floppies are out. Cartridges and network load is the logical replacement.
Composing on emulators, well - I have been doing this since I re-started my scene-entry in 2013 - I wish 6581 could be emulated correctly, and maybe some clever scener will be able to crack the problem for good. As I see it, all the old 6581 has so many revisions and quirks that's it really hard to tell which is the "right version" - everyone will say their own C64 had the right sound. My old breadbox had a very light filter, which I didn't know back then, so when I mailed my CombatZone tunes to Martin, he couldn't hear the leads - so we invented a "GUI setting" in our game for the FILTER of the music. How many games have you seen with that kinda option back in 1989/90?

I don't see a reason for why we as musicians shouldn't do what everyone else is doing already, aka use a modern platform for SID composing. Coders use cross-platform assemblers, testing in VICE. Graphic-artists, can use programs such as Photoshop, Pixcen, GIMP, Project One etc. loads of helpers there. So why shouldn't we as musicians not do the same? in the end, its about the end result being able to give entertainment to the audience.

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

As mentioned above, 6581 is hard to emulate + which version do we consider to be "the right" revision? 8580 is easier and also the platform that Offence has decided to stick with to make all tunes work on same chip.

***What is the worst and the better sid you composed?***

worst... damn, the worst tunes are worktunes, but in terms of released tunes. I had a period of "tigger tunes" where all the beats where "jumpy and sort of experimenting" - the leads sucked bigtime, but I learned a lot about instruments and effects doing them.

But take tunes like: Thief, Tecnochip, Noxius, Jingle, Impuls and Blue Sounds, they were all build up around the same "bass/drum" kinda method. I didn't notice back then, but all the leads are sort of "searching" for a direction, but they were made without plans. Just improvising what could be fun - the tunes also change style very often, because they were made over time or I got bored with the sound.

Even worse tunes are the first ones in Futurecomposer from Demo_of_the_Year_88 (part of Triangle's demo of the year) where I coded the part and made the tune. I invented these three volumeters for the triggers and that's the most interesting part about that. LOL, but I recently heard the tunes for a longer period and discovered that they had some hidden seqments/references I had forgotten all about oh yes and New Wave Part 1 and 2.

For my better tunes? well, back then I can only say Galactix and my CombatZone tunes. If we take 2013 and forward, hmmm.. those I am most proud of or listen to most frequently... hmmm... I was happy to dedicate my Back2Basic to PAL/OFFENCE who believed in me and invited me into the group, but my Connected, Electrosphere Part 1 (especially the last minute or two is where it gets creative), Starhiker - because that was so special to be able to have two giant tunes in SCROLLWARS - my first participation in a demo for 23 years, my cover of INXS, my "Greatest Pal" and ofcause Singularity which I have had running on loop for hours the past few days. Usually I cant do that with my own tunes, but this one I really like myself. So its purely made for my own self. I often compose with a goal/vision of what soundscape or visual effect/mood to support, like a storyboard for a demo or intro part.

But a common thing about my tunes: I don't release them if don't like listening to them. So usually I like my tunes. Ofcause deadlines and memorylimits can set a stopper to being creative, but I like that challenge a lot about SID composing.

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

Today or back then? My all-time favorites were of cause Galway, Hubbard and Tel. Of cause there were many others back then, but it's been 23 years and so many new composers have emerged. I still have a LOT of tunes to hear for the very first time in HVSC. I knew Drax back then, he even tried to play Galactix on MY keyboard at a demo-party once, where I corrected him on the tones, and we argued a bit about who was right, only to discover afterwards who we both were.. LOL JCH was laughing out loud at us. But JCH, Laxity, FurtureFreak, Bjerregaard, Drax and MANY others were also highly skilled. Hard to pick a single, they all had something special.

Talking about a guy who has been really helpful to me regarding getting up to speed with SID again since 2013, its Timo for his amazing help with CheeseCutter and speaking in terms of using effects such as vibrators and filters, it's Søren Lund (previously known as JEFF) - others has helped some, but most have sort of kept to themselves, like it's really dangerous to tell the secrets of the SID to others.

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

I don't have a single favorite. I hear a lot of highly skilled musicians doing some very technically advanced sounds in SID today, but to me, a great SID is not just about the instruments/sound, it's about providing a good melody/tune that you wanna hear again and again. Many of the highspec SIDs I hear in todays demo's are just beats and bass and effects. But little or no leads. That's sad in my opinion. In fact I promised PAL that I would make the SID scream again. I hope you all are satisfied with my attempts so far.

To pick a single all-time favorite, well I have already mentioned Lightforce, Delta and Parallax title, Wizball has a special place too. Most Galway tunes have for me. Then we have Thrust, Warhawk oh and JT... Cybernoid, Myth and SOO many others from his hands. Ghouls'n'Ghosts, Cauldron dang.. the list is endless. I cant really decide. it's like with Jarre, there is a tune for every mood.

***Finally, many thanks for the time you give for this interview, and now would you say something else to the our readers?***

I compose because I love the sound of SID, but I have had multiple requests for a place to buy/donate and download my tunes as MP3 etc.

So I'm planning my first ever album this year or minimum 2016. it will contain new SIDs and some of my existing SIDs, all arranged/mixed and upgraded to 6 channeled SIDs and released as MP3/FLAC downloads only. No SIDs, so buyers will have special versions of my tunes.

If curious, then feel free to support me at my www.soundcloud.com/scarix and www.facebook.com/scarzix - give me thumbs up and likes, if you like my tunes and tell me what you think. I compose because I can't stop it, but knowing that others like it too, makes me even more happy and sometimes it also gives me new ideas for other tunes.

If you wanna learn how to compose with CheeseCutter, I have my own Scarzix channel on Youtube + we have the community on FB, just request membership and wait for me to let you join us. Right now we are 180 people in there. Many are just backers who don't compose or old composers who enjoy the sound, but don't compose any longer.

COMPOSE A WHILE.... COMPOSE FOREVER...

Cheers!

If you want more information about Scarzix, then check those links:

http://csdb.dk/scener/?id=3304 (back to the scene)
http://www.berggreens.dk/ (living activity)
http://www.berggreens.dk/64/ (his story)

https://soundcloud.com/scarzix
https://www.facebook.com/scarzix
https://www.facebook.com/groups/529257460474557/ (CheeseCutter community)

And to end all in a good manner, the LAST YEAR he was finally able to pay JCH back so he has no more debt/shame towards his friendship. They were cool before, but now he doesn't have to feel guilt anymore.

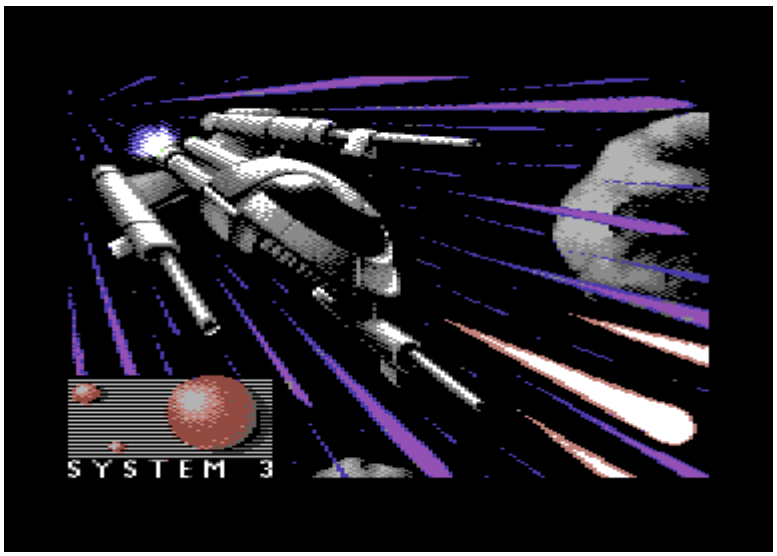
# Inside Matt Gray Dominator player

by Stefano Tognon <ice00@libero.it>

Matt Gray had released the source code of his player for a musical competition related to his Reformation project: http://www.remix64.com/articles/matt-gray-reformation-chiptune-driver-competition.html

A musician has to use his player to compose a tune and enter into the competition. So here his player will be analyzed to let musician with now programming skill to be able to create a tune with it. The source code is composed with 65tass syntax, so if you want that the tune will be accepted by Matt, it must stay into this format (even if you can modify his player for adding some little stuff).

If you remember in number #2 of SIDin there was my analysis of Matt Gray's Driller player by a reverse engineering work. The player used in Driller was a initial version, while the actual one being analyzed is the most advantage and was used into Dominator game.




As now we can look at commented labels it is more clear the purpose of each peace of code, so the previous work is substantially correct if compared with the actual knowledge we can get from this player. However the player is changed from the previous one in a non compatible way and it has lot of more features.

In the following part so it will be described the new player and it will be showed what is changed from the previous. I will try to describe the player not in the standard SIDin way (that means you are a coder and so had not difficult in use a source code), but I will go in trivial description of some tasks that for a musician with no programming knowledge will let him to use the source. Else I will based this works into the source code you will see at the end of this article: it is the same of Matt with only two modifications:

1. lot of comments at each line is inserted, so it is more simple to look at it
2. code is separated with some carriage return to better visually shows block of codes. It is so more easy to understand it.

Finally there is an addendum file to this number that contains the original source, the here commented source and another empty commented source that can be used as base for creating the tune (as we will see later in more details).

## Starting

The first step cover here is how to compile the actual source for having a PRG executable for listing to the music, and only after looks at the player code and how to modify it.

1. Download the Matt source code from here: http://www.remix64.com/services/files/matt-gray-dom6-public-source-1.zip
2. Extract all the contents of the archive in a given directory of your pc. Let suppose it is inside *C:\WORK* (on Windows) or *\opt\work* (on Linux)
3. For Linux get a copy of *c64tass assembler* (maybe just a *yum install 64tass* of *apt-get install 64tass* is enough in most Linux distributions), for windows it is inside the zip.
4. Open a Dos console on Windows (by executing CMD at program search file prompt) of a shell in Linux (just the one you want, like Bash)
5. Goes into the working directory by *cd c:\work* (on Windows) or *cd /opt/work* (on Linux)
6. Execute *64tass.exe dom6-public.asm* (on Windows) or *c64tass dom6-public.asm* (on Linux)
7. Get the *a.out* output and use in Vice or another C64 emulator to listen to the tune.
8. After the step 2 if you want to use the re-commented source code, just use the file from the addendum and replace the one inside *C:\work* or */opt/work*

Here there is the result of task 6 (on Linux):

```
[ice@localhost tmp]$ 64tass dom6-public.asm
64tass Turbo Assembler Macro V1.50.486?
64TASS comes with ABSOLUTELY NO WARRANTY; This is free software, and you
are welcome to redistribute it under certain conditions; See LICENSE!

Assembling file:   dom6-public.asm
Error messages:    None
Warning messages:  None
Passes:            3
Memory range:      $0801-$080d
Memory range:      $c000-$cee2
[ice@localhost tmp]$
```

## Background

Before being able to modify the source for creating your tune, you need to figure what there is into an assembly program. This section is perfectly trivial for a programmer and so he can goes to the next part, but may be is helpful for a musician. I also suggest to all to take a modern editor with syntax highlight for editing the source, as the different colors simplify the task (in Windows take *Notepad++* and in Linux *KWrite* for example and set the language to Assembly).

A source code of an assembly program can be essentially divided into some parts:

1. Machine code instructions for the processors
2. Data definitions (byte, word, string, ...)
3. Compiler directives
4. Constants declarations
5. Label declarations
6. Comments
7. Compiler facilities like macro

There are many different assemblers for C64, like *64tass*, *dasm*, *ca64* and even if all produces the same result, the syntax can vary from one to another, so here we will focused onto 64tass even if it will be gives some hints that works for all compilers.

The first assembler token we describe is a *comment*. A comment is all that follow a **;** mark.

Example:

```
; this is a comment
                PHA ; this is a comment after an instructions
```

An highlight syntax editor usually shows the comments in different color and maybe in different char attributes (like in Italic, as you can see in the above example).

The machine *code instructions* are formed by some token of reserved keywords (like LDA, STA, CMP, BNE, ...) and are inserted into the source at least after a blank space (for some compiler this is a rule) from the first column position. It is a best practice to align all those instructions at the same column (in some native C64 compilers they should be at a given numbered column position)

Example:

```
                LDA (BARS),Y
                CMP #$FF        ; now a comment on an instruction
                BNE FXSETUP
```

At compilation time, those instructions will be translated into their bytes representation and this will form the code of the program. For doing this the compiler need to know the memory locations where each instruction must run into the Commodore 64 (so the first step into an assembly is to declare the starting point as we see later).

*Directives of compiler* are given by special reserved keywords that usually begins with a **.** like .BYTE, .WORD, .ORG and each have a special meanings (there are even directive for creating macro code, or conditional if like .IFDEF, but that are used not by all compilers).

For this work, we need to know three compiler directive:

1. .BYTE is a directive that is used for inserting data bytes into the code, for example:

   ```
   .BYTE 10, 12, $22
   ```

   In this example the number 10, 12 and 34 are inserted into the code at the position where there is the directive. Notes that is more easy when working at machine level to use hexadecimal numbers that are declared by adding a $ onto it: $A is a 10. If you are not able to use hexadecimal numbers then use the decimals, but as you will see when we describes the player, all is more simple if you look at hexadecimal numbers (this is because a bytes is formed by 8 bits that becomes a number of two hexadecimal digits, and a word has 16 bits, so 4 hexadecimal digits).
2. .WORD is a directive that are used for inserting two (related) bytes into the codes as memory address locations, for example:

   ```
   .WORD $FFFF   ; insert the address $FFFF as two bytes
   ```

3. .TEXT is a directive for declaring a sequence of bytes from his (PET)ASCII representation. for example:

```
        .TEXT    '(C)1988 MG'
```

At this point we need to know just two other special features of an assembly source: *constant* and *label* declaration.

A label is a name given to a memory location inside the code or a constant if it is followed by a **=** mark and his value.

The fist (special) constant we look for is: *= $0801

This fix the starting point of the program at location $0801, so all the followed instructions are related to this position. Other compilers use the .ORG directive for this.
Labels are usually taken in first column and are limited into some compilers to 8 chars, while in other are freedom in length.

So look at this example:

```
BB = 55
*=$6677
CC   .byte BB
     LDA CC
```

In this code a constant BB is defined with the value of 55, then in memory at position $6677 it is putted the value of BB (so 55) and the label CC points to the location $6677 (CC should be seen as a variable). The next assembler instruction means to put into the accumulator of the processor the value that there is in location CC (so at location $6677) that we know to be 55.

OK, lets stop here otherwise we goes into too technical matters that are not relevant to know if you only need to use the player.

The last function to know is that there are two special operators that are to be used for getting the low or the high byte of an memory address. Look at this example:

```
*=$6677
CC    .BYTE 11
LO   .BYTE CC&255   ; get low byte of address = $77
HI   .BYTE CC/256   ; get high byte of address = $66
```

In the above example CC is a memory location at $6677 that points to a value of 11. In LO and HI locations we put the value of the CC memory address, so $77 for low and $66 for high. How this happen?

Simple: the CC memory location $6677 is bit-wise ANDed ($) with 255 that is $FF or 1111111 in binary, and so the operation gives $77 - the low address - (you can verify this with a scientific calculator). Instead for the high address, $6677 is divided (/) by 256 that is 2^8, so we get $66 (again use the scientific calculator).

Take present that other compiler uses < and > for getting low and high value of an memory location address.

## Player

Now it is time to look at the Matt player. We start in describe it from the beginning going step by step into more details.

The code is structured for starting a program from the standard location $0801 and to install a

IRQ (SETIRQ) that is synchronized with VIC raster (so 50HZ on PAL machine) and all tunes are played at speed 1X. If you need to have multi-speed tune or CIA based music, then you need to modify the source code (but in this case you already know how to do it) as the player is target for standard speed.

The player is however located to memory address $C000. You probably don't need to change this, but if you will make a 30 minutes long tune, maybe you could finish the free memories that follow that high address, so for changing the starting address, look at this constant definitions inside the code:

```
STARTADD        =$C000
```

All you need to do is changing this value to another location from the (safe) minimum of $1000.

A next optional step is to defined the frequency used by each notes. This is optional as those frequency is hard-coded into the player. You should know that in one octave we have 12 tones and that the frequency of one note an octave above has double frequency from the previous. So, if you fix the frequency that the note A4 should have (440HZ as standard), you should calculate all the frequency values of all notes and then translate in LOW and HIGH byte to put in SID registers for generating them.

Why changing those frequency? I just shows you two cases:

1. You want to use another musical system with different notes from the actual standard (like for play some ancient old music)
2. You just want to set a different A4 note starting Hz value

The NTL and NTH labels into the source reference to memory location containing the low and high value of each register to put in the SID, starting from note C0.

Example:

```
NTL             .BYTE   12,28,45,62,81,102,123,145,169,195
NTH             .BYTE   1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2
```

This means that notes C0 has 01*256+12=268 SID word value, or an effective frequency of 268*0,0587235=15,73HZ.
If you look at the A4 notes, it has a frequency of 424HZ (on PAL system. Interesting the frequency will be 441HZ on NTSC system).

In the source code there are the definition as constant of all the notes from C0 to B7 that are corresponding to frequency table:

```
C0              =1
CS0             =2
D0              =3
DS0             =4
...
AS7             =95
B7              =96
```

That values will be used later for inserting the music notes in the source code.

The next step is to defines the songs (tunes) that compose your music. Even if for the compo you only need to create a single tune, here we show how to insert some sub-tunes.

Inside the code you will see:

```
TN              .BYTE   3               ; song (track) number
```

This label refers to a variable in memory that contains the Track Number to play.

For example actually his value is 3, so the starting of program will make it to play the sub-tune 3: the player uses it the first time for initialize all and then make it as value $AB to remember he no needs to initialize again all the stuff. That values will goes to 0 when the tune is finish, indicating that no tunes are playing (or you can use it for stopping the tune playback at any time)

So your program has only to choose a value to put into TN to let the player to choose that tune to play.

If you make only a tune for the compo, that value should be 1.

The player is able to manage a speed for every songs and this is defined into the TDATA table:

```
;=========================
; Song speed
;=========================
TDATA           .BYTE   0,5,3,4
```

Here we see that there are 4 values of speed (0, 5, 3, 4), that corresponding to song 0, 1, 2 and 3. From the previous variable TN we already know that song 0 is for stopping sound, so the first speed value (0) is just to fill the table with 4 values.
Tune 3 as a speed of 4 and that means that the shorter note you can manage by the player will take 4 frames to be achieved.
The higher is that value and more slowly is the tune.

If you have only one tune with speed of 15, you need only to insert one value, like in this example:

```
TDATA           .BYTE   0,15
```

A song in the player is a sequences of 3 tracks, one for each voice, so it needs to define the memory locations of each tracks for being able to access to it. Those are achieved by low and high pointers tables:

```
;================================
; Songs (tunes) pointers
;================================
VOICE1L         .BYTE   0,TUNE1&255,OVER1&255,FIN1&255
VOICE1H         .BYTE   0,TUNE1/256,OVER1/256,FIN1/256

VOICE2L         .BYTE   0,TUNE2&255,OVER2&255,FIN2&255
VOICE2H         .BYTE   0,TUNE2/256,OVER2/256,FIN2/256

VOICE3L         .BYTE   0,TUNE3&255,OVER3&255,FIN3&255
VOICE3H         .BYTE   0,TUNE3/256,OVER3/256,FIN3/256
```

As you can see there is a table of values for voice 1 low pointer (VOICE1L) and for high pointer (VOICE1H) and then another 4 (two for voice 2 and two for voice 3).

Here there is the same convention of speed table: the first value in table is for tune 0 and so it is not defined (0); the second is for tune 1 address, the third for tune 2 address and the fourth for

tune 3 address.
So, the address of the track to execute for VOICE2 of second tune is OVER2 memory location label (remember from the introduction to assembly we give that OVER2&255 gives low byte of OVER2 and OVER2/256 gives high byte of OVER2).
OVER2 will contains the value (patterns) of the tracks relative to tune 2 and voice 2.

Now what should you do for adding a your song? Simple. You have to defines 3 memory location addresses (the contents of them will be added later) that are for each voices of your tune.
The name is totally arbitrary but you could have:

```
MYTUNE_VOICE1
MYTUNE_VOICE2
MYTUNE_VOICE3
```

Now fix the pointers in the source as expected:

```
;================================
; Songs (tunes) pointers
;================================
VOICE1L          .BYTE   0,MYTUNE_VOICE1&255
VOICE1H          .BYTE   0,MYTUNE_VOICE1/256

VOICE2L          .BYTE   0,MYTUNE_VOICE2&255
VOICE2H          .BYTE   0,MYTUNE_VOICE2/256

VOICE3L          .BYTE   0,MYTUNE_VOICE3&255
VOICE3H          .BYTE   0,MYTUNE_VOICE3/256
```

## Tracks

We actually have that a song is formed by tracks, but what is a track?

Simple answer: a track is a sequences of patterns and special flow commands over them.
More complete answer: a track is a sequences of values that can be as of those:

| | |
|---|---|
| XX | pattern XX to play (XX=$00..$FD) |
| $FF | repeat all the track (flow control) |
| $FE | end of music (flow control) |

In this version of the player (but was not present into Driller) there is an automatically fade out volume effects at the end of the tune (so when the $FE is reached).

The pattern (that is a sequence of actions to play) is an index to a table of pointers that point to pattern data definitions.

So, lets see an example:

```
MYTUNE_VOICE1   .BYTE   0, 1, 0, $FF
```

In this we set the player to plays pattern 0, then pattern 1, then pattern 0 and then to restart the sequence from the beginning ($FF) and so forever.

The BARLO and BARHI labels point to the tables of low pointers and high pointers of patterns definitions:

```
;=======================================
; pointer to bars (patterns) low address
;=======================================
BARLO           .BYTE   T0&255,T1&255
;=======================================
; pointer to bars (patterns) high address
;=======================================
BARHI           .BYTE   T0/256,T1/256
```

Here pattern index 0 refers to memory location T0 and index 1 to memory location T1; the labels name is so significant: 0 -> T0, 1 ->T1, but one can use the label he want like 0 -> PIPPO, 1 -> PLUTO. Now is however more difficult to remember that index 0 is for PIPPO and not for PLUTO, unless you define a constant even for index:

```
IND_PIPPO = 0
IND_PLUTO = 1

MYTUNE_VOICE1   .BYTE   IND_PIPPO, IND_PLUTO, IND_PIPPO, $FF

;=======================================
; pointer to bars (patterns) low address
;=======================================
BARLO           .BYTE   PIPPO&255,PLUTO&255
;=======================================
; pointer to bars (patterns) high address
;=======================================
BARHI           .BYTE   PIPPO/256,PLUTO/256

PIPPO   .BYTE $FE
PLUTO   .BYTE $FE
```

At this point in the source you have to fill the table of pointers using the naming you like for labels of patterns using the above instructions.

## Patterns

The contents of a pattern is a sequence of actions (like note to play, instruments to use) to performs, based onto the following table's rule:

| | |
|---|---|
| $00 | rest |
| $01..$6F | note xx |
| $70..$F9 | duration kk-70 |
| $FA NN | select instrument NN |
| $FB MM | slide down (negative portamento) (-MM) |
| $FC KK | slide up (positive portamento) (+KK) |
| $FD CI | plex (arpeggio) CI (C=counter, I=index in table) |
| $FF | end of pattern |

So, a value of 1 means to performs a C0 note. A value of $75 means that a note has a duration of 5 ($75-$70). The $FF command is for ending the pattern (so the player will take the next pattern into the track for executing).

In the version used in Driller, there were the $FD command for setting the note duration and now this has another meaning. For sure this new duration mechanism is more efficient as it uses less byte for a command that could be used lot in a tune.

Following the commands, the $FA is the one for setting an instrument that has the given index (NN). We will see later the instrument definition, so actually a 0 is for first instrument, 1 for second instrument and so on.

$FB and $FC commands are for creating slide down and up for the next note played. You can ever see this as a portamento. The MM value is the one to subtract or add to actual note pitch of the note at each cycle.

The $FD is a plex effect and it can be used for creating the typical SID arpeggio. In previous Driller player this were achieved at instrument definition, while actually it is ported to be at pattern command and removed from instrument definition. The value to insert after the $FD is a concatenation of two information (CI): the number of notes that form the arpeggio and the index in plex table. In hexadecimal this is easy: the most significant digit is the value C of counter, while the less significant digit is the index in table (I).

So, lets choose this command: $FD, $30.

This means, use the plex with table index 0 and 3 values from it.

Instead, $FD, $4A means use the plex at table index 10 (=$A) and 4 values from it.

At this point we need to know how the plex table is done.
Inside the source there is:

```
PLEXLH          .WORD   P0,P1,P2
```

and then the P0, P1 and P2 label table definitions:

```
P0              .BYTE   $07,$03,$00
P1              .BYTE   $09,$05,$00
P2              .BYTE   $08,$03,$00
```

Again here P0 is a name that remember it refers to table at index 0, but you can use MINNIE or whatever name you prefer, as in the above example about pattern pointers.

The values inside Px are the relative notes to play, so for P0, it means to play current note+7, then current note+3, then current note+0 and so repeat forever this sequence.

Lets gives a more exotic example:

pattern command: $FD, $30 and then $FD, $40.

The above command all refers to plex index 0:

```
PLEXLH          .WORD   P0

P0              .BYTE   $07,$03,$00,$09
```

Here the fist command play note+7, note+3 then note+0 while the second play note+7, note+3, note+0, then note+9

Finally there is only the command value 00 to analyze: it is a rest (no sound), but it is imple-

mented by set up the gate bit to off, so starting the release phase of ASDR and not simply by setting a SID frequency to 0.

Now we can sum the above information and look a one pattern definition from the source:

```
T56             .BYTE   $FA,$06,$7F,$FD,$36,G4,$FD,$31,F4
                .BYTE   $FD,$32,G4,$FD,$35,F4,$FF
```

In this pattern the instructions are:

1. takes instrument 6 (counting from 0)
2. takes a note duration of $F (15)
3. takes a plex of index 6 (counting from 0) that uses 3 values: $07, $04 and $00
4. plays the note G4
5. takes a plex of index 1 (counting from 0) that uses 3 values: $09, $05 and $00
6. plays the note F4
7. takes a plex of index 2 (counting from 0) that uses 3 values: $08, $03 and $00
8. plays the note G4
9. takes a plex of index 5 (counting from 0) that uses 3 values: $07, $04 and $00
10. plays the note F4
11. end of pattern

Lets see another example:

```
T6              .BYTE   $FA,$02,$EF,$FC,$0A,AS3,$FB,$0A,AS4,$FF
```

In this pattern the instructions are:

1. takes instrument 2 (counting from 0)
2. takes a note duration of 127 ($EF-$70=$7F)
3. starts a slide up of +10 ($0A)
4. plays note AS3
5. starts a slide down of -10 ($0A)
6. plays note AS4
7. end of pattern

At this point a question should pop up in your mind: can I combine the above commands together (like having slice and plex)? For how long it will continue to be executed a command?

| COMMAND | EXECUTION |
|---|---|
| $FA - New Instrument | It will be used for all next commands until the reach of another $FA command |
| $70..$F9 - Duration | It will be used as duration for all the next notes until a new duration is fount |
| $FB - Slide down | When activated, Plex and Vibrato (see Instrument) are stopped. It lies only for the next note to play |
| $FC - Slide down | When activated, Plex and Vibrato (see Instrument) are stopped. It lies only for the next note to play |
| $FD - Plex (arpeggio) | When activated, Slide and Vibrato (see Instrument) are stopped. Upon activate it stays up until a new instrument is activated (or there is a slide) |

# Instruments

The last part of the player is the definition of instruments. As we have just seen, the instrument is activate by selecting his index with pattern command $FA. So we should expect to have a table that translate that index to some memories locations that contains the instrument definition.

This is quit true, as the table is not make like patterns, but instead it points directly into memories of 8 bytes each and the index is just index*8 (so you can have a maximum of 256/8=32 instruments):

```
VDATA           .BYTE   $87,$11,$00,$E6,$00,$00,$10,$01
                .BYTE   $31,$41,$00,$ED,$15,$00,$40,$02
                .BYTE   $71,$41,$00,$8C,$30,$00,$40,$02

VDATA2          .BYTE   $00,$00,$81,$00,$00,$01,$8E,$00
                .BYTE   $00,$00,$81,$00,$00,$00,$8E,$00
                .BYTE   $00,$00,$81,$00,$00,$00,$8E,$00
```

Here we have three instruments defined and as one instruments needs 16 bytes, there are two memories area (VDATA and VDATA2) that contains those values. So the second instrument is defined by those bytes: $31,$41,$00,$ED,$15,$00,$40,$02,$00,$00,$81,$00,$00,$00,$8E,$00

For understanding the meaning of each field, follow this summary table:

| ***Index*** | ***Definition*** |
|---|---|
| 0 (0 table 1) | Wave form pulsation amplitude LO/HI -> 00HI/LO00 |
| 1 (1 table 1) | Control register |
| 2 (2 table 1) | A/D value |
| 3 (3 table 1) | S/R value |
| 4 (4 table 1) | Wave amplitude inc/dec value |
| 5 (5 table 1) | Not used (were used in old player for plex/arpeggio) |
| 6 (6 table 1) | Control register 2 (at new instrument and new note start) |
| 7 (7 table 1) | Instrument effect:<br>• 1: drum table effect<br>• 2: a pulse wave effect<br>• 4: implex (switch between waveform)<br>• 16: hat effect |
| 8 (0 table 2) | Oscillating frequency value (for vibrato) |
| 9 (1 table 2) | Length of vibrato intensity (for vibrato) |
| 10 (2 table 2) | Control register for effect implex (4) |
| 11 (3 table 2) | Slide value |
| 12 (4 table 2) | Slide flag (0= none, 1= down, 2= up, 3= down high, 4= up high) |
| 13 (5 table 2) | Drum table index |
| 14 (6 table 2) | Wave form pulsation amplitude LO/HI limit -> 00LO/xxxx .. 00HI/xxxx |
| 15 (7 table 2) | Not used |

So lets start to talk from some obvious fields:

Bytes at index 2 and 3 are the ADSR values to set into SID register for his working. The byte at index 1 is the Control register to use normally into the instrument. In the case the Control register will use a rectangular based waveform ($4x), there are other registers to take care of for this:

Byte at index 0 contains a codified low and high pulse value to use for the duty cycle of the rectangular waveform. Essentially the byte formed by the two nibble are swapped and used for the high an low value of pulse.

Lets take an example. The value $12 will be put in SID registers as $02 and $10 for high part and low part of duty cycle ($0210=528).

The byte at index 7 is a bitwise flag for activate some instrument effects. If the second bit is 1 ("a value of 2"), it is activate a pulse effect (variable pulse modulation).

This effect will use the:

- byte at index 4 as the amount of value to add/subtract from current duty cycle at each step
- byte at index 14 (index 6 of VDATA2) as the low and high limit of high value of pulse where to invert the pulse modulation direction. So, for example $12 will be $01 and $02 as low and high limit.

So, just give a simple example about how this works:

- $23 = Wave form pulsation amplitude LO/HI -> 00HI/LO00
- $A0 = Wave amplitude inc/dec value
- $18 = wave form pulsation amplitude LO/HI limit -> 00LO/xxxx .. 00HI/xxxx

The first value gives as $03 and $20 for high and low value of pulse ($0320=800): this is the starting value of pulse, then it starts to go up with a rate of $A0 (=160):

- $0320 =800
- $0320+$A0=$03C0 =960
- $03C0+$A0=$0460 =1120
- $0460+$A0=$0500 =1280
- $0500+$A0=$05A0 =1440
- $05A0+$A0=$0640 =1600
- $0640+$A0=$06E0 =1760
- $06E0+$A0=$0780 =1920
- $0780+$A0=$0820 =2080

When this progression will stop?

The $18 gives us the answer as it becomes: $01 as low limit and $08 as high limit. So the progression now is stopped as $08 is the high values of pulse ($0820->$08xx) and it starts to decrease with the same $A0 steps until the high part will reach $01 ($01xx).

If we compare this procedure with the one in Driller, the unique difference is that here the low and high limit is selectable by user, while in Driller it was fixed into the source.

The byte at index 8 (index 0 of VDATA2) is a frequency value to add/sub for a vibrato effect, while byte at index 9 (index 1 of VDATA2) is the length of the vibrato effects.

The vibrato uses this scheme, based onto a vibrato direction flag:

- 0 = down (first time)
- 1 = up
- 2 = up
- 3 = down
- 4 = down

It so goes with this sequences: 0, 1, 2, 3, 4, 1, 2, 3, 4, 1, 2...

We see previously that a slide can be achieved with pattern command, but inside instrument there is another slide that can be activated:

inside byte at index 11 (index 4 of VDATA2) there is the amount of frequency to add/sub and byte at index 12 (index 5 of VDATA2) activate the slide with according to those values of slide flag:

- 0= none
- 1= down
- 2= up
- 3= down high
- 4= up high

The difference between 1 and 3 or 2 and 4 is that the first will use low part of frequency, while the seconds use the high part of frequency (alternatively you can think at frequency and frequency*256 for the value to add/sub, so instrument slide is more capable in comparison with pattern slide).

The byte at index 6 of instrument data contains a control register that are used like a "basic" restart of note. In fact, it is used with this rules:

- When a new instrument is selected, it puts that value with gate bit forced to 0 as control register for the SID voice, then it will be used the control register that will follow the other rules.
- Instead when there is a new note, it puts this control register for the note, just before using the control register of index position 1.

At this point we need only to finish to understand the other effects that can be activated by making to on some bits in index 7 of data definitions.

By using the third bit ("a value of 4") it can be activated a implex effect (switch between waveform at start of note). The implex can be viewed as a more elaborated re-start of note. When playing a new note, it is used the control register value locate in byte at index 10 (byte 2 of VDATA2) for the first time, then the other normal control register.

By using the 5° bit ("a value of 16") it can be activate an hat effect. An hat effect consists into a brief noise waveform at frequency $50xx played after every note duration decrements. This effect was not present into Driller player.

Finally we are at the last and complex effect activated with bit 1 ("a value of 1") of byte at index 7 of instrument definition that was for bass/drum creation. This effect was primitively implemented into Driller, while actually it is a full functional one and it is based onto tables of values (you can have up to 256 tables, even if you effectively can use only one for instrument, so 32 maximum).

If you activate a drum at instrument, you need to set in the value at index 13 (index 5 in VDATA2) of instrument definitions with the index of a drum table.

In the source there are two pointers to tables:

```
DTL             .BYTE   DT&255,BT&255
DTH             .BYTE   DT/256,BT/256
```

In DTL there is the low address of memory location of table and into DTH the high memory location of table.

In the example there are two possible indexes for drum table:

- index 0 that point to label DT and index 1 that points to label BT (remember how &255 and /256 works with memory address).

Each table contains a sequences of two commands to be executed at each step and an end mark, so you can have a maximum of 127 entries:

| | |
|---|---|
| position 0+2*n | control register (CT) |
| position 1+2*n | high frequency (CT<0 - SET, CT>0 - SUB) |
| $FF | end of table |

As you can see the first command is the control register to use for the SID and the other is the high part of frequency to use if control register is a negative number and a value to subtract to current pattern/effect high frequency for positive number.

At this point you need a little of informatic number theory to understand what means that a byte is positive or negative.

A byte (8 bits) can store a number from 0 to 255, so for us it is always positive. But if you take the most significant bit (the last one) and make it 0 for positive and 1 for negative, then you have that the number can be seen from this interval: -128 to +127 (the number is in two's complement representation).

So, if the control is in interval 0..127 ($00..$7F) then the operation is to take actual high frequency (the one from note being played plus eventual slide/vibrato effects) and subtract to the value of second command. That new frequency is the one even stored for use in the next step.

Instead, if the value is in 128..255 ($80..$FF) then the operation is to put the value of second command as high frequency.

So a "negative value" of control register is when you choose a noise waveform ($8x) for creating a drum effect, while a "positive" value is for the other waveform (like $4x) that you can use for a bass.

At this point let analyze the two tables used into Dominator:

```
DT              .BYTE   $81,$30,$11,$02,$41,$04
                .BYTE   $80,$30,$80,$15,$80,$20,$80,$10
                .BYTE   $80,$20,$80,$20,$80,$10,$80,$20,$FF
BT              .BYTE   $81,$30,$41,$03,$40,$03,$80,$20
                .BYTE   $80,$10,$80,$20,$80,$10,$80,$20,$FF
```

So, starting from the first:

- $81,$30: noise waveform + gate bit on, frequency $30xx (>=721HZ AS4/B4 note)
- $11,$02: triangular waveform + gate bit on, frequency of note - $02xx (>11HZ)

- $41,$04: pulse waveform + gate bit on, frequency of note - $04xx (>22HZ)
- $80,$30: noise waveform + gate bit off, frequency $30xx (>=721HZ B4 note)
- $80,$15: noise waveform + gate bit off, frequency $15xx (>=360HZ B3 note)
- $80,$20: noise waveform + gate bit off, frequency $20xx (>=480HZ E4 note)
- $80,$10: noise waveform + gate bit off, frequency $10xx (>=240HZ F3 note)
- $80,$20: noise waveform + gate bit off, frequency $20xx (>=480HZ E4 note)
- $80,$20: noise waveform + gate bit off, frequency $20xx (>=480HZ E4 note)
- $80,$10: noise waveform + gate bit off, frequency $10xx (>=240HZ F3 note)
- $80,$20: noise waveform + gate bit off, frequency $20xx (>=480HZ E4 note)
- $FF: end of table

The second:

- $81,$30: noise waveform + gate bit on, frequency $30xx (>=721HZ AS4/B4 note)
- $41,$03: pulse waveform + gate bit on, frequency of note - $03xx (>15HZ)
- $40,$03: pulse waveform + gate bit off, frequency of note - $03xx (>15HZ)
- $80,$20: noise waveform + gate bit off, frequency $20xx (>=480HZ E4 note)
- $80,$10: noise waveform + gate bit off, frequency $10xx (>=240HZ F3 note)
- $80,$20: noise waveform + gate bit off, frequency $20xx (>=480HZ E4 note)
- $80,$10: noise waveform + gate bit off, frequency $10xx (>=240HZ F3 note)
- $80,$20: noise waveform + gate bit off, frequency $20xx (>=480HZ E4 note)
- $FF: end of table

For ending this analyze, lets see some instrument used into Dominator.

Instrument 0:

- $87: Wave form pulsation amplitude LO/HI -> 0780 - not used
- $11: Triangular waveform + gate bit on
- $00: A/D value
- $E6: S/R value
- $00: Wave amplitude inc/dec value - not used
- $00: Not used
- $10: Triangular waveform + gate bit off (at new instrument and new note start)
- $01: Instrument effect: 1 - drum table effect
- $00: Oscillating frequency value (for vibrato) - not used
- $00: Length of vibrato intensity (for vibrato) - not used
- $81: Noise waveform + gate bit on - not used
- $00: Slide value - not used
- $00: Slide flag (0= none, 1= down, 2= up, 3= down high, 4= up high) - not used
- $01: Drum table index - BT table
- $8E: Wave form pulsation amplitude LO/HI limit -> 80xx - E0xx - not used
- $00: not used

This instrument uses the drum table BT and after the notes are played by a simple triangular waveform

Instrument 1:

- $31: Wave form pulsation amplitude LO/HI -> $0130
- $41: Pulse waveform + gate bit on
- $00: A/D value
- $ED: S/R value
- $15: Wave amplitude inc/dec value +/- $15
- $00: Not used
- $40: Pulse waveform + gate bit off (at new instrument and new note start)

- $02: Instrument effect: pulse wave effect
- $00: Oscillating frequency value (for vibrato) - not used
- $00: Length of vibrato intensity (for vibrato) - not used
- $81: Noise waveform + gate bit on - not used
- $00: Slide value - not used
- $00: Slide flag (0= none, 1= down, 2= up, 3= down high, 4= up high) - not used
- $00: Drum table index - not used
- $8E: Wave form pulsation amplitude LO/HI limit -> $08xx - $0Exx
- $00: not used

This instrument use a pulse waveform with pulse modulation. The duty cycles start from $0130, then reach $0E00 at $15 steps, then go down to $08xx, and repeat up/down forever.

Instrument 4:

- $F1: Wave form pulsation amplitude LO/HI -> $01F0
- $41: Pulse waveform + gate bit on
- $0F: A/D value
- $00: S/R value
- $20: Wave amplitude inc/dec value +/- $20
- $00: Not used
- $40: Pulse waveform + gate bit off (at new instrument and new note start)
- $12: Instrument effect: pulse wave effect + hat
- $00: Oscillating frequency value (for vibrato) - not used
- $00: Length of vibrato intensity (for vibrato) - not used
- $81: Noise waveform + gate bit on - not used
- $00: Slide value - not used
- $00: Slide flag (0= none, 1= down, 2= up, 3= down high, 4= up high) - not used
- $00: Drum table index - not used
- $33: Wave form pulsation amplitude LO/HI limit -> $03xx - $03xx
- $00: not used

This instrument uses pulse modulation that start from $01F0 with a $20 steps and goes until $03xx where it stay around. There is even the hat effect applied.

Instrument 9:

- $44: Wave form pulsation amplitude LO/HI -> $0440
- $41: Pulse waveform + gate bit on
- $00: A/D value
- $7C: S/R value
- $C0: Wave amplitude inc/dec value +/- $C0
- $00: Not used
- $40: Pulse waveform + gate bit off (at new instrument and new note start)
- $02: Instrument effect: pulse wave effect
- $90: Oscillating frequency value (for vibrato) +/- $90
- $02: Length of vibrato intensity (for vibrato) $02
- $81: Noise waveform + gate bit on - not used
- $00: Slide value - not used
- $00: Slide flag (0= none, 1= down, 2= up, 3= down high, 4= up high) - not used
- $00: Drum table index - not used
- $35: Wave form pulsation amplitude LO/HI limit -> $03xx - $05xx
- $00: not used

This instrument is a more complex: it has vibrato and pulse modulation too. The pulse start from $0440 and goes up to $0500, then goes down to $0300 and up/down forever, while the vibrato

has a frequency step of $C0 and a shot length of $02.

Instrument 13:

- $00: Wave form pulsation amplitude LO/HI - not used
- $11: Triangular waveform + gate bit on
- $0F: A/D value
- $00: S/R value
- $00: Wave amplitude inc/dec value - not used
- $00: Not used
- $10: Triangular waveform + gate bit off
- $00: Instrument effect: none
- $00: Oscillating frequency value (for vibrato) - not used
- $00: Length of vibrato intensity (for vibrato) - not used
- $81: Noise waveform + gate bit on - not used
- $B3: Slide value: -$B300
- $03: Slide flag (0= none, 1= down, 2= up, 3= down high, 4= up high) - 3 down hight
- $00: Drum table index - not used
- $8E: Wave form pulsation amplitude LO/HI limit -> $08xx - $0exx - not used
- $00: not used

This triangular waveform has a "big" negative slide effect

## Source

At this point I present the re-commented source code of Dominator player. The license stay as at the original has you can find inside the Matt's source code package.

```
;------------------------------------------------------------------------------
; BASIC HEADER (WILL AUTOSTART FILE WHEN DROPPED INTO VICE)
; THE "SETIRQ" LINE REFERS TO A LABEL FURTHER DOWN THE CODE

        *= $0801
        .word (+), 2005
        .null $9e, ^SETIRQ
+       .word 0

;------------------------------------------------------------------------------

                ;PLAYER V4.2
                ;(C)1987
                ;MATT GRAY
                ;This work is licensed
                ;under a Creative Commons
                ;Attribution-NonCommercial 4.0
                ;International License
STARTADD        =$C000          ; starting address of player
C0              =1
CS0             =2
D0              =3
DS0             =4
E0              =5
F0              =6
FS0             =7
G0              =8
GS0             =9
A0              =10
AS0             =11
B0              =12
C1              =13
CS1             =14
D1              =15
DS1             =16
E1              =17
F1              =18
FS1             =19
G1              =20
GS1             =21
A1              =22
AS1             =23
```

```
B1              =24
C2              =25
CS2             =26
D2              =27
DS2             =28
E2              =29
F2              =30
FS2             =31
G2              =32
GS2             =33
A2              =34
AS2             =35
B2              =36
C3              =37
CS3             =38
D3              =39
DS3             =40
E3              =41
F3              =42
FS3             =43
G3              =44
GS3             =45
A3              =46
AS3             =47
B3              =48
C4              =49
CS4             =50
D4              =51
DS4             =52
E4              =53
F4              =54
FS4             =55
G4              =56
GS4             =57
A4              =58
AS4             =59
B4              =60
C5              =61
CS5             =62
D5              =63
DS5             =64
E5              =65
F5              =66
FS5             =67
G5              =68
GS5             =69
A5              =70
AS5             =71
B5              =72
C6              =73
CS6             =74
D6              =75
DS6             =76
E6              =77
F6              =78
FS6             =79
G6              =80
GS6             =81
A6              =82
AS6             =83
B6              =84
C7              =85
CS7             =86
D7              =87
DS7             =88
E7              =89
F7              =90
FS7             =91
G7              =92
GS7             =93
A7              =94
AS7             =95
B7              =96
POINTS          =$FC                    ; track pattern pointer
BARS            =$FE                    ; pattern pointer
V2LO            =V1LO+7
V2HI            =V1HI+7
V3LO            =V1LO+14
V3HI            =V1HI+14
                *=STARTADD
TN              .BYTE   3               ; song (track) number
FADE            .BYTE   0               ; fade value

DRIVER
                LDX #$00                ; Voice 1
                JSR MAIN
                LDX #$07                ; Voice 2
                JSR MAIN
```

```
                LDX #$0E                ; Voice 3
                JSR MAIN
                RTS

MAIN            LDA TN                  ; song (track) number
                BNE PLAYMUSIC2
                STA $D418               ; Select volume and filter mode (to 0)
                RTS
PLAYMUSIC2
                CMP #$AB                ; current songs ?
                BEQ MUSIC
                JMP SETPOINTS           ; init the tracks

SETCONT         LDA #0                  ; clear all sid registers
                LDY #23         ;;;
SIDLOOP         STA $D400,Y
                DEY
                BPL SIDLOOP

                LDA #$0F                ; full volume
                STA $D418               ; Select volume and filter mode
                STA VOLUME              ; remember actual volume

                LDY #0
                STY BARCOUNT            ; set track 1 position to the beginning
                STY BARCOUNT+7          ; set track 2 position to the beginning
                STY BARCOUNT+14         ; set track 3 position to the beginning
                STY V1DUR               ; actual note length duration voice 1
                STY V1DUR+7             ; actual note length duration voice 2
                STY V1DUR+14            ; actual note length duration voice 3
                STY BEATCOUNT           ; set pattern index to the beginning
                STY BEATCOUNT+7
                STY BEATCOUNT+14
                STY FADE                ; clean fade
                INY
                STY SPEED               ; actual cycle timer (speed of song)
                STY SPEED2              ; not used
                JMP QUIT                ; exit with dec cycle timer
MUSIC
                LDA FADE                ; is there a fade to apply?
                BEQ OKMUSIC

                DEC VOLTIME             ; decrement volume delay
                BPL OKMUSIC

                LDA FADE                ; fade value
                STA VOLTIME             ; set volume delay
                DEC VOLUME              ; decrement the actual volume
                BPL OKFADE

                LDA #0                  ; stop sound
                STA TN                  ; song (track) number
                RTS

OKFADE          LDA VOLUME              ; actual volume
                STA $D418               ; Select volume and filter mode

OKMUSIC         LDY SOUND,X             ; index of instrument data
                LDA VDATA+7,Y           ; load instrument effect
                AND #4                  ; is effect implex (4)?
                BEQ NOIMPLEX

                LDA IMPLEX,X            ; read implex flag
                BEQ NORMAL

                DEC IMPLEX,X            ; dec implex (reset)
                LDA VDATA2+2,Y          ; control register
                STA $D404,X             ; Voice 1: Control registers
                BNE NOIMPLEX

NORMAL          LDA VDATA+1,Y           ; control register
                STA $D404,X             ; Voice 1: Control registers
NOIMPLEX
                LDA VDATA+7,Y           ; load instrument effect
                AND #$10                ; hat effect?
                BEQ NOHAT

                LDA HAT,X               ; read hat indicator
                BEQ CANCELHAT

                DEC HAT,X               ; dec hat indicator

VAL             LDA #$50
                STA $D401,X             ; Voice 1: Frequency control (hi byte)

                LDA #$81                ; noise + ADS
                STA $D404,X             ; Voice 1: Control registers
                BNE NOHAT
```

```
                                ; cancel hat by set original value
CANCELHAT       LDA C1NHIGH,X   ; high frequency for vibrato/slide/drum
                STA $D401,X     ; Voice 1: Frequency control (hi byte)
                LDA VDATA+1,Y   ; control register
                STA $D404,X     ; Voice 1: Control registers

NOHAT           LDA SPEED       ; actual cycle timer (speed of song - tempo)
                BNE GOFX

                LDA #1
                STA HAT,X       ; hat indicator

DELAYS2         DEC V1DUR,X     ; actual note length duration voice 1
                BMI MAINLOOP    ; jump if note is finished
GOFX            JMP CHECKFX
                                ; init tracks with selected songs
SETPOINTS       LDY TN          ; load song number
                LDA VOICE1L,Y   ; get track 1 from song (low)
                STA V1LO        ; set current track 1 position (base low)
                LDA VOICE1H,Y   ; get track 1 from song (high)
                STA V1HI        ; set current track 1 position (base high)
                LDA VOICE2L,Y   ; get track 2 from song (low)
                STA V2LO        ; set current track 2 position (base low)
                LDA VOICE2H,Y   ; get track 2 from song (high)
                STA V2HI        ; set current track 2 position (base high)
                LDA VOICE3L,Y   ; get track 3 from song (low)
                STA V3LO        ; set current track 2 position (base low)
                LDA VOICE3H,Y   ; get track 3 from song (high)
                STA V3HI        ; set current track 3 position (base high)

                LDA TDATA,Y     ; read the song speed
                STA TEMPOBYTE   ; set cycle timer (speed of song - tempo)
                JMP SETCONT     ; init music
                                ; decrement the cycle timer (speed of song - tempo)
QUIT            CPX #$0E        ; is voice 3 ?
                BNE QUIT2       ; no, exit
                DEC SPEED       ; dec actual cycle timer (speed of song - tempo)
                BPL QUIT2       ; jump on positive value
                LDA TEMPOBYTE   ; get cycle timer (speed of song - tempo)
                STA SPEED       ; set actual cycle timer (speed of song - tempo)
QUIT2
                LDA #$AB        ; current song
                STA TN          ; set song (track) number
QUIT3           RTS

MAINLOOP        LDA V1LO,X      ; current track 1 position (base low)
                STA POINTS      ; track pattern pointer (low)
                LDA V1HI,X      ; current track 1 position (base high)
                STA POINTS+1    ; track pattern pointer (high)

AGAIN4          LDY BARCOUNT,X  ; read actual track position (offset - bar counter)
                LDA (POINTS),Y  ; read actual track pattern pointer value

NOTEND2         TAY
                LDA BARLO,Y     ; read pattern pointer low
                STA BARS        ; pattern pointer low
                LDA BARHI,Y     ; read pattern pointer high
                STA BARS+1      ; pattern pointer high

                LDA #$FF
                STA GATEBYTE    ; mask gate byte to let all as is

                LDA #0          ; no slide
                STA V1SLIDE,X   ; slide flag
                                ; read the pattern value of note to play
AGAIN           LDY BEATCOUNT,X ; load pattern index of this voice
                LDA (BARS),Y    ; read a pattern value
                BNE AGAIN3
                JMP PLAYNOTE

AGAIN3          CMP #$FD        ; plex?
                BCC SLIDE       ; jump if <$FD

                INY             ; next pattern value index
                INC BEATCOUNT,X ; next (saved) pattern value index
                LDA (BARS),Y    ; read a pattern value (plex)
                JMP PLEXSETUP

REGET           INC BEATCOUNT,X ; next (saved) patter value index
                BNE AGAIN

SLIDE           CMP #$FB        ; slide down ?
                BCC NEWVOICE    ; jump if <$FB

                CMP #$FB        ; slide down ?
                BNE SLIDEUP     ; jump if <>$FB

                LDA #1          ; negative slide (down - portamento)
```

```
SLIDECONT       STA V1SLIDE,X          ; store in slide flag
                INY                    ; next pattern value index
                INC BEATCOUNT,X        ; next (saved) patter value index
                LDA (BARS),Y           ; read a pattern value
                STA SLIDELO,X          ; store slide value

                LDA #0
                STA V1PLEX,X           ; no plex (arpeggio)
                STA V1VIB,X            ; no vibrato
                BEQ REGET

SLIDEUP         LDA #$02               ; positive slide (up - portamento)
                BNE SLIDECONT          ; store portamento value

NEWVOICE        CMP #$FA               ; is new instrument?
                BCC VIBDELAY           ; jump if <$FA
;=======================
; New Instrument
;=======================
                                       ; select new instruments (voice)
                INY                    ; next pattern value index
                INC BEATCOUNT,X        ; next (saved) pattern value index
                LDA (BARS),Y           ; read a pattern value (instrument index)

                ASL A
                ASL A
                ASL A                  ; index * 8
                STA SOUND,X            ; index of instruments data
                TAY

                LDA VDATA+6,Y          ; control register 2
                AND #$FE               ; gate off (release phase on)
                STA $D404,X            ; Voice 1: Control registers

                LDA VDATA,Y            ; read Hi/Lo of pulsation amplitude
                PHA
                AND #$0F
                STA V1PULSEHI,X        ; actual wave form pulsation amplitude (hi byte)
                STA PWH,X              ; Wave form pulsation amplitude (hi byte)
                PLA

                AND #$F0
                STA V1PULSELO,X        ; actual wave form pulsation amplitude (lo byte)
                STA PWL,X              ; Wave form pulsation amplitude (lo byte)

                LDA VDATA2+6,Y         ; pulsation amplitude low/high value
                PHA
                AND #$0F
                STA PUCH,X             ; pulsation amplitude (00xx/0000 high limit)
                PLA
                AND #$F0
                ROL A
                ROL A
                ROL A
                ROL A
                STA PUCL,X             ; pulsation amplitude (00xx/0000 low limit)

NOPR            LDA #0
                STA VDELAY,X           ; no vibrato delay
                STA V1VIB,X            ; no vibrato
                STA V1PLEX,X           ; no plex
                BEQ REGET

VIBDELAY        CMP #$F9               ; vibrato delay?
                BCC NOTEDUR
                INY
                INC BEATCOUNT,X        ; next pattern index
                LDA (BARS),Y
                STA VDELAY,X           ; store vibrato delay
                JMP REGET

NOTEDUR         CMP #$70               ; note duration?
                BCC PLAYNOTE
                SBC #$70               ; extract note duration value
                STA NEWDUR,X           ; new note duration
                JMP REGET

;=======================
; Play a new note
;=======================
PLAYNOTE        BEQ NOBV               ; no note (rest)?
                CLC
                ADC TP,X               ; transpose (seems to not be used)

NOBV            STA BARVALUE,X         ; value of bar (pattern) - note to play
                LDA NEWDUR,X           ; new note duration
                STA V1DUR,X            ; actual note length duration voice 1
```

```
                LDA #0
                STA DRUM2,X             ; reset drum table index to beginning

                LDA #1
                STA IMPLEX,X            ; reset implex effect

                LDA BARVALUE,X          ; value of bar (pattern) - note to play
                BEQ PLAYCONT2

                LDY SOUND,X             ; index of instrument data
                LDA VDATA+7,Y           ; instrument effect
                AND #$02                ; is effect wave (2)?
                BEQ PLAYCONT
                                        ; reload wave modulation for the new note
                LDA PWL,X               ; Wave form pulsation amplitude (lo byte)
                STA V1PULSELO,X         ; actual wave form pulsation amplitude (lo byte)

                LDA PWH,X               ; Wave form pulsation amplitude (hi byte)
                STA V1PULSEHI,X         ; actual wave form pulsation amplitude (hi byte)

PLAYCONT        LDA BARVALUE,X          ; value of bar (pattern) - read note to play
                BNE NOREST

;===============
; A rest (no note)
;===============
PLAYCONT2       LDA TEMP3,X             ; note to play
                STA BARVALUE,X          ; value of bar (pattern) - read note to play
                LDA #$00
                STA TEMP3,X             ; note to play
                LDY SOUND,X             ; index of instrument data
                DEC GATEBYTE            ; release gate
                BNE NOPITCH

;===============
; Out note
;===============
NOREST          STA TEMP3,X             ; note to play
                TAY
                LDA NTH,Y               ; get high frequency from table
                STA $D401,X             ; Voice 1: Frequency control (hi byte)
                STA V1HIFREQ,X          ; Voice 1: Frequency control (hi byte) for effect 1
                STA C1NHIGH,X           ; high frequency for vibrato/slide/drum

                LDA NTL,Y               ; get low frequency from table
                STA $D400,X             ; Voice 1: Frequency control (lo byte)
                STA V1LOFREQ,X          ; Voice 1: Frequency control (lo byte)
                STA C1NLOW,X            ; low frequency for vibrato/slide/drum

                LDY SOUND,X             ; index of instrument data
                LDA VDATA+6,Y           ; control register 2
                STA $D404,X             ; Voice 1: Control registers

                LDA VDATA+2,Y           ; read A/D value
                STA $D405,X             ; Generator 1: Attack/Decay

                LDA VDATA+3,Y           ; read S/R value
                STA $D406,X             ; Generator 1: Sustain/Release

                LDA V1PULSELO,X         ; actual wave form pulsation amplitude (lo byte)
                STA $D402,X             ; Voice 1: Wave form pulsation amplitude (lo byte)

                LDA V1PULSEHI,X         ; actual wave form pulsation amplitude (hi byte)
                STA $D403,X             ; Voice 1: Wave form pulsation amplitude (hi byte)

                LDA VDELAY,X            ; vibrato delay
                STA VIBD,X              ; actual vibrato delay
NOPITCH
                LDA VDATA+1,Y           ; control register
                AND GATEBYTE            ; manipulate with gate byte
                STA $D404,X             ; Voice 1: Control registers

                INC BEATCOUNT,X         ; next pattern index
                LDY BEATCOUNT,X         ; read pattern index
                LDA (BARS),Y
                CMP #$FF                ; end of pattern?
                BNE FXSETUP

                LDA #$00
                STA BEATCOUNT,X         ; reset pattern index
                INC BARCOUNT,X          ; inc actual track position (offset - bar counter)
                LDY BARCOUNT,X          ; read actual track position (offset - bar counter)
                LDA (POINTS),Y
                CMP #$FF                ; repeat the song (track)?
                BNE NOTEND

                LDA #$00                ; start at beginning
                STA BARCOUNT,X          ; actual track position (offset - bar counter)
```

```
                BEQ FXSETUP

NOTEND          CMP #$FE                ; end of song (track)?
                BNE FXSETUP

                LDA #$5F
                STA FADE                ; fade value
                INC BARCOUNT,X          ; inc actual track position (offset - bar counter)
FXSETUP
                LDA TEMP3,X             ; temp pattern value (note to play)
                BEQ CHECKFX

                LDY SOUND,X             ; index of instrument data
                LDA V1SLIDE,X           ; slide flag
                BNE ALREADY

                LDA VDATA2+4,Y          ; instrument slide flag
                BEQ NOBEND

                STA V1SLIDE,X           ; slide flag
                LDA VDATA2+3,Y          ; instrument slide value
                STA SLIDELO,X           ; slide value
ALREADY         JMP SLIDECHECK
NOBEND

VIBCHECK
                LDA VDATA2,Y            ; vibrato step
                BEQ NOVIB
                JMP VIBSETUP

NOVIB           STA V1VIB,X             ; vibrato flag
                JMP QUIT
;=================================
; pulse modulation timbre routine
;=================================
CHECKFX
                LDA VDATA+4,Y           ; Wave amplitude inc/dec value
                STA PTEMP               ; store for late use
                BEQ PLEXCHECK

                LDA PMODDIR,X           ; direction of pulse modulation
                BNE PDOWN

                CLC
                LDA V1PULSELO,X         ; actual wave form pulsation amplitude (lo byte)
                ADC PTEMP               ; add incremental value
                STA V1PULSELO,X         ; actual wave form pulsation amplitude (lo byte)
                STA $D402,X             ; Voice 1: Wave form pulsation amplitude (lo byte)

                LDA V1PULSEHI,X         ; actual wave form pulsation amplitude (hi byte)
                ADC #$00
                STA V1PULSEHI,X         ; actual wave form pulsation amplitude (hi byte)
                STA $D403,X             ; Wave form pulsation amplitude (hi byte)

                CLC
                CMP PUCH,X              ; pulsation amplitude (00xx/0000 high limit)
                BCC PLEXCHECK

                INC PMODDIR,X           ; change direction of pulse modulation
                BNE PLEXCHECK

PDOWN           LDA V1PULSELO,X         ; actual wave form pulsation amplitude (lo byte)
                SEC
                SBC PTEMP
                STA V1PULSELO,X         ; actual wave form pulsation amplitude (lo byte)
                STA $D402,X             ; Voice 1: Wave form pulsation amplitude (lo byte)
                LDA V1PULSEHI,X         ; actual wave form pulsation amplitude (hi byte)
                SBC #$00
                STA V1PULSEHI,X         ; actual wave form pulsation amplitude (hi byte)
                STA $D403,X             ; Wave form pulsation amplitude (hi byte)
                CLC
                CMP PUCL,X              ; pulsation amplitude (00xx/0000 low limit)
                BCS PLEXCHECK
                DEC PMODDIR,X           ; change direction of pulse modulation

PLEXCHECK
                LDA V1PLEX,X            ; plex flag
                BEQ VIBUPDATE

;===============
; plex timbre routine
;===============
                LDA PLEXTEMP,X          ; plex index in table
                ASL A
                TAY                     ; index * 2
                LDA PLEXLH,Y            ; plex table index low
                STA PLEXADD+1
                LDA PLEXLH+1,Y          ; plex table index high
```

```
                STA PLEXADD+2

                LDA PLEXC,X             ; read actual index
                CMP PLEXCOUNT,X         ; plex counter (table dimension) reached?
                BNE PLEXCONT

                LDA #$00                ; reset actual index
                STA PLEXC,X             ; actual index
PLEXCONT        TAY
                LDA BARVALUE,X          ; value of bar (pattern) - read note to play
                CLC
PLEXADD         ADC P0,Y                ; add the tone to note to play

                TAY
                LDA NTL,Y               ; frequency table low of note
                STA $D400,X             ; Voice 1: Frequency control (lo byte)
                LDA NTH,Y               ; frequency table high of note
                STA $D401,X             ; Voice 1: Frequency control (hi byte)
                INC PLEXC,X             ; inc actual plex index
                JMP QUIT

VIBUPDATE
                LDA V1VIB,X             ; vibrato flag
                BNE OKVIB1
                JMP SLIDECHECK

OKVIB1          LDA VIBD,X              ; actual vibrato delay
                BEQ OKVIB

                DEC VIBD,X              ; dec actual vibrato delay
                JMP SLIDECHECK

;================================
; make the vibrato
; Vibrato direction:
;  0 = down (first time)
;  1 = up
;  2 = up
;  3 = down
;  4 = down
;================================
OKVIB           LDA VIBDIR,X            ; vibrato direction flag
                BEQ VIBDOWN1
                CMP #$03
                BCC VIBUP               ; jump if <03
                    ; vibrato down
VIBDOWN         SEC
                LDA C1NLOW,X            ; Voice 1: Frequency control (lo byte) for vibrato
                SBC VIBSTEP,X           ; sub vibrato step
                STA C1NLOW,X            ; Voice 1: Frequency control (lo byte) for vibrato
                STA $D400,X             ; Voice 1: Frequency control (lo byte)

                LDA C1NHIGH,X           ; Voice 1: Frequency control (hi byte) for vibrato
                SBC #0
                STA C1NHIGH,X           ; Voice 1: Frequency control (hi byte) for vibrato
                STA $D401,X             ; Voice 1: Frequency control (hi byte)

                DEC VIBTEMP,X           ; decrement temporary vibrato value
                BNE VIBEND1

                LDA VIBTIME,X           ; read stored value of vibrato time (counter)
                STA VIBTEMP,X           ; set to actual temporary vibrato
                INC VIBDIR,X            ; change vibrato direction flag
                LDA VIBDIR,X            ; vibrato direction flag
                CMP #$05
                BCC VIBEND1             ; jump if <05

                LDA #$01                ; direction up
                STA VIBDIR,X            ; change vibrato direction flag
VIBEND1         JMP QUIT
                                        ; vibrato down high
VIBDOWN1        SEC
                LDA C1NLOW,X            ; Voice 1: Frequency control (lo byte) for vibrato
                SBC VIBSTEP,X           ; sub vibrato step
                STA C1NLOW,X            ; Voice 1: Frequency control (lo byte) for vibrato
                STA $D400,X             ; Voice 1: Frequency control (lo byte)

                LDA C1NHIGH,X           ; Voice 1: Frequency control (hi byte) for vibrato
                SBC #0
                STA C1NHIGH,X           ; Voice 1: Frequency control (hi byte) for vibrato
                STA $D401,X             ; Voice 1: Frequency control (hi byte)

                DEC VIBTEMP,X           ; dec actual temporary vibrato
                BNE VIBEND2

                LDA VIBTIME,X           ; vibrato time (counter)
                STA VIBTEMP,X           ; set to actual temporary vibrato
                INC VIBDIR,X            ; change vibrato direction flag
```

```
VIBEND2         JMP QUIT
                                ; vibrato up
VIBUP           CLC
                LDA C1NLOW,X    ; Voice 1: Frequency control (lo byte) for vibrato
                ADC VIBSTEP,X   ; add vibrato step
                STA C1NLOW,X    ; Voice 1: Frequency control (lo byte) for vibrato
                STA $D400,X     ; Voice 1: Frequency control (lo byte)

                LDA C1NHIGH,X   ; Voice 1: Frequency control (hi byte) for vibrato
                ADC #0
                STA C1NHIGH,X   ; Voice 1: Frequency control (hi byte) for vibrato
                STA $D401,X     ; Voice 1: Frequency control (hi byte)

                DEC VIBTEMP,X   ; dec actual temporary vibrato
                BNE NODRUMS

                LDA VIBTIME,X   ; read stored count value (vibrato)
                STA VIBTEMP,X   ; set to actual temporary vibrato
                INC VIBDIR,X    ; change vibrato direction flag
                BNE NODRUMS
                JMP QUIT
;================================
; Slide timbre routine
;================================
; slide flag:
;  0= none
;  1= down
;  2= up
;  3= down high
;  4= up high
SLIDECHECK      LDA V1SLIDE,X   ; slide flag
                BEQ NOMOREFX
                CMP #$01        ; negative slide (down)
                BEQ SLIDEDOWN2
                CMP #$02        ; positive slide (up)
                BEQ SLIDEUP2
                CMP #$03
                BEQ HIGHDOWN    ; negative only high slide (down)

                CLC
                LDA C1NHIGH,X   ; Voice 1: Frequency control (hi byte) for slide
                ADC SLIDELO,X   ; add slide value
                STA C1NHIGH,X   ; Voice 1: Frequency control (hi byte) for slide
                STA $D401,X     ; Voice 1: Frequency control (hi byte)
                JMP NOMOREFX

SLIDEDOWN2      CLC
                LDA C1NLOW,X    ; Voice 1: Frequency control (lo byte) for slide
                SBC SLIDELO,X   ; sub slide value
                STA C1NLOW,X    ; Voice 1: Frequency control (lo byte) for slide
                STA $D400,X     ; Voice 1: Frequency control (lo byte)

                LDA C1NHIGH,X   ; Voice 1: Frequency control (hi byte) for slide
                SBC #$00
                STA C1NHIGH,X   ; Voice 1: Frequency control (hi byte) for slide
                STA $D401,X     ; Voice 1: Frequency control (hi byte)
                JMP NOMOREFX
                                ; negative slide high (down)
HIGHDOWN        SEC
                LDA C1NHIGH,X   ; Voice 1: Frequency control (hi byte) slide
                SBC SLIDELO,X   ; sub slide value
                STA C1NHIGH,X   ; Voice 1: Frequency control (hi byte) slide
                STA $D401,X     ; Voice 1: Frequency control (hi byte)
                JMP NOMOREFX
                                ; positive slide up
SLIDEUP2        CLC
                LDA C1NLOW,X    ; Voice 1: Frequency control (lo byte) for slide
                ADC SLIDELO,X   ; add slide value
                STA C1NLOW,X    ; Voice 1: Frequency control (lo byte) for slide
                STA $D400,X     ; Voice 1: Frequency control (lo byte)

                LDA C1NHIGH,X   ; Voice 1: Frequency control (hi byte) for slide
                ADC #$00
                STA C1NHIGH,X   ; Voice 1: Frequency control (hi byte) for slide
                STA $D401,X     ; Voice 1: Frequency control (hi byte)

NOMOREFX        LDY SOUND,X     ; index of instrument data
                LDA VDATA+7,Y   ; load instrument effect
                AND #1          ; is effect drum (1)?
                BEQ NODRUMS
                JMP DRUMMOD2
NODRUMS         JMP QUIT


V1VIB           .BYTE   0               ; vibrato flag
V1PLEX          .BYTE   0               ; plex flag
V1SLIDE         .BYTE   0               ; slide flag (0= none, 1= down, 2= up, 3= down high, 4= up high)
PUCH            .BYTE   0               ; pulsation amplitude (00xx/0000 high limit)
```

```
PUCL            .BYTE   0                       ; pulsation amplitude (00xx/0000 low limit)
BEATCOUNT       .BYTE   0                       ; pattern index
PMODDIR         .BYTE   0                       ; direction of pulse modulation
                .BYTE   0,0,0,0,0,0,0,0,0,0,0,0,0

SLIDELO         .BYTE   0                       ; slide value
FADEFLAG        .BYTE   0                       ; not used
NEWDUR          .BYTE   0                       ; new note duration
SOUND           .BYTE   0                       ; index of instrument data
V1PULSELO       .BYTE   0                       ; Actual Wave form pulsation amplitude (lo byte)
PWL             .BYTE   0                       ; Wave form pulsation amplitude (lo byte)
V1PULSEHI       .BYTE   0                       ; Actual Wave form pulsation amplitude (hi byte)
                .BYTE   0,0,0,0,0,0,0,0,0,0,0,0,0

PWH             .BYTE   0                       ; Wave form pulsation amplitude (hi byte)
PLEXTEMP        .BYTE   0                       ; plex index in table
V1LO            .BYTE   0                       ; current track 1 position (base low)
V1HI            .BYTE   0                       ; current track 1 position (base high)
BARCOUNT        .BYTE   0                       ; bar counter (track position)
SEQNUMBER       .BYTE   0                       ; not used
V1DUR           .BYTE   0                       ; actual note length duration voice
                .BYTE   0,0,0,0,0,0,0,0,0,0,0,0,0

TRACK           .BYTE   0                       ; not used
PLAYFLAG        .BYTE   0                       ; not used
TEMPOBYTE       .BYTE   2                       ; cycle timer (speed of song - tempo)
PTEMP           .BYTE   0                       ; Wave form pulsation amplitude step
SPEED           .BYTE   0                       ; actual cycle timer (speed) of song
GATEBYTE        .BYTE   0                       ; set ON/OFF the gate (ADS phase) - mask gate byte

C1NLOW          .BYTE   0                       ; low frequency for vibrato/slide
V1LOFREQ        .BYTE   0                       ; Frequency control (low byte)
V1HIFREQ        .BYTE   0                       ; Frequency control (high byte)
BARVALUE        .BYTE   0                       ; value of bar (pattern)
C1NHIGH         .BYTE   0                       ; high frequency for vibrato/slide/drum
PLEXCOUNT       .BYTE   0                       ; plex counter (table dimension)
PLEXC           .BYTE   0                       ; actual plex counter
                .BYTE   0,0,0,0,0,0,0,0,0,0,0,0,0

VIBDIR          .BYTE   0                       ; vibrato direction
VIBSTEP         .BYTE   0                       ; vibrato step
VIBTIME         .BYTE   0                       ; vibrato time (counter)
VIBTEMP         .BYTE   0                       ; temporary vibrato value
VIBH            .BYTE   0                       ; not used
VIBL            .BYTE   0                       ; not used
TEMP3           .BYTE   0                       ; temp pattern value (note to play)
                .BYTE   0,0,0,0,0,0,0,0,0,0,0,0,0

IMPLEX          .BYTE   0                       ; implex indicator flag
HAT             .BYTE   0                       ; hat indicator flag
VDELAY          .BYTE   0                       ; vibrato delay
VIBD            .BYTE   0                       ; actual vibrato delay
TP              .BYTE   0                       ; transpose (seems to not be used)
TWAVE           .BYTE   0                       ; control register for wave (!=80)
DRUM2           .BYTE   0                       ; control register for drum (=80)
                .BYTE   0,0,0,0,0,0,0,0,0,0,0,0,0

;=======================
; Note frequency table
;=======================
NTL             .BYTE   12,28,45,62,81,102,123,145,169,195
                .BYTE   221,250,24,56,90,125,163,204,246,35
                .BYTE   83,134,187,244,48,112,180,251,71,152
                .BYTE   237,71,167,12,119,233,97,225,104,247
                .BYTE   143,48,218,143,78,24,239,210,195,195
                .BYTE   209,239,31,96,181,30,156,49,223,165
                .BYTE   135,134,162,223,62,193,107,60,57,99
                .BYTE   190,75,15,12,69,191,125,131,214,121
                .BYTE   115,199,124,151,30,24,139,126,250,6
                .BYTE   172,243,230,143,248,46
NTH             .BYTE   1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2
                .BYTE   3,3,3,3,3,4,4,4,4,5,5,5,6,6,7,7,7
                .BYTE   8,8,9,9,10,11,11,12,13,14,14,15,16,17,18
                .BYTE   19,21,22,23,25,26,28,29,31,33,35,37,39,42
                .BYTE   44,47,50,53,56,59,63,67,71,75,79,84,89,94
                .BYTE   100,106,112,119,126,134,142,150,159,168
                .BYTE   179,189,200,212,225,238,253

;=======================
; Plex table index
;=======================
PLEXLH          .WORD   P0,P1,P2,P3,P4,P5,P6
;=======================
; Plex definitions
;=======================
P0              .BYTE   $07,$03,$00
P1              .BYTE   $09,$05,$00
P2              .BYTE   $08,$03,$00
```

```
P3              .BYTE   $18,$0C,$00
P4              .BYTE   $07,$05,$00
P5              .BYTE   $07,$04,$00
P6              .BYTE   $08,$05,$00

;====================
; DRUM TABLE
;====================
DTL             .BYTE   DT&255,BT&255
DTH             .BYTE   DT/256,BT/256
;================================================
; pos 0+2*n: control register (CT)
; pos 1+2*n: high frequency (CT<0 - SET, CT>0 - SUB)
; FF: end of table
;================================================
DT              .BYTE   $81,$30,$11,$02,$41,$04
                .BYTE   $80,$30,$80,$15,$80,$20,$80,$10
                .BYTE   $80,$20,$80,$20,$80,$10,$80,$20,$FF
BT              .BYTE   $81,$30,$41,$03,$40,$03,$80,$20
                .BYTE   $80,$10,$80,$20,$80,$10,$80,$20,$FF
SETIRQ
                SEI
                LDA #INTER&255
                STA $0314
                LDA #INTER/256
                STA $0315
                LDX #$00
                STX $DC0E
                INX
                STX $D01A
                CLI
                RTS

INTER           LDA #$01
                STA $D019
                LDA #$82
                STA $D012
                LDA #$1B
                STA $D011
                LDA #$01
                STA 53280
                JSR DRIVER
                DEC $D020
                JMP $EA31
;-------------------------------------------------------------------------------
                .TEXT   '(C)1988 MG'            ; CHANGED FROM .BYTE TO .TEXT
;-------------------------------------------------------------------------------

VOLUME          .BYTE   0               ; actual volume level
VOLTIME         .BYTE   0               ; volume level
TEM2            .BYTE   0               ; not used
TEM3            .BYTE   5               ; not used
SPEED2          .BYTE   0               ; not used

;================
; Set up the plex
;================
PLEXSETUP       PHA
                AND #$0F
                STA PLEXTEMP,X          ; plex index in table
                PLA
                AND #$F0
                LSR A
                LSR A
                LSR A
                LSR A
                STA PLEXCOUNT,X         ; plex counter

                LDA #$00
                STA PLEXC,X             ; reset actual plex counter

                LDA #1
                STA V1PLEX,X            ; no plex

                LDA #0
                STA V1VIB,X             ; no vibrato
                JMP REGET

;===============
; Set up the vibrato
;===============
VIBSETUP        STA VIBSTEP,X           ; vibrato step
                LDA VDATA2+1,Y
                STA VIBTIME,X           ; set vibrato time (counter)
                STA VIBTEMP,X
                LDA #0
                ;STA V1PLEX,X
                STA VIBDIR,X            ; reset actual vibrato delay
```

```
                LDA #1
                STA V1VIB,X             ; start vibrato
                JMP QUIT

;================================
; instruments part 1
; 0: wave form pulsation amplitude LO/HI -> 00HI/LO00
; 1: Control register
; 2: A/D value
; 3: S/R value
; 4: Wave amplitude inc/dec value
; 5: not used
; 6: Control register 2 (at new instrument and new note start)
; 7: instrument effect
;    1: drum table effect
;    2: a pulse wave effect
;    4: implex (switch between waveform)
;
;   16: hat effect
;================================

VDATA           .BYTE   $87,$11,$00,$E6,$00,$00,$10,$01
                .BYTE   $31,$41,$00,$ED,$15,$00,$40,$02
                .BYTE   $00,$15,$0F,$00,$00,$00,$14,$00
                .BYTE   $71,$41,$00,$8C,$30,$00,$40,$02
                .BYTE   $F1,$41,$0F,$00,$20,$00,$40,$12
                .BYTE   $00,$00,$00,$00,$00,$00,$00,$00
                .BYTE   $00,$11,$00,$A0,$00,$00,$10,$00
                .BYTE   $87,$81,$00,$E8,$00,$00,$80,$01
                .BYTE   $20,$21,$00,$AD,$00,$00,$20,$00
                .BYTE   $44,$41,$00,$7C,$C0,$00,$40,$02
                .BYTE   $00,$80,$00,$A0,$00,$00,$10,$10
                .BYTE   $C0,$41,$00,$9C,$25,$00,$40,$02
                .BYTE   $C0,$41,$00,$9C,$25,$00,$40,$00
                .BYTE   $00,$11,$0F,$00,$00,$00,$10,$00
                .BYTE   $00,$11,$0F,$00,$00,$00,$10,$00
                .BYTE   $F0,$41,$0B,$00,$30,$00,$40,$02
                .BYTE   $31,$41,$00,$8C,$A0,$00,$40,$02
                .BYTE   $00,$21,$00,$8C,$00,$00,$20,$00

;================================
; instruments part 2
; 0: oscillating frequency value (for vibrato)
; 1: length of vibrato intensity (for vibrato)
; 2: Control register for effect implex (4)
; 3: slide value
; 4: slide flag (0= none, 1= down, 2= up, 3= down high, 4= up high)
; 5: drum table index
; 6: wave form pulsation amplitude LO/HI limit -> 00LO/xxxx .. 00HI/xxxx
; 7: not used
;================================

VDATA2          .BYTE   $00,$00,$81,$00,$00,$01,$8E,$00
                .BYTE   $00,$00,$81,$00,$00,$00,$8E,$00
                .BYTE   $00,$00,$81,$00,$00,$00,$8E,$00
                .BYTE   $00,$00,$81,$00,$00,$00,$46,$00
                .BYTE   $00,$00,$81,$00,$00,$00,$33,$00
                .BYTE   $00,$00,$00,$00,$00,$00,$8E,$00
                .BYTE   $00,$00,$81,$00,$00,$00,$8E,$00
                .BYTE   $00,$00,$41,$00,$00,$00,$8E,$00
                .BYTE   $00,$00,$81,$00,$00,$00,$8E,$00
                .BYTE   $90,$02,$81,$00,$00,$00,$35,$00
                .BYTE   $00,$00,$81,$00,$00,$00,$8E,$00
                .BYTE   $90,$02,$81,$00,$00,$00,$27,$00
                .BYTE   $90,$02,$81,$00,$00,$00,$27,$00
                .BYTE   $00,$00,$81,$B3,$03,$00,$8E,$00
                .BYTE   $FF,$08,$81,$00,$00,$00,$86,$00
                .BYTE   $00,$00,$81,$00,$00,$00,$8C,$00
                .BYTE   $80,$02,$81,$00,$00,$00,$8C,$00
                .BYTE   $90,$02,$81,$00,$00,$00,$8C,$00

;=======================================
; pointer to bars (patterns) low address
;=======================================
BARLO           .BYTE   T0&255,T1&255,T2&255,T3&255,T4&255,T5&255
                .BYTE   T6&255,T7&255,T8&255,T9&255,T10&255
                .BYTE   T11&255,T12&255,T13&255,T14&255,T15&255
                .BYTE   T16&255,T17&255,T18&255,T19&255,T20&255,T21&255
                .BYTE   T22&255,T23&255,T24&255,T25&255,T26&255
                .BYTE   T27&255,T28&255,T29&255,T30&255,T31&255
                .BYTE   T32&255,T33&255
                .BYTE   T34&255,T35&255,T36&255,T37&255
                .BYTE   T38&255,T39&255
                .BYTE   T40&255,T41&255,T42&255,T43&255
                .BYTE   T44&255,T45&255
                .BYTE   T46&255,T47&255,T48&255
                .BYTE   T49&255,T50&255,T51&255
                .BYTE   T52&255,T53&255,T54&255,T55&255,T56&255
```

```
;=====================================
; pointer to bars (patterns) high address
;=====================================
BARHI           .BYTE   T0/256,T1/256,T2/256,T3/256,T4/256,T5/256
                .BYTE   T6/256,T7/256,T8/256,T9/256,T10/256
                .BYTE   T11/256,T12/256,T13/256,T14/256,T15/256
                .BYTE   T16/256,T17/256,T18/256,T19/256,T20/256,T21/256
                .BYTE   T22/256,T23/256,T24/256,T25/256,T26/256
                .BYTE   T27/256,T28/256,T29/256,T30/256,T31/256
                .BYTE   T32/256,T33/256
                .BYTE   T34/256,T35/256,T36/256,T37/256
                .BYTE   T38/256,T39/256
                .BYTE   T40/256,T41/256,T42/256,T43/256
                .BYTE   T44/256,T45/256
                .BYTE   T46/256,T47/256,T48/256
                .BYTE   T49/256,T50/256,T51/256
                .BYTE   T52/256,T53/256,T54/256,T55/256,T56/256

;================================
; Songs (tunes) pointers
;================================
VOICE1L         .BYTE   0,TUNE1&255,OVER1&255,FIN1&255
VOICE1H         .BYTE   0,TUNE1/256,OVER1/256,FIN1/256

VOICE2L         .BYTE   0,TUNE2&255,OVER2&255,FIN2&255
VOICE2H         .BYTE   0,TUNE2/256,OVER2/256,FIN2/256

VOICE3L         .BYTE   0,TUNE3&255,OVER3&255,FIN3&255
VOICE3H         .BYTE   0,TUNE3/256,OVER3/256,FIN3/256

;===============
; Make drum
;===============
DRUMMOD2        LDA POINTS              ; track pattern pointer (low)
                PHA                     ; backup it
                LDA POINTS+1            ; track pattern pointer (high)
                PHA                     ; backup it

                LDA VDATA2+5,Y          ; drum table index
                TAY
                LDA DTL,Y               ; drum table low
                STA POINTS
                LDA DTH,Y               ; drum table high
                STA POINTS+1

                LDY DRUM2,X             ; read actual drum table index
                LDA (POINTS),Y          ; read control from table
                BPL DSTAGE2

                CMP #$FF                ; end of table?
                BEQ DEND

DSTAGE3         STA $D404,X             ; Voice 1: Control registers
                INY
                INC DRUM2,X             ; inc actual drum table index
                LDA (POINTS),Y          ; read control from table
                STA $D401,X             ; Voice 1: Frequency control (hi byte)

                INY
                INC DRUM2,X             ; inc actual drum table index
                BNE DEND

DSTAGE2         STA TWAVE,X             ; control register for wave (!=80)
                INY
                INC DRUM2,X             ; inc actual drum table index

                SEC
                LDA C1NHIGH,X           ; high frequency for vibrato/slide/drum
                SBC (POINTS),Y
                STA C1NHIGH,X           ; high frequency for vibrato/slide/drum
                INY
                INC DRUM2,X             ; inc actual drum table index

DEND2           LDA TWAVE,X             ; control register for wave (!=80)
                STA $D404,X             ; Voice 1: Control registers
                LDA C1NHIGH,X           ; high frequency for vibrato/slide/drum
                STA $D401,X             ; Voice 1: Frequency control (hi byte)

DEND            PLA                     ; restore it
                STA POINTS+1            ; track pattern pointer (high)
                PLA                     ; restore it
                STA POINTS              ; track pattern pointer (low)
                JMP QUIT

;=======================
; Song speed
;=======================
TDATA           .BYTE   0,5,3,4
```

```
;=======================
; song patterns
;=======================
;  XX:  pattern XX
; $FF:  repeat the track
; $FE:  end of music
TUNE1           .BYTE   5,5,7,7,7,7,14,14,14,14,14,14,14,17
                .BYTE   15,15,15,15,15,15,15,20,16,18,16,18,16,18,16,19
                .BYTE   16,18,16,18,16,18,16,18
                .BYTE   16,18,16,19,16,18,16,18
                .BYTE   16,18,16,18,16,18,16,18,16,18,16,22
                .BYTE   24,24,24,24,24,24,24,24,27,27,27,27,27,27,27,27
                .BYTE   16,18,16,18,16,18,16,22,27,27,27,27,27,27,27,27
                .BYTE   27,27,27,27,27,27,27,22
                .BYTE   27,27,27,27,27,27,27,22
                .BYTE   27,27,27,27,27,27,27,22
                .BYTE   27,27,27,27,27,27,27,22
                .BYTE   16,18,16,18,16,18,16,18
                .BYTE   15,34,15,34,15,34,15,22
                .BYTE   15,34,15,34,15,34,15,22
                .BYTE   15,34,15,34,15,34,15,22,$FF
TUNE2           .BYTE   6,10,10
                .BYTE   11,11,11,11,12,12,12,12,11,11,11,11,13,13,13,13
                .BYTE   11,11,11,11,12,12,12,12,11,11,11,11,13,13,13,13
                .BYTE   11,11,11,11,12,12,12,12,11,11,11,11,13,13,13,13
                .BYTE   11,11,11,11,12,12,12,12,11,11,11,11,13,13,13,13
                .BYTE   11,11,11,11,12,12,12,12,11,11,11,11,13,13,13,13
                .BYTE   11,11,11,11,12,12,12,12,11,11,11,11,13,13,13,13
                .BYTE   5,10,10,1,3,1,4,1,3,1,4,1,3,1,4,1,3,1,4
                .BYTE   1,3,1,4,1,3,1,4,1,3,1,4,1,3,1,4,1,3,1,4,1,3,21
                .BYTE   11,11,11,11,12,12,12,12,11,11,11,11,13,13,13,13
                .BYTE   11,11,11,11,12,12,12,12,11,11,11,11,13,13,13,13
                .BYTE   11,11,11,11,12,12,12,12,11,11,11,11,13,13,13,13
                .BYTE   26,26,30,30,30,30,33,33,35,36,38,39,37,37,37,37
                .BYTE   40,40,41,41,42,42,43,43
                .BYTE   40,40,41,41,42,42,43,43,44
                .BYTE   40,40,41,41,42,42,43,43
                .BYTE   40,40,41,41,42,42,43,43,44,44,$FF
TUNE3           .BYTE   8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,8,9,8,9,8,9,8,9
                .BYTE   8,9,8,9,8,9,8,9,8,8,8,8,8,8,8,8,1,3,1,4,1,3,1,4
                .BYTE   1,3,1,4,1,3,1,4,1,3,1,4,1,3,1,4
                .BYTE   1,3,1,4,1,3,1,4,1,3,1,4,1,3,1,4
                .BYTE   1,3,1,4,1,3,1,4,1,3,1,4,1,3,1,4,25,23
                .BYTE   1,3,1,4,1,3,1,4,1,3,1,4,1,3,1,4
                .BYTE   28,29,28,29,28,29,28,29,28,29,28,29,28,29,28,29
                .BYTE   28,31,32,29,28,31,32,29
                .BYTE   28,31,32,29,28,31,32,29
                .BYTE   28,31,32,29,28,31,32,29
                .BYTE   28,31,32,29,28,31,32,29
                .BYTE   28,31,32,29,28,31,32,29
                .BYTE   28,31,32,29,28,31,32,29
                .BYTE   28,31,32,29,28,31,32,29
                .BYTE   28,31,32,29,28,31,32,29
                .BYTE   45,45,45,45,45,45,45,45,$FF


;=================================
; pattern data
;=================================
; format:
; $00           : rest
; $01..$6F      : note xx
; $70..$F9      : duration kk-70
; $FA  NN       : select instrument NN
; $FB  MM       : slide down (negative portamento) (-MM)
; $FC  KK       : slide up (positive portamento) (+KK)
; $FD  CI       : plex (arpeggio) CI (C=counter, I=index in table)
; $FF           : end of pattern


T0              .BYTE   $AF,$FA,$05,$00,$FF
T1              .BYTE   $FA,$04,$71,D2,D2,$70,F2,D2,F2,$71,G2,G2,$70,G2
                .BYTE   $FF
T3              .BYTE   $71,A2,F2,$FF
T6              .BYTE   $FA,$02,$EF,$FC,$0A,AS3,$FB,$0A,AS4,$FF
T2              .BYTE   $7F,$FA,$05,$00,$FF
T4              .BYTE   $71,A1,C2,$FF
T5              .BYTE   $FA,$01,$AF,D1,0,$FF
T7              .BYTE   $FA,$01,$7F,D1,G1,D1,G1,$FF
T8              .BYTE   $FA,$04,$71,D2,D2,D2,$70,D2,D2,$71,D2,D2,D2,$70
                .BYTE   D2,D2,$FF
T9              .BYTE   $FA,$04,$71,G2,G2,G2,$70,G2,G2,$71,G2,G2,G2,$70
                .BYTE   G2,G2,$FF
T10             .BYTE   $FA,$03,$FD,$30,$77,D5,0
                .BYTE   $FD,$31,D5,0,$FD,$30,D5,0,$FD,$32,B4,0,$FF
T11             .BYTE   $FA,$06,$FD,$33,$70,A4,F4,D4,F4,$FF
T12             .BYTE   $FA,$06,$FD,$33,$70,B4,G4,D4,G4,$FF
```

```
T13             .BYTE   $FA,$06,$FD,$33,$70,G4,D4,B3,D4,$FF
T14             .BYTE   $FA,$00,$73,C4,C4,C4,C4,$FF
T15             .BYTE   $FA,$00,$73,C4,$FA,$07,D4,$FA,$00,C4,$FA,$07,D4
                .BYTE   $FF
T16             .BYTE   $FA,$00,$71,C4,$FA,$06,$FD,$34,D6
                .BYTE   $FA,$07,D4,$FA,$06,$FD,$34,D6
                .BYTE   $FA,$00,C4,$FA,$06,$FD,$34,D6
                .BYTE   $FA,$07,D4,$FA,$06,$FD,$34,D6,$FF
T17             .BYTE   $FA,$00,$73,C4,C4,$71,C4,$FA,$07,$71,D4,D4,$70
                .BYTE   D4,D4,$FF
T18             .BYTE   $FA,$00,$71,C4,$FA,$06,$FD,$30,D6
                .BYTE   $FA,$07,D4,$FA,$06,$FD,$30,D6
                .BYTE   $FA,$00,C4,$FA,$06,$FD,$30,D6
                .BYTE   $FA,$07,D4,$FA,$06,$FD,$30,D6,$FF
T19             .BYTE   $FA,$00,$71,C4,$FA,$06,$FD,$30,D6
                .BYTE   $FA,$07,D4,$FA,$06,$FD,$30,D6
                .BYTE   $FA,$00,C4,$FA,$06,$FD,$30,D6
                .BYTE   $FA,$07,$70,D4,D4,D4,D4,$FF
T20             .BYTE   $FA,$00,$73,C4,$FA,$07,D4,$FA,$00,$71,C4
                .BYTE   $FA,$07,$70,D4,D4,$71,D4,$70,D4,D4,$FF
T21             .BYTE   $FA,$04,$71,D2,D2,$70,D2,D2,D2,$71,F2,F2
                .BYTE   $70,F2,$71,G2,G2,$FF
T22             .BYTE   $FA,$07,$71,D4,D4,$70,D4,D4,D4,$71,D4,D4
                .BYTE   $70,D4,$71,D4,D4,$FF
T23             .BYTE   $FA,$04,$71,D3,D3,$70,D3,D3,D3,$71,F3,F3
                .BYTE   $70,F3,$71,G3,G3,$FF
T24             .BYTE   $FA,$00,$71,C4,C4,$70,C4,C4,C4,$71,C4,C4,$70,C4
                .BYTE   $71,C4,C4,$FF
T25             .BYTE   $FA,$02,$EF,$FC,$0A,AS4,$DF,$FB,$0A,AS5,$FF
T26             .BYTE   $FA,$09,$F9,$08,$72,A4,D5,F5,A5,$71,F5,A5
                .BYTE   $72,B5,G5,D5,B4,$71,A4,G4,$FF
T27             .BYTE   $FA,$00,$71,C4,C4
                .BYTE   $FA,$07,$71,D4
                .BYTE   $FA,$00,$70,C4,$71,C4,C4,$70,C4
                .BYTE   $FA,$07,$71,D4
                .BYTE   $FA,$00,C4,$FF
T28             .BYTE   $FA,$04,$70,D2,D2,D2,D2,D3,D3,D2,D2,D3,D3
                .BYTE   D2,D2,C3,B2,C3,D3,$FF
T29             .BYTE   $FA,$04,$70,G2,G2,G2,G2,G3,G3,G2,G2,G3,G3
                .BYTE   G2,G2,F3,E3,F3,G3,$FF
T30             .BYTE   $FA,$01,$7F,D2,G2,$FF
T31             .BYTE   $FA,$04,$70,F2,F2,F2,F2,F3,F3,F2,F2,F3,F3
                .BYTE   F2,F2,DS3,D3,DS3,F3,$FF
T32             .BYTE   $FA,$04,$70,C2,C2,C2,C2,C3,C3,C2,C2,C3,C3
                .BYTE   C2,C2,AS2,A2,AS2,C3,$FF
T33             .BYTE   $FA,$03,$77,$FD,$30,D5,0,$FD,$31,C5,0,$FD,$35,C5
                .BYTE   0,$FD,$32,B4,0,$FF
T34             .BYTE   $FA,$00,$73,C4,$FA,$07,D4,$FA,$00,C4,$72,$FA,$07
                .BYTE   D4,$70,D4,$FF
T35             .BYTE   $FA,$0B,$F9,$0A,$75,D4,E4,$73,F4
                .BYTE   $75,A4,G4,$73,F4
                .BYTE   $75,E4,F4,$73,G4
                .BYTE   $77,D4,0,$FF
T36             .BYTE   $FA,$0B,$F9,$0A,$75,D4,E4,$73,F4
                .BYTE   $75,A4,G4,$73,F4
                .BYTE   $75,D5,C5,$73,B4
                .BYTE   $77,B4,0,$FF
T37             .BYTE   $FA,$0C,$F9,$0A,$77,C4,$FC,$0B,C4
                .BYTE   D4,$FB,$0B,D4,$FF
T38             .BYTE   $FA,$0B,$F9,$0A,$75,D5,C5,$73,D5
                .BYTE   $75,F5,D5,$73,F5
                .BYTE   $75,G5,F5,$73,E5
                .BYTE   $77,D5,0,$FF
T39             .BYTE   $FA,$0B,$F9,$0A,$75,D5,C5,$73,D5
                .BYTE   $75,F5,D5,$73,F5
                .BYTE   $75,C6,B5,$73,G5
                .BYTE   $7B,G5,$73,$FB,$90,G5,$FF
T40             .BYTE   $FA,$0B,$FD,$33,$71,F4,D4,A3,D4,$FF
T41             .BYTE   $FA,$0B,$FD,$33,$71,F4,C4,A3,C4,$FF
T42             .BYTE   $FA,$0B,$FD,$33,$71,E4,C4,G3,C4,$FF
T43             .BYTE   $FA,$0B,$FD,$33,$71,D4,B3,G3,B3,$FF
T44             .BYTE   $FA,$0D,$AF,D5,$FA,$0E,D5,$FF
T45             .BYTE   $FA,$0A,$70,1,1,1,1,1,1,1,1,1,1,1,1,1,1,$FF
OVER1           .BYTE   46,$FE,0,0
OVER2           .BYTE   47,0,0
OVER3           .BYTE   48,0,0
T46             .BYTE   $FA,$01,$77,C1,G1,DS1,AS1,$70,F1,$EE,0,$FF
T47             .BYTE   $FA,$0B,$77,$FD,$36,G5,$FD,$34,G5,$FD,$32,G5
                .BYTE   $FD,$31,F5,$FD,$35,$77,F5,$97,0,$FF
T48
                .BYTE   $FA,$09,$77,$FD,$36,G4,$FD,$34,G4,$FD,$32,G4
                .BYTE   $FD,$31,F4,$FD,$35,$77,F4,$E7,0,$FF
FIN1            .BYTE   49,49,49,49,49,49,56,56,49,49,$FE,49,49,49,49,49
                .BYTE   49,49
FIN2            .BYTE   50,51,52,53,$FF
FIN3            .BYTE   56,54,54,55,$FE,54,56
T49             .BYTE   $FA,$00,$73,A3,$FA,$07,$71,D4
                .BYTE   $FA,$00,$73,A3,$71,A3,$FA,$07,$73,D4
```

```
                .BYTE   $FA,$00,$72,A3,$70,A3,$71,$FA,$07,D4
                .BYTE   $FA,$00,$73,A3,$71,A3,$FA,$07,D4,D4,$FF
T50
                .BYTE   $FA,$0F,$71,C2,C3,C3,C2,C3,C2,G2,AS2,$FF
T51             .BYTE   AS1,AS2,AS2,AS1,AS2,AS1,F1,G1,$FF
T52             .BYTE   DS2,DS3,DS3,DS2,DS3,DS2,AS1,C2,$FF
T53             .BYTE   F2,F3,F3,F2,F3,F2,C2,D2,$FF
T54             .BYTE   $FA,$10,$F9,$0A,$75,C5,D5,$73,DS5
                .BYTE   $75,F5,DS5,$73,D5
                .BYTE   $75,DS5,D5,$73,AS4
                .BYTE   $77,AS4,A4
                .BYTE   $75,C5,D5,$73,DS5
                .BYTE   $75,F5,DS5,$73,D5
                .BYTE   $75,DS5,D5,$73,AS4
                .BYTE   $77,C5,0,$FF
T55             .BYTE   $FA,$11,$F9,$10,$75,C5,D5,$73,DS5
                .BYTE   $75,F5,DS5,$73,D5
                .BYTE   $75,DS5,D5,$73,AS4
                .BYTE   $77,AS4,A4
                .BYTE   $75,C5,D5,$73,DS5
                .BYTE   $75,F5,DS5,$73,D5
                .BYTE   $75,DS5,D5,$73,AS4
                .BYTE   $77,C5,$77,0,$FF
T56             .BYTE   $FA,$06,$7F,$FD,$36,G4,$FD,$31,F4
                .BYTE   $FD,$32,G4,$FD,$35,F4,$FF
E               .BYTE   0
```

## Use it

As I know that many that are not so expert in assembly programming can achieve some errors in remove the Dominator data for adding their own tune and so receive not comprehensive errors messages from assembler, I take another version of the source that is full empty.

Inside the addendum file you can find the *empty_public.asm* code. It has all the same comments of the player listed into the previous paragraph, but all data is cleared to an empty state.

The player is configured for having:

- 1 tune only
- 32 empty instruments inserted (from 0 to 31)
- 166 empty patterns inserted (from 0 to 165)
- 32 empty drums tables (from 0 to 31)
- 16 empty plexs (arpeggio) table (prepared for 3+4+2)

So, follow those steps for creating your tune:

STEP 1:

Set your desired tune speed by modify the value at label TDATA (in the example you have to modify the 5 into whatever you want:

```
;=========================
; Song speed
;=========================
TDATA           .BYTE   0,5
```

STEP 2:

Create your instruments by going to modify the empty bytes that are present into:

```
VDATA           .BYTE   $00,$00,$00,$00,$00,$00,$00,$00   ; instrument 0 part 1
                .BYTE   $00,$00,$00,$00,$00,$00,$00,$00   ; instrument 1 part 1
                .BYTE   $00,$00,$00,$00,$00,$00,$00,$00   ; instrument 2 part 1
```

```
                [...]
                .BYTE   $00,$00,$00,$00,$00,$00,$00,$00   ; instrument 31 part 1

VDATA2          .BYTE   $00,$00,$00,$00,$00,$00,$00,$00   ; instrument 0 part 2
                .BYTE   $00,$00,$00,$00,$00,$00,$00,$00   ; instrument 1 part 2
                .BYTE   $00,$00,$00,$00,$00,$00,$00,$00   ; instrument 2 part 2
                [...]
                .BYTE   $00,$00,$00,$00,$00,$00,$00,$00   ; instrument 31 part 2
```

So, if you want to create the instrument number 1 (the one you activate with pattern instruction $FA $01), you have to modify the two rows that has "instrument 1 part 1" and "instrument 1 part 2" comments.

As you can see the source is prepared for 32 instruments, the maximum possible.

STEP 3:

If your instrument uses drum table, you have to modify the empty table at DBT0..DBT31

```
;=================================================
; pos 0+2*n: control register (CT)
; pos 1+2*n: high frequency (CT<0 - SET, CT>0 - SUB)
; FF: end of table
;=================================================
DBT0            .BYTE   $FF
DBT1            .BYTE   $FF
DBT2            .BYTE   $FF
[...]
DBT31           .BYTE   $FF
```

Even here we have a maximum of 32 tables, so one table for instrument in case you use all instruments with a different drum table.

So suppose you want to modify the drum table for index 2 (the value you put in byte at index 5 of VDATA2 into instrument definition), then pick up DTB2 and add all the needed couple of bytes before the $FF end pattern.

For example:

```
DBT2            .BYTE   $81,$30,$11,$02,$FF
```

STEP 4:

If you will use plex (arpeggio) into pattern command $FD, then goes to modify the P0..P15

```
;=================================
; Plex definitions
;
; P0..P8 prepared for 3 notes
; P9..P12 prepared for 4 notes
; P13..P15 prepared for 2 notes
;=================================
P0              .BYTE   $00,$00,$00
P1              .BYTE   $00,$00,$00
[...]
P8              .BYTE   $00,$00,$00

P9              .BYTE   $00,$00,$00,$00
```

```
[...]
P12             .BYTE   $00,$00,$00,$00

P13             .BYTE   $00,$00
P14             .BYTE   $00,$00
P15             .BYTE   $00,$00
```

As you remember the length of a entries is coded inside the $FD following byte when you have that the high nibble is the length, while the low nibble is the index in plex table to use.

So, here a good solution would be to put 16 empty bytes for each Px rows and then you fill the bytes you need. However 95% of arpeggio are based into 3 values and maybe a little few with 2 or 4 values.

Just for commodity, here P0..P8 are predisposed for 3 values, P9..P12 for four and P13..P15 for two. It is a commodity, because it is the length you put inside the $FD commands that effectively choose how many notes to uses.

However suppose you have to use pattern command $FD $31 (3 values for plex at index 1, so P1):

```
P1              .BYTE   $00,$00,$00
```

You have just to modify the above row with the 3 notes to use, likes

```
P1              .BYTE   $08,$03,$00
```

Did you need to have an arpeggio of 7 notes? No problem, freely modify whatever Px you want:

```
P2              .BYTE   $10,$0E,$08,$05,$03,$01,$00
```

and then use the right pattern command: $FD $72

STEP 5:

Fill your pattern data. In the code there is prepared 166 empty patterns, from 0 to 165:

```
;================================
; pattern data
;================================
; format:
; $00            : rest
; $01..$6F       : note xx
; $70..$F9       : duration kk-70
; $FA  NN        : select instrument NN
; $FB  MM        : slide down (negative portamento) (-MM)
; $FC  KK        : slide up (positive portamento) (+KK)
; $FD  CI        : plex (arpeggio) CI (C=counter, I=index in table)
; $FF            : end of pattern
T0              .BYTE   $FF
T1              .BYTE   $FF
T2              .BYTE   $FF
[...]
T165            .BYTE   $FF
```

I think that 165 patterns are enough for a tune, but if you still want to make a 32 minutes long one and goes out of free patterns, please let me know for filling a 256 free empty patterns source

code.

So, all you need it to start to fill the various T0..T165 with your pattern data, letting the $FF to be last byte, like:

```
T0              .BYTE   $FA, $01, $75, G1, $FF
```

STEP 6:

The last point is to fill the tracks with the patterns:

```
;======================
; song patterns
;======================
;  XX:  pattern XX
; $FF:  repeat the track
; $FE:  end of music
TUNE1           .BYTE   $FF
TUNE2           .BYTE   $FF
TUNE3           .BYTE   $FF
```

TUNE1 is the track for voice 1, TUNE2 the track for voice 2 and TUNE3 the track for voice 3.

So, insert all your pattern sequences before the $FF mark and if you want a tune that did not repeat replace $FF with $FE.

Example:

```
TUNE1           .BYTE   T0, T1, T1, T2, T12, $FF
```

STEP 7:

If you are a person that don't like to remember what a number is related to one element you have created, then use constant value for it (before creating a constant, search if that string is already into the source and change it if it matches, otherwise the code did not compile and terminates with error).

Suppose you have created two instruments, the one at index 0 and the second at index 1. The first is a flute, the second a guitar.

In the pattern you actually use $FA, $00 for using a flute and $FA, $01 for using a guitar.

You can define 3 labels (just put them inside the code in the beginning for commodity)

```
INSTR  = $FA
FLUTE  = $00
GUITAR = $01
```

then into the pattern you can use:

```
INSTR, FLUTE
```

or

```
INSTR, GUITAR
```

the same you can make for example for plex.

Instead of $FD, $32 (so, plex of table index 2 with 3 elements: P2 .BYTE $08, $05, $00 ) you can add:

```
ARPEG = $FD
P850  = $32
```

and so the pattern becomes ARPEG, P850

You can apply the same even for note duration command. A duration of $05 (that is a $75 command) becomes DUR5 with this constant:

```
DUR5 = $75
DUR6 = $76
```

So, suppose you have this pattern:

```
$FA, $00, $75, F2, $FD, $32, E2, $FA, $01, F1, $76, G1, $FF
```

It now becomes:

```
INSTR, FLUTE, DUR5, F2, ARPEG, P850, E2, INSTR, GUITAR, F1, DUR6, G1, $FF
```

As you can see it is more easy to read the last pattern instead of the first one.

## Conclusion

At this point you should have all information about the Matt Gray player for being able to create a tune for his competition.

If for unknown reason you will get an error message while compiling the program, please write a post inside this thread for being helped:
http://csdb.dk/forums/?roomid=14&topicid=107150

Have fun in creating a tune and thanks to Matt for making this competition.

# Inside Hunter's Moon

by Stefano Tognon <ice00@libero.it>

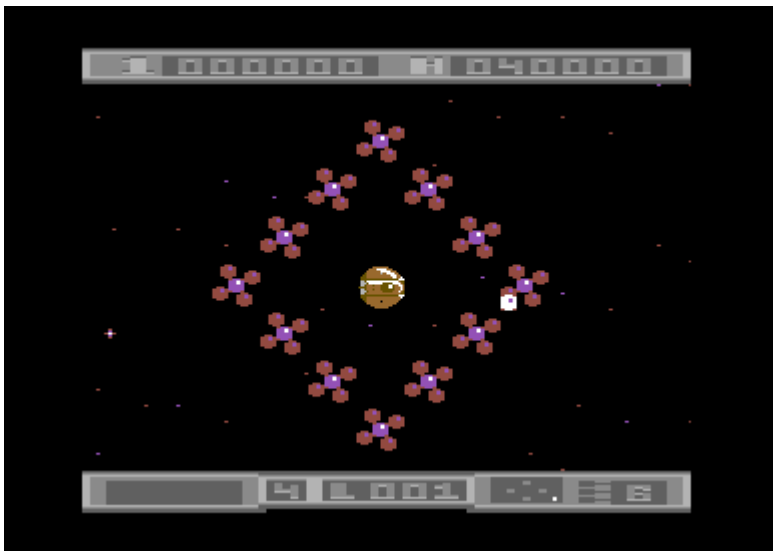
Hunter's Moon is one of the best game ever. It features music by Matt Gray, codes, graphics and sound effects by Martin Walker.

Maybe it seems to be very difficult to play at beginning, but it has the right feeling, and it contains 128 levels!


One point that make this game absolute atmospheric is that sound effects are one of the best created ever: the hives seems to be true living.

If you are wandered how this sound were created, now you have the answers! In tradition of SIDin Magazine now we will see the reverse engineering source code of the Martin Walker sound engine.

Remember that all copyright stay to Martin Walker, so contact him for a businesses use of it.



## Engine

As almost all sound engines, it starts by a routine that initialize itself: **InitEngine**

This routine clear all the SID registers and set volume to maximum. As music is played using Matt Gray engine, the initialization is required to be executed before starting the part of game play (otherwise SID registers may be wrong set).

It also clears the pointers to the indexes that access to the table used by the engine.

In fact the engine uses some table of values for achieving the sound creation and one index for voice is all that is needed for activate it (plus and minus.. the are some other little parameters that we will see later).

So, lets play a sound:

```
lda   #SOUND1
sta   indexInTable

lda   #SOUND2
sta   indexInTable+7

lda   #SOUND3
sta   indexInTable+14
```

We set each index for one voice to the number of sound it must reproduce. That's all.

Else, you can have sound that use only one voice (e.g. **SoundFire**) and other that uses the other two (e.g **soundPlonk**): you can so have the user that fire the enemy and at the same time other sound effects being played.

It is the **playSound** routine (called each Vic frame) that create the sound by calling **setUp-Sound** for each voices.

## Tables

The index that we set for each voices point to many tables of value that the engine uses:

| *Table* | *Description* |
|---|---|
| Attack/Decay (**tableAD**) | Set the Attack/Decay value to put for this sound |
| Sustain/Release (**tableSR**) | Set the Sustain/Reelase value to put for this sound |
| Codified wave pulse width (**tableWave**) | Set the wave pulse width to put for this sound.<br>Pulse has low and high value, this byte is so translated into this form:<br>XY ---> 0YX0 |
| Low frequency (**tableFreqLo**) | Set the low frequency used for this sound |
| High frequency (**tableFreqHi**) | Set the high frequency used for this sound |

| *Table* | *Description* |
|---|---|
| Control (**tableControlReg**) | Set the control value to use for this sound |
| Control2 (**tableControlReg2**) | Set the control value to use at the end of this sound |
| (**tableWaveStep**) | Set the value to add/dec from wave. It cycles from 08xx to 0Exx |
| (**tableLoopValue**) | Duration of loop for frequency effect. At end of loop, it is used the control2 |
| (**tableFreqLoStep**) | Set the value for frequency (low) step (inc/dec) |
| (**tableFreqHiStep**) | Set the value for frequency (high) step (inc/dec) |
| (**tableFreqEffect**) | Select an effect to apply:<br>• 01: frequency add<br>• 02: frequency subtract<br>• 03: looped frequency add<br>• 04: looped frequency subtract<br>• 05: looped frequency add, then sub<br>• 06: looped frequency sub, then add |
| (**tableFreqEffect2**) | Select another effect to apply:<br>• 01: swap lo/hi frequency<br>• 02: sub hi frequency with eor DF<br>• 03: sub hi frequency with eor 12<br>• 04: random add high frequency |

So, let see some example.

The first is the sound of the fire action (you press the button and a bullet is ejected).
It is very simple: it uses only one voice and the index is 12h (see *soundFire*), so the parameters are:

- AD: 00h
- SR: 30h
- Control: 11h
- Control2: 10h
- Loop: 06h
- Frequency low: F0h
- Frequency High: E0h
- Freq. Effect2: 01h

Essentially it plays a triangle sound that change frequency from two fixed value (0Eh and 0Fh)

The second sound is a plonk (*soundPlonk)*. It has two voices that use index 04h and 07h:

- AD: 30h
- SR: 98h
- Control: 15h
- Control2: 14h
- Loop: 08h
- Frequency low: 00h
- Frequency high: F4h
- Frequency low step: 00h

- Frequency high step: 0Dh
- Frequency effect: 05h

This is a vibrato like sound, but is uses ring modulation. The second sound has infact those parameters:

- AD: 00h
- SR: 00h
- Control: 10h
- Control2: 10h
- Loop: 08h
- Frequency low: 11h
- Frequency high: 20h
- Frequency low step: 00h
- Frequency high step: 01h
- Frequency effect: 06h

It has triangular waveform to let ring modulation to take place and the vibrato effect is opposite to the one used into the other voice.

All the other sounds are almost done with the same approach, but there is a variation in the engine that allow to have a long effect apply. This can be see into the *soundExplosion* or *soundStarCell.*

*T*he variation is done by setting pr*eLongEffect* and *longEffect* variables.

At this point it is the time to look at the source code as it is not so complicated as a complete music engine.

## Source Code

```
; Hunter's Moon sound effects by Martin Walker 1987
; extracted from original TAP

  processor 6502

 .org $0882

 .byte "PSID"
 .word $0200          ; version 2
 .word $7C00          ; data offset
 .word $0000          ; load address in cbm format
 .byte >initSongs
 .byte <initSongs
 .byte >playSound
 .byte <playSound
 .word $1000          ; songs
 .word $0100          ; default song
 .word $0000
 .word $0000
 .byte "Hunter's Moon",0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
 .byte "Martin Walker",0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
 .byte "1987 Thalamus",0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
 .word $0000
 .word $0000
 .word $0000

 .byte $00
 .byte $09

 .org $0900

initSongs:
      pha
      jsr  InitEngine
      pla

      tax
      lda  lowTable,x
      sta  jmpAdd+1
```

```
      lda  highTable,x
      sta  jmpAdd+2

jmpAdd:
      jsr  $FFFF
      rts

lowTable:
  .byte <soundExplosion             ; 1
  .byte <soundStarCell              ; 2
  .byte <soundChooseSystem          ; 3
  .byte <soundAltChoose             ; 4
  .byte <soundBell                  ; 5
  .byte <soundShortTick             ; 6
  .byte <soundLongDrill             ; 7
  .byte <soundPlonk                 ; 8
  .byte <soundFire                  ; 9
  .byte <soundBell2                 ; 10
  .byte <soundBonus                 ; 11
  .byte <soundLaunching             ; 12
  .byte <soundAfterBonus            ; 13
  .byte <soundStageStart            ; 14
  .byte <soundMiddle                ; 15
  .byte <soundEnemyExplos           ; 16

highTable:
  .byte >soundExplosion
  .byte >soundStarCell
  .byte >soundChooseSystem
  .byte >soundAltChoose
  .byte >soundBell
  .byte >soundShortTick
  .byte >soundLongDrill
  .byte >soundPlonk
  .byte >soundFire
  .byte >soundBell2
  .byte >soundBonus
  .byte >soundLaunching
  .byte >soundAfterBonus
  .byte >soundStageStart
  .byte >soundMiddle
  .byte >soundEnemyExplos

;================================
; Bell like sound
;================================
soundBell:
      lda  #$00
      sta  flagExplosion
      sta  flagStarCell
      sta  longEffect

      lda  #$16
      sta  indexInTable
      sta  indexInTable+7

      lda  #$17
      sta  indexInTable+14
      rts

;================================
; sound for choosing a new system
;================================
soundChooseSystem:
      lda  #$16
      sta  indexInTable

      lda  #$15
      sta  indexInTable+7

      lda  #$13
      sta  indexInTable+14
      rts

;============================================
; alternative choose system (?)
;============================================
soundAltChoose:
      lda  #$14
      sta  indexInTable

      lda  #$13
      sta  indexInTable+7

      lda  #$17
      sta  indexInTable+14
      rts
```

```
;=============================================
; sound of short tick
;=============================================
soundShortTick:
      lda  #$16
      sta  indexInTable
      sta  indexInTable+7

      lda  #$1C
      sta  indexInTable+14
      rts

;=============================================
; sound of long drill
;=============================================
soundLongDrill:
      lda  #$1E
      sta  indexInTable+14
      rts

;=============================================
; sound of plonk like
; in the game it is played some times when
; activated
;=============================================
soundPlonk:
      lda  #$07
      sta  indexInTable+7

      lda  #$04
      sta  indexInTable+14
      rts

;=============================================
; sound of fire action
;=============================================
soundFire:
      lda  #$12
      sta  indexInTable
      rts

;=============================================
; Called in the source
;=============================================
;      lda  #$16
;      sta  indexInTable
;      rts

;=============================================
; Another bell sound
;=============================================
soundBell2:
       lda  #$01
       sta  indexInTable+7
       lda  #$17
       sta  indexInTable+14
       rts

;=============================================
; Sound in bonus
;=============================================
soundBonus:
       lda  #$09
       sta  indexInTable

       lda  #$1F
       sta  indexInTable+7

       lda  #$12
       sta  indexInTable+14
       rts
;=============================================
; Sound of perpare for launching
;=============================================
soundLaunching:
       lda  #$16
       sta  indexInTable

       lda  #$16
       sta  indexInTable+7

       lda  #$09
       sta  indexInTable+14
       rts

;=========================================
; Sound afther the bonus
```

```
;=========================================
soundAfterBonus:
       lda  #$03
       sta  indexInTable

       lda  #$09
       sta  indexInTable+7

       lda  #$18
       sta  indexInTable+14
       rts

;=========================================
; Sound where game starts
;=========================================
soundStageStart:
       lda  #$0A
       sta  indexInTable+7

       lda  #$0B
       sta  indexInTable+14
       rts

;=========================================
; Middle long sound
;=========================================
soundMiddle:
       lda  #$1D
       sta  indexInTable
       rts

;=========================================
; sound of enemy explosion
;=========================================
soundEnemyExplos:
       lda  #$0F
       sta  indexInTable+7

       lda  #$10
       sta  indexInTable+14
       rts

; codified wave: XY = 0YX0 value
tableWave:
  .byte $00, $00, $00, $00, $00, $00, $00, $00
  .byte $00, $01, $00, $00, $00, $00, $00, $00
  .byte $00, $00, $00, $00, $00, $00, $00, $00
  .byte $00, $00, $00, $00, $00, $18, $00, $00

; Generator Attack/Decay
tableAD:
  .byte $00, $00, $20, $00, $30, $00, $30, $00
  .byte $60, $0E, $00, $0F, $00, $00, $00, $00
  .byte $00, $4F, $00, $0B, $03, $03, $00, $0F
  .byte $0C, $CF, $CF, $CF, $04, $39, $4D, $00

; Generator: Sustain/Release
tableSR:
  .byte $00, $00, $98, $00, $98, $00, $E2, $00
  .byte $E9, $4A, $EC, $ED, $EA, $EA, $E9, $E9
  .byte $A9, $E9, $30, $10, $E9, $76, $00, $EA
  .byte $E8, $AD, $ED, $AD, $25, $E9, $2D, $4A

tableControlReg:
  .byte $00, $10, $15, $10, $15, $10, $15, $10
  .byte $15, $15, $81, $15, $81, $81, $15, $81
  .byte $15, $11, $11, $15, $15, $15, $80, $11
  .byte $15, $81, $81, $81, $11, $41, $41, $15

tableControlReg2:
  .byte $00, $10, $14, $10, $14, $10, $14, $10
  .byte $14, $14, $80, $14, $80, $80, $14, $80
  .byte $14, $10, $10, $15, $14, $14, $80, $14
  .byte $14, $80, $80, $80, $14, $40, $41, $15

tableLoopValue:
  .byte $00, $03, $08, $00, $08, $04, $0B, $08
  .byte $15, $49, $04, $08, $0B, $05, $11, $03
  .byte $03, $03, $06, $16, $1F, $09, $00, $02
  .byte $0A, $35, $28, $23, $02, $02, $02, $06

; Frequency control (lo byte)
tableFreqLo:
  .byte $00, $00, $FF, $00, $00, $00, $00, $11
  .byte $00, $00, $00, $DD, $00, $00, $00, $00
  .byte $00, $1E, $F0, $00, $00, $10, $FF, $9B
  .byte $7B, $00, $00, $00, $AB, $00, $00, $00
```

```
; Frequency control (hi byte)
tableFreqHi:
  .byte $00, $30, $FF, $61, $F4, $3A, $3C, $20
  .byte $C0, $00, $FE, $07, $03, $09, $00, $1F
  .byte $18, $21, $E0, $00, $00, $10, $FF, $9A
  .byte $02, $31, $00, $13, $DD, $4F, $1D, $90

; Frequency (lo) control step (inc/dec)
tableFreqLoStep:
  .byte $00, $00, $00, $00, $00, $00, $00, $00
  .byte $00, $00, $00, $00, $0B, $00, $00, $00
  .byte $00, $00, $00, $00, $00, $00, $00, $00
  .byte $70, $15, $4A, $05, $00, $00, $00, $00

; Frequency (hi) control step (inc/dec)
tableFreqHiStep:
  .byte $00, $00, $0A, $01, $0D, $05, $02, $01
  .byte $0A, $01, $03, $F9, $00, $02, $02, $01
  .byte $F6, $01, $00, $60, $B0, $C8, $00, $00
  .byte $00, $00, $00, $00, $00, $02, $00, $01

tableFreqEffect:
  .byte $00, $00, $01, $05, $05, $03, $05, $06
  .byte $05, $00, $03, $02, $02, $03, $01, $01
  .byte $03, $01, $00, $01, $04, $01, $00, $00
  .byte $06, $02, $02, $02, $00, $05, $00, $03

; Voice: Wave form pulsation amplitude step
tableWaveStep:
   .byte $00, $00, $00, $00, $00, $00, $00, $00
   .byte $00, $00, $00, $00, $00, $00, $00, $00
   .byte $00, $00, $00, $00, $00, $00, $00, $00
   .byte $00, $00, $00, $00, $00, $B1, $04, $00

tableFreqEffect2:
   .byte $00, $00, $01, $00, $00, $00, $00, $01
   .byte $00, $03, $00, $01, $00, $04, $00, $00
   .byte $00, $01, $01, $01, $01, $01, $00, $01
   .byte $03, $00, $00, $00, $01, $03, $03, $03

; Voice 1
indexInTable:    .byte $00        ; index in table
actLoWave:       .byte $00        ; Actual Wave form pulsation amplitude (lo byte)
actHiWave:       .byte $00        ; Actual Wave form pulsation amplitude (hi byte)
control2Reg:     .byte $10        ; Control 2 registers
freqLoStep:      .byte $00        ; Frequency (lo) control step (inc/dec)
controlReg:      .byte $10        ; Control registers
freqHiStep:      .byte $00        ; Frequency (hi) control step (inc/dec)

; Voice 2
                 .byte $00
                 .byte $00        ; Actual Wave form pulsation amplitude (lo byte)
                 .byte $00        ; Actual Wave form pulsation amplitude (hi byte)
                 .byte $80        ; Control 2 registers
                 .byte $00        ; Frequency control step (inc/dec)
                 .byte $80        ; Control registers
                 .byte $03        ; Frequency (hi) control step (inc/dec)

; Voice 3
                 .byte $00
                 .byte $00        ; Actual: Wave form pulsation amplitude (lo byte)
                 .byte $00        ; Actual: Wave form pulsation amplitude (hi byte)
                 .byte $14        ; Control 2 registers
                 .byte $00        ; Frequency control step (inc/dec)
                 .byte $14        ; Voice 3: Control registers
                 .byte $F9        ; Frequency (hi) control step (inc/dec)

; Voice1 (4DB5)
                 .byte $00        ; [Missing!!]
actFreqHi:       .byte $F0        ; Actual: Frequency control (hi byte)
actFreqLo:       .byte $F0        ; Actual: Frequency control (lo byte)
freqEffect2:     .byte $01        ; Frequency efffect 2
actDirWave:      .byte $00        ; direction indicator (0=up 1=down) for Wave form pulsation amplitude step
indexInTable2:   .byte $12        ; copy of index in table
loopValue:       .byte $00        ; loop value for frequency effect

; Voice 2
freqEffect:      .byte $00        ; [Voice 1!!] Frequency efffect
                 .byte $07        ; Actual: Frequency control (hi byte)
                 .byte $00        ; Actual: Frequency control (lo byte)
                 .byte $00
                 .byte $00        ; direction indicator (0=up 1=down) for Wave form pulsation amplitude step
                 .byte $0A
                 .byte $00        ; loop value for frequency effect

; Voice 3 (4DC3)
                 .byte $03        ; [Voice 2!!] Frequency efffect
                 .byte $B9        ; Actual: Frequency control (hi byte)
```

```
                .byte $8F       ; Actual: Frequency control (lo byte)
                .byte $01
                .byte $00       ; direction indicator (0=up 1=down) for Wave form pulsation amplitude step
                .byte $0B
                .byte $00       ; loop value for frequency effect

; Voice 1 (4DCA)
                .byte $02       ; [Voice 3!!]  Frequency efffect
                .byte $00
waveStep:       .byte $00       ; Wave form pulsation amplitude step (inc/dec)
                .byte $00
                .byte $00
                .byte $00
                .byte $00

                .byte $00
                .byte $00
                .byte $00       ; Wave form pulsation amplitude step (inc/dec)
                .byte $00
                .byte $00
                .byte $00
                .byte $00

                .byte $00
                .byte $00
                .byte $00       ; Wave form pulsation amplitude step (inc/dec)
                .byte $00
longEffect:     .byte $00       ; "long effect": make long effect after init of sid
flagExplosion:  .byte $00       ; explosion sound indicator
flagStarCell:   .byte $00       ; explosion sound indicator

;=================================
; Init the music engine
;=================================
InitEngine:
      lda  #$00
      sta  indexInTable2
      sta  indexInTable2+7
      sta  indexInTable2+14

      sta  indexInTable
      sta  indexInTable+7
      sta  indexInTable+14

      sta  loopValue            ; loop value for frequency effect
      sta  loopValue+7
      sta  loopValue+14

      sta  longEffect
      sta  flagExplosion
      sta  flagStarCell

      ldy  #$18
      lda  #$00
loopZero:
      sta  $D400,y              ; SID: clear all registers
      dey
      bpl  loopZero

      lda  #$0F                 ; max volume
      sta  $D418                ; Select volume and filter mode
      rts

;=================================
; Play routine
;=================================
playSound:
      dec  longEffect
      lda  longEffect
      cmp  #$FF
      bne  checkFlag

      lda  #$00
      sta  longEffect

checkFlag:
      bit  flagExplosion
      bpl  skipSoundEcplosion
      jsr  soundExplosion
      jmp  skipStarCell

skipSoundEcplosion:
      bit  flagStarCell
      bpl  skipStarCell
      jsr  soundStarCell

skipStarCell:
      ldx  #$00                 ; voice 1
```

```
      jsr  setUpSound

      ldx  #$07                  ; voice 2
      jsr  setUpSound

      ldx  #$0E                  ; voice 3
      jsr  setUpSound

      bit  preLongEffect
      bpl  skipSetupLongEffect

      lda  #$32                  ; set up long effect
      sta  longEffect

skipSetupLongEffect:
      lda  #$00                  ; make long effect now
      sta  preLongEffect
      rts

preLongEffect:                   ; to activate for a long effect of sound
      .byte $00

;================================
; Setup sound effect for a voice
;================================
setUpSound:
      cpx  #$00                  ; is voice 1?
      beq  testIndex

      lda  longEffect
      beq  testIndex
                                 ; clear table postion (for long effect)
      lda  #$00
      sta  indexInTable,x
      jmp  skipOutSidValue

testIndex:
      lda  indexInTable,x
      beq  skipOutSidValue

      jsr  outSidValue

skipOutSidValue:
      jsr  makeLoop
      jsr  testAndMakeEffect
      rts


;====================================
; Sound effect for explosion
;====================================
soundExplosion:
      lda  #$00
      sta  flagExplosion
      sta  longEffect
                                    ; set explosion (0C/0D/0E)
      lda  #$0C
      sta  indexInTable

      lda  #$0D
      sta  indexInTable+7

      lda  #$0E
      sta  indexInTable+14

      lda  #$FF
      sta  preLongEffect
      rts

;================================
; Sound Effect for taking a star
; cell
;================================
soundStarCell:
      lda  #$00
      sta  flagStarCell
      sta  longEffect

      lda  #$07
      sta  indexInTable+7

      lda  #$08
      sta  indexInTable+14

      lda  #$FF
      sta  preLongEffect
      rts
```

```
;================================
; Output the sid value
;================================
outSidValue:
      lda  #$00
      sta  $D404,x               ; SID: Control registers

      lda  indexInTable,x
      sta  indexInTable2,x
      tay

      lda  tableFreqLo,y
      sta  $D400,x               ; SID: Frequency control (lo byte)
      sta  actFreqLo,x           ; Actual: Frequency control (lo byte)

      lda  tableFreqHi,y
      sta  $D401,x               ; SID: Frequency control (hi byte)
      sta  actFreqHi,x           ; Actual: Frequency control (hi byte)

      lda  tableLoopValue,y
      sta  loopValue,x           ; loop value for frequency effect

      lda  tableWave,y
      pha
      and  #$F0
      sta  $D402,x               ; SID: Wave form pulsation amplitude (lo byte)
      sta  actLoWave,x           ; Actual Wave form pulsation amplitude (lo byte)
      pla
      and  #$0F
      sta  $D403,x               ; SID: Wave form pulsation amplitude (hi byte)
      sta  actHiWave,x           ; Actual Wave form pulsation amplitude (hi byte)

      lda  tableAD,y
      sta  $D405,x               ; SID: Generator 1: Attack/Decay
      lda  tableSR,y
      sta  $D406,x               ; SID: Generator 1: Sustain/Release

      lda  tableControlReg,y
      sta  controlReg,x
      lda  tableControlReg2,y
      sta  control2Reg,x

      lda  tableFreqLoStep,y
      sta  freqLoStep,x          ; Frequency (lo) control step (inc/dec)
      lda  tableFreqHiStep,y
      sta  freqHiStep,x          ; Frequency (hi) control step (inc/dec)

      lda  tableFreqEffect,y
      sta  freqEffect,X          ; Frequency efffect

      lda  tableWaveStep,y
      sta  waveStep,X            ; Wave form pulsation amplitude step (inc/dec)

      lda  tableFreqEffect2,y
      sta  freqEffect2,x         ; Frequency efffect 2

      lda  controlReg,x
      sta  $D404,x               ; SID: Control registers

      lda  #$00
      sta  indexInTable,x
      rts

;================================
; Make the loop. On finish can
; change the control register
;================================
makeLoop:
      lda  loopValue,x           ; loop value for frequency effect
      bne  decLoopValue

      lda  control2Reg,x
      cmp  controlReg,x
      bne  changeControl
      rts

changeControl:
      sta  $D404,x               ; SID: Control registers
      sta  controlReg,x
      rts

decLoopValue:
      dec  loopValue,x           ; loop value for frequency effect
      rts

;================================
; Test and make effect for timbre
; according to the actual values
```

```
; in table
;================================
testAndMakeEffect:
      lda  indexInTable2,x
      bne  skipZero
      rts

skipZero:
      lda  waveStep,x            ; Wave form pulsation amplitude step (inc/dec)
      beq  skipPulseTimbre
      jsr  pulseTimbre

skipPulseTimbre:
      lda  freqEffect,x          ; Frequency efffect
      beq  skipMakeFreqEffect

      jsr  makeFreqEffect

skipMakeFreqEffect:
      lda  freqEffect2,x         ; Frequency efffect 2
      bne  makeFreqEffect2_
      rts

makeFreqEffect2_:
      jsr  makeFreqEffect2
      rts

;================================
; pulse-width timbre routine
;================================
pulseTimbre:
      lda  actDirWave,x          ; direction indicator (0=up 1=down) for Wave form pulsation amplitude step
      bne  decWave

      clc
      lda  actLoWave,x           ; Actual Wave form pulsation amplitude (lo byte)
      adc  waveStep,x            ; Wave form pulsation amplitude step (inc/dec)
      sta  actLoWave,x           ; Actual Wave form pulsation amplitude (lo byte)
      sta  $D402,x               ; SID: Wave form pulsation amplitude (lo byte)

      lda  actHiWave,x           ; Actual Wave form pulsation amplitude (hi byte)
      adc  #$00
      sta  actHiWave,x           ; Actual Wave form pulsation amplitude (hi byte)
      sta  $D403,x               ; SID: Wave form pulsation amplitude (hi byte)
                                 ; test for high limit reached
      cmp  #$0E                  ; high value limit for pulsation amplitude
      bcc  exitPulseTimbre
                                 ; change direction to down
      lda  #$01
      sta  actDirWave,x

decWave:
      sec
      lda  actLoWave,x           ; Actual Wave form pulsation amplitude (lo byte)
      sbc  waveStep,x            ; Wave form pulsation amplitude step (inc/dec)
      sta  actLoWave,x           ; Acutal Wave form pulsation amplitude (lo byte)
      sta  $D402,x               ; SID: Wave form pulsation amplitude (lo byte)

      lda  actHiWave,x           ; actual Wave form pulsation amplitude (hi byte)
      sbc  #$00
      sta  actHiWave,X           ; Actual Wave form pulsation amplitude (hi byte)
      sta  $D403,x               ; SID: Wave form pulsation amplitude (hi byte)
                                 ; test for low limit reached
      cmp  #$08                  ; low limit for pulsation amplitude
      bcs  exitPulseTimbre
                                 ; change direction to up
      lda  #$00
      sta  actDirWave,x          ; direction indicator (0=up 1=down) for Wave form pulsation amplitude step
exitPulseTimbre:
      rts

;================================
; Make frequency effect according
; to the A value:
;
; 01: frequency add
; 02: frequency subtract
; 03: looped frequency add
; 04: looped frequency substract
; 05: looped frequency add, then sub
; 06: looped frequency sub, then add
;================================
makeFreqEffect:
      cmp  #$01
      bne  test02

addFreq:                         ; add frequency
      clc
```

```
      lda  actFreqLo,x           ; Actual: Frequency control (lo byte)
      adc  freqLoStep,x          ; Frequency (lo) control step (inc/dec)
                                 ; missing store????
      sta  $D400,x               ; SID: Frequency control (lo byte)

      lda  actFreqHi,x           ; Actual: Frequency control (hi byte)
      adc  freqHiStep,x          ; Frequency (hi) control step (inc/dec)
      sta  actFreqHi,x           ; Actual: Frequency control (hi byte)
      sta  $D401,x               ; SID: Frequency control (hi byte)
      rts

test02:
      cmp  #$02
      bne  test03

subFreq:                         ; subtract frequency
      sec
      lda  actFreqLo,x           ; Actual: Frequency control (lo byte)
      sbc  freqLoStep,x          ; Frequency (lo) control step (inc/dec)
      sta  actFreqLo,x           ; Actual: Frequency control (lo byte)
      sta  $D400,x               ; SID Frequency control (lo byte)

      lda  actFreqHi,x           ; Actual: Frequency control (hi byte)
      sbc  freqHiStep,x          ; Frequency (hi) control step (inc/dec)
      sta  actFreqHi,x           ; Actual: Frequency control (hi byte)
      sta  $D401,x               ; SID: Frequency control (hi byte)
      rts

test03:
      cmp  #$03
      bne  test04

addFreqLoop:
      lda  loopValue,x           ; loop value for frequency effect
      bne  addFreq
      rts

test04:
      cmp  #$04
      bne  test05

subFreqLoop:
      lda  loopValue,x           ; loop value for frequency effect
      bne  subFreq
      rts

test05:
      cmp  #$05
      bne  test06

      lda  loopValue,X           ; loop value for frequency effect
      bne  addFreq
      beq  subFreq

test06:
      cmp  #$06
      bne  exitMakeFreqEffect

      lda  loopValue,X           ; loop value for frequency effect
      bne  subFreq
      beq  addFreq
exitMakeFreqEffect
      rts

;================================
; Make frequency efffect 2
; according to A value
;
; 01: swap lo/hi frequency
; 02: sub hi frequency with eor DF
; 03: sub hi frequency with eor 12
; 04: random add high frequency
;================================
makeFreqEffect2:
      cmp  #$01
      bne  test02_

swapLoHiFreq:
      lda  actFreqLo,x           ; Actual: Frequency control (lo byte)
      ldy  actFreqHi,x           ; Actual: Frequency control (hi byte)
      sta  actFreqHi,x           ; Actual: Frequency control (hi byte)
      sta  $D400,x               ; SID: Frequency control (lo byte)
      tya
      sta  actFreqLo,x           ; Actual: Frequency control (lo byte)
      sta  $D401,x               ; SID: Frequency control (hi byte)
      rts

test02_:
```

```
      cmp  #$02
      bne  test03_

      sec
      lda  actFreqHi,x            ; Actual: Frequency control (hi byte)
      sbc  freqHiStep,x           ; Frequency (hi) control step (inc/dec)
      eor  #$DF
      sta  actFreqHi,x            ; Actual: Frequency control (hi byte)
      sta  $D401,x                ; SID Frequency control (hi byte)
      rts

test03_:
      cmp  #$03
      bne  test04_

      sec
      lda  actFreqHi,x            ; Actual: Frequency control (hi byte)
      sbc  freqHiStep,x           ; Frequency (hi) control step (inc/dec)
      eor  #$12
      sta  actFreqHi,x            ; Actual: Frequency control (hi byte)
      sta  $D401,x                ; SID: Frequency control (hi byte)
      rts

test04_:
      cmp  #$04
      bne  exitMakeFreqEffect2

randomAddFreq:
      lda  $D41B                  ; Random numbers generator oscillator 3
      and  #$0F
      clc
      adc  actFreqHi,x            ; Actual: Frequency control (hi byte)
      sta  $D401,x                ; SID: Frequency control (hi byte)
      rts

exitMakeFreqEffect2:
      rts
```

# Conclusion

Before I reverse engineering the Martin code, I was sure it was a very complicated engine as the sound is the most realistic you can find into a game. But the engine is simple and the secret is just how the sid is manipulated in a convenient way :)

The other secret is that he uses even more SID voices for some effects, so the sound is more elaborated (and maybe this is why music is not played during game play: have all the 3 voices used for the effects will destroy too much a running music in background).

*Din 14 end*