“Michelle Hunzinker”

Vice snapshot with CCS64 palette

Made with the GIMP from a MH photo
and converted to C64 320x200
HiRes Mode Bitmap
by Stefano Tognon
in 2003

“Why Sample?”
...

Free Software Group

SVDin 4
version 1.00
21 June 2003

General Index

Editorials....4
News....5
- NinjaTracker V1.04....5
- HardSID Quattro PCI....5
- TSID2 0.2....5
- SidPlug 1.0b2 Mac OS X....6
- Goattrk 1.4 Mac OS X....6
- SidPlay 3.12 Mac OS X....6
- PSID64 0.6....6
- SidWine 2 Compo....7
- HVSC 5.3....7
- SIDBrowser v2.2....7
- Steppe's and Yodelking's Ripcompo....8

Chris Abbott Interview!....9
Ripping BASIC program into RSID....12
- RSID rip....13
- RSID code....14
- Conclusion....22

Martin Galway's Arkanoid music routine....23
- Songs....23
- Tracks/Patterns....23
- Instructions/Notes....24
- Note and Duration....25
- Instrument Table....25
- Frequencies Timbre....27
- Rectangular Wave Timbre....28
- Filter Table....29
- Other Instrument Effects....30
- Game Sound Effects....30
- Sample....31
- Source....32
- Conclusion....98

Editorials

Stefano Tognon <ice00@libero.it>

Hi, again.

Finally in this issue the Arkanoid reverse engineering work is ready. Some points remain a little unclear, but maybe the most of the engine is completed.

I very hope that into recreating the 5000 and over lines of the source code, no error was inserted. If you find something musically not correct, let me know for a fix onto the engine.

As you see this number is very big into pages number used, but I prefer to insert all the source code of the Martin engine, instead of a little part :)

However this work had required many times, and so the planned second part of Pattern Searching will skip to the next issue, as I have not manage all the features of the new kind of searching method.

So, I have insert another Basic ripping article: in this case we see how RSID rip can be used for converting Basic music that did not use any kind of timing: the rip cannot be equal to the original, but maybe it is better that nothing.

Remember that the copyright of the presented codes remain to the original authors: contact them for a business use of it.

The last thing: as I can make some errors while I make this magazine, if you find somethings wrong, let me know. I have already made the version 1.1 of each issue with some corrections that will be released as soon as I find the time to complete the task.
In the file at http://digilander.iol.it/ice00/tsid/sidin/revision.txt there are the already fixed points.

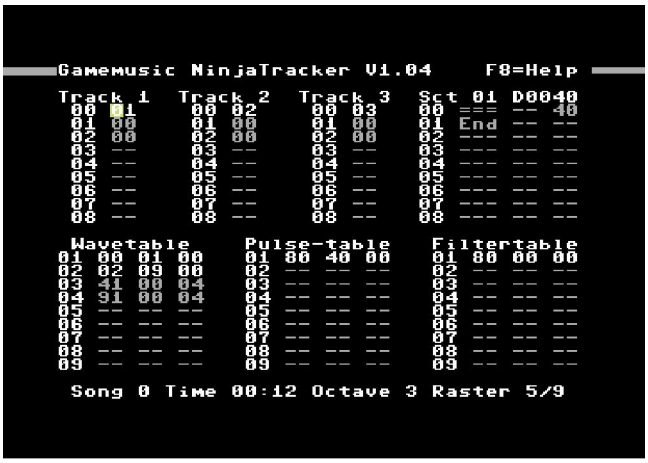
Bye
S.T.

News

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

- NinjaTracker V1.04
- HardSID Quattro PCI
- TSID2 0.2
- SidPlug 1.0b2 Mac OS X
- Goattrk 1.4 Mac OS X
- Sidplay 3.12 Mac OS X
- PSID64 0.6
- SidWine 2 compo
- HVSC 5.3
- SIDBrowser v2.2
- Steppe's and Yodelking's Ripcompo

NinjaTracker V1.04



Some changes are made in this version of the tracker released on 5 March 2003 by Cadever: brings back the filter-command, new gamemusic version that doesn't save playroutine with music (playroutine separately as source code, can play music from any address) and doesn't lie about the rastertime anymore :)

Else, GoatTracker -> NinjaTracker Convertor V1.1 is released: now tunes shouldn't bug anymore after packing/relocation, and support for almost all the effects added.

Download the tracker from http://covertbitops.c64.org/tools/ninjatrk.zip and the converter from http://covertbitops.c64.org/tools/goatninj.zip

HardSID Quattro PCI

The PCI version of the HardSid Quattro card is now available to buy at http://www.hardsid.com

There are already available the 32-bit kernel-mode WDM MIDI drivers for Windows XP / Windows 2000 and updated MIDI drivers for Win98/ME

TSID2 0.2

Even if this new version of TSID2 is to be considerate for test porpoise, now the library that collect time statistic of listened sids is more stable.

Download the stuff from http://www.sf.net/projects/tsid

SidPlug 1.0b2 Mac OS X

Available from 13 April 2003, this sidplayer plug-in supports all the major OS X web browsers (Safari, Camino, Omniweb, MSIE, etc.). Features oscilloscope output in the browser window and is built using libsidplay2, so it' s able to load RSID files.

Download from http://samhain.c64.org/~sid/sidplay/sidplug-osx-1.0b2.dmg.gz

Goattrk 1.4 Mac OS X

Available from 13 April 2003, this is the port for Mac OS X of Goattrk 1.4. Supports output via reSID or the Catweasel card.

Download from http://samhain.c64.org/~sid/sidplay/goattracker-osx-1.4.dmg.gz

SidPlay 3.12 Mac OS X

Available from 13 April 2003, SIDPLAY for Mac OS X has the following features:

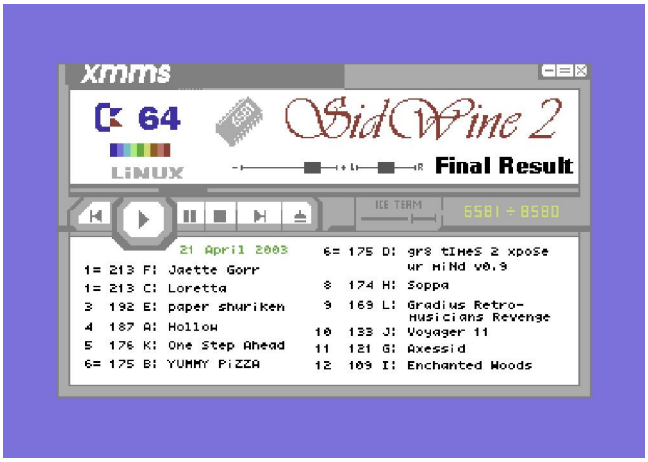
- Includes the SIDPLAY playback and emulation engine by Michael Schwendt (libsidplay 1.36.57)
- Can optionally use the SIDPLAY playback library with the reSID emulation engine by Dag Lem (reSID 0.13)
- Includes the SIDPLAY2 playback engine by Simon White (libsidplay 2.1.0, cvs from 20-Oct-2002)
- Support for the SID chip on the Catweasel MK3 PCI card
- Built-in directory browser, ideal for browsing the HVSC
- Playlist support, with built-in playlists for HVSC Top 100 and Best of VARIOUS
- Favorites playlist
- Support for the HVSC songlength database (put Songlengths.txt into DOCUMENTS folder of HVSC)
- Fast search function, can search the whole HVSC in seconds
- Built-in STIL viewer, based on STILView 2.17 by Imre ' LaLa' Olajos jr.
- AIFF export function
- MP3 export function (using the LAME mp3 encoder)
- PRG C64 executable export (using PSID64 by Roland Hermans)
- Oscilloscope and SID register views
- First-ever graphical adjustment of reSID' s filter emulation
- Oversampling support in SIDPLAY and SIDPLAY2
- Automatic HVSC update functionality
- Full tempo control

Download from http://samhain.c64.org/~sid/sidplay/SIDPLAY-3.1.2.dmg.gz

PSID64 0.6

The new version of the program that convert a Sid tune into a C64 runnable program is available here: http://www.sf.net/projects/psid64.

SidWine 2 Compo



The second SidWine C64 online music competition is terminated, with two winning tunes at the first place.

You can see all the event by looking at: http://digilander.iol.it/ice00/tsid/sidwine2

A special result pack with all voters comments, statistics, all the tunes in psid and prg format, and a demo disk is available:

http://digilander.iol.it/ice00/tsid/sidwine2/sidwine2_res.zip

The final classification:

Pos.	Points	Title	Author
1=	213	Jaette Gorr	Peter Sanden & Per Bolmstedt
1=	213	Loretta	Stellan Andersonn (Dane/Crest)
3	192	paper shuriken	Ronny Engmann (Dalezy/Creators)
4	187	Hollow	Kenho
5	176	One Step Ahead	Michal Hoffmann (Smalltown Boy)
6=	175	YUMMY PiZZA	Opaque Fear (Thomas Sorensen)
6=	175	gr8 tImeS 2 xpoSe ur miNd v0.9	Turtle / TDM
8	174	Soppa	Hukka (Pietari Toivonen)
9	169	Gradius Retromusicians Revenge	Luca Carrafiello (Luca/Fire)
10	133	Voyager 11	Pater Pi (Stephan Drost)
11	121	Axessid	Hukka (Pietari Toivonen)
12	109	Enchanted Woods	Warren Pilkington (Waz/Padua)

HVSC 5.3

Update 35 of the High Voltage Sid Collection is available at http://www.hvsc.c64.org

This update contains:

```
276 new SIDs
102 fixed/better rips
 38 fixes of PlaySID/Sidplay1 specific SIDs (ie: RSID files)
  6 repeats/bad rips eliminated
368 SID credit fixes
```

for a total of 20660 sids.

SIDBrowser v2.2

From http://www.sidbrowser.com the new version of SIDBrowser can be downloaded
SIDBrowser is the easiest way to browse into HVSC, with features like STIL, sid tune length, playlists, favorites paths.

Steppe's and Yodelking's Ripcompo

Steppe's and Yodelking's Ripcompo is over. 65 SID HUNTs, 8 bugged tunes and 88 RSID rips have been done by the participators:

```
 1:  Ninja /The Dreams                    123 points
 2:  Inge Pedersen /HVSC crew              74 points
 3:  Adam Lorentzon /HVSC crew             73 points
 4:  Smalltown Boy /HVSC crew/MSL          59 points
 5:  Peter Sandén /HVSC crew               30 points
 6:  Jan Krolzig                           16 points
 7:  Stefano Tognon                        15 points
 8:  Magnate /Obsession                     9 points
 9:  St0fF /Neoplasia                       7 points
10: Murdock /Tropyx/Draco/Cascade           5 points
10: Zyron /F4CG/Nostalgia/Oxsid Planetary   5 points
10: Steppe /HVSC-crew                       5 points
13: Graeme Norgate                          3 points
14: Roland Hermans                          2 points
14: Fenek / Arise                           2 points
```

Chris Abbott Interview!

by Stefano Tognon

In this issue you can read an interview made with Chris Abbott that was done by some emails into this month. If you don't know Chris as a Sid composer, I'm sure that you know his C64Audio related things like Back in Time Cds and Bit Live Events!

Hello Chris,
why not start by giving us some information about you?

Born 1970, Ilford Essex. No particular music education, but went to good school where you learnt an instrument in the first year (guitar). Got keyboard at 15 and played Axel F all the time. First computer was an Atari 400 with 16k memory. Borrowed a friend's C64 and fell in love with Hunchback and Attack of the Mutant Camels. Got C64 eventually. Had MIDI interface and sequencer for the Atari 800 and the C64. Programmed some SIDs for Superior Software which are in HVSC. Left school with many A Levels. Went to University to study psychology. Married Tanya in 1990. Worked for IBM during 1990. Did no music until 1994 when I got an AWE32.
Started programming MIDIs of C64 tunes. Hooked up with crap German publishing company for possible C64 CD (I didn't know they were crap at the time, though I should have done!). Eventually did it alone, hence Back in Time 1. Many CDs follow. Back in Time Live.
Had twins (John and Stephanie) in 2002. A drain on the productivity, but they're beautiful children, and a lot of fun :) I spend most of my time looking after them.

During all that time I've been an IT contractor, doing various programming, including being the only person in the world to have cracked AOLs "Rainman" protocol. Whew :)

We have recently see that C64Audio.com has paid over £20,000 to C64 composers since 1997 in royalties and other payments. Look forward at 1997: did you believe that this could be possible?

Actually, yes. I even hoped for more than that: which would have been possible if most of the people who bought C64s had remembered more of the tunes.
However, the CD market was fairly limited, because most of the people who had C64s aren't particularly nostalgic for them, I think. I run into people who had C64s all the time, but they don't remember the music much, and certainly never had the same longing for CD versions that I had all through the period from 1985 -> 1997.

What is more easy to do: a Back in Time CD or a Bit Live event? An the most fun?

A BIT Live Event isn't fun for me. Deciding what things to do is fun. The actual event mostly consists of moving heavy objects around and running around London trying to find a video adapter before the shops close! And explaining to everyone why there are either too many or too few people. I enjoyed BIT Live 2 though.
I see my mission to make sure everyone else has fun. I'm naturally a bit of a party pooper.

What are the new CDs music that you are planned to produce? (if you can anticipate something)

"The Galway Project - The Perfect SID Collection"
"Back in Time 4 (Martial Arts)"
"Back in time 5 (Fantasy)"
"Back in Time 6 (completion)"
"Music for Starchildren (a new age CD for children that coincidentally includes some C64 stuff)"
"Crystal Dreamscapes 2"
"01:04 Eternal, by FTC"
Some album by Makke

Is that enough? :) ("Yes" shout the audience)...

Last minute question: is there any possibility from you to compose a new Sid music in the C64 like you did in the past?

I'd love to, but as you can see above, my time is limited for musical things.

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

Emulators are great, and often a lot more convenient that the real thing. Sometimes though you can't beat sitting in front of a TV with a real joystick and a real SID playing :)

6581 vs 8580 chip: any (musical) preference?

6581. 8580s filters don't love my ears :)

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

There was a pretty crap Magnetic Fields one I did in the Software House demo. The best was probably the Chess stuff. That was pretty damn complex!
I'm quite pleased with Galaforce 2, too. Though of course it was Martin Galway's soundtrack that made it into the game *fume* :)

Who are your best sid authors?

Unfair to ask :) Martin Galway and Rob Hubbard hit the spot more times than most, because they combined experience, hard work, technical genius and sheer musicality.
Everyone else lacked at least one (and sometimes more) of these qualities. In the old days, my favourite SID artists were Graeme Hansford, Chris Cox, James Lisney, Paul Norman and David Dunn. I'm still fond of these people even now :)

What are the best sids ever in your opinion?

Thing on a Spring was without doubt the most influential SID ever made, though I prefer Gerry the Germ, Spellbound and Kentilla. From Martin, I loved Hypersports and Roland's Rat Race, first of all. Fred Gray's finest moments were Mutants, Shadowfire and Frankie Goes to Hollywood. From Ben Daglish I loved Trap, Kettle and William Wobbler. From Dave Whittaker I liked 180 and BMX Simulator. There's a huge list :)

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

People don't understand that it's only through the sensible generation and use of money that the SID scene is as well integrated as it is. The CDs give the scene journalistic credibility, as do the live events. And do you think the composers would have been as keen to take part if there was no possible money in it? Would Martin Galway have lent us his C128D and source disks if there was no commercial project to build with it?
Would Rob Hubbard be performing unplugged unless he had been bought back into the scene by BIT Live?

People who have supported the CDs and live events make all these things possible, so it makes me pretty damn angry when people spread lies about C64Audio and what we do because of malice or ignorance.

As for the "C64 music should be free" movement: free C64 music is a privilege, not a right.
It's only through the generosity of copyright holders and composers that RKO and HVSC exist and flourish.

Webography:

C64Audio:
www.C64Audio.com

Ripping BASIC program into RSID

by Stefano Tognon <ice00@libero.it>

I had have recently the opportunity to produce another rip of a Basic game: Telengard.

The game is all in Basic and it is a text based adventure game with quite interesting sound in it.

Ninja had already produce a rip of the this game, going extracting the Basic part that manage music from all the game, but this cannot be executed into a Sidplayer emulator as it is not in machine code (however it can be run in a C64 emulator).

My work was so to convert the Basic rip into machine code: but as we can see after, this is more difficult that in Testcard (see the last issue) as no interrupts are used in the game and also Basic TI variable is used only in some part.

BASIC code

Before going into details it is better to look at the basic rip:

```
0 rem ripped by ninja/the dreams in 2003
10 dim nt%(59):tf=256:th=255: gosub 50000
20 a=peek(2)+1:i=1:on a gosub 50200,50600,50100:end
19000 ti$="000000"
19002 if ti<90then 19002
19003 return

50000 rem *setup sound*
50005 f=54272: f1=f: f2=f+7: f3=f+14: fv=f+24: ft=38400: f4=f+4: f5=f+11:
      f6=f+18
50010 i=8098: r=61176/64814: for k=59 to 0 step -1: nt%(k)=int (i): i=i*r: next
50015 for q= 0to 24: poke f+q,0: next: return

50100 rem *chime i times*
50105 poke f1+1,11: poke f1,0: poke f+5,43: poke f+6,0: poke f3+1,5: poke f3,0:
      poke fv,15
50110 for q=1 to i: poke f4,20: poke f4,21: for qq=1 to 10: poke
      fz,rnd(1)*8+200: next
50112 for qq=1 to 600
50120 next: next :poke f4,20: gosub 19000: poke fv,0: poke fz,200: return

50200 rem *titlepage music*
50205 poke f+5,144: poke f+6,217: poke f+12,251: poke f+13,27: poke f+19,251:
      poke f+20,27
50210 poke f3,70: poke f3+1,6: poke f5,0
50215 poke fv,8: for j=1 to 500: next: poke f6,129: ti$="000000"
50225 poke fz,rnd(1)*4+200: if ti<480 then 50225
50230 poke f2,251: poke f2+1,9: poke f5,129: ti$="000000"
50240 poke fz,rnd(1)*4+200: if ti<540 then 50240
50245 poke f6,0:poke f+19,16: poke f+20,215: f(0)=17: f(1)=129: f(2)=17: poke
      fv,15
50247 poke sz,200: poke f+5,16: poke f+6,215
50250 ff=ft: gosub 50500
50255 i=5: gosub 50100
50295 q=aq+1: poke f4,0: poke f5,0: poke f6,0: for q=0 to 24: poke f+q,0: next:
      return
```

```
50500 rem *music sequencer*
50502 k=f1+1
50505 ti$="000000": fs=peek(ff)*3: if fs=0 then return
50515 ff=ff+1: for i=0 to 2: q=peek(ff): if q=0 then 50520
50516 if q=255 then poke f4+i*7,0: goto 50520
50517 poke f1+i*7,nt%(q) and th: poke k+i*7,nt%(q)/tf: poke f4+i*7,f(i)
50520 ff=ff+1: next i
50535 if ti<fs then 50535
50540 goto 50505

50600 rem *throne music*
50605 for fq=0 to 2: poke f+5+fq*7,18: poke f+6+fq*7,244: f(fq)=17: next:
ff=38545
50610 poke fv,10: gosub 50500: goto 50750
50750 rem *turn sound off*
50755 for fq=0 to 24: poke f+fq,0: next :return
```

The engine has tree routines for sound:

1. 50100: perform a gong like sound: it is not temporized: the duration of the sound is given by the duration of the execution of each Basic instructions
2. 50600: a music performed by using table of notes
3. 50200: some long sounds, follow by a music based onto notes in table, follow with some gong sound of 50100

The music based onto table is performed by the 50500 routine: a music sequencer in Basic that use TI variable. Note that the tables are not present in the Basic code, however we add them in the machine code porting.

If you look carefully into the Basic code, you probably are thinking that the sequencer that use TI variable can be converted into IRQ routine, like I did into Testcard rip, but the other are very difficult to convert into IRQ, as they are not timing based.

So what to do?

RSID rip

Fortunately, today we have another format that can help into this: the RSID format.

RSID (Real SID) are PSID file that can be executed into some real C64 environment (e.g. real C64 or sidplay emulator), as it use the real features of the machine, not the compatibly one introduced by old PSID format.

This means that you can listen to sample base music as in the real C64, as no digi emulation are done by the sidplayer. The are other features but the most important that we will use is that we can execute a flow of instructions without using IRQ routine at all.

So the main idea of the RSID rip is to convert each BASIC instructions to a sequence of machine code instructions that gives the same result: we can so convert the part like routine 50100 without using IRQ!

Let me give an example:

This simple Basic instruction

```
POKE 54272,10
```

will be converted into something like:

```
JSR  XXXDelayCicle
LDA  #10
STA  $D400
```

where xxxDelayCicle is a routine that has to waste the exact number of cycles that the BASIC instruction will need for being executed into the real machine.

Well, there is a big work to do as there are lot of instructions to convert and many cycles to calculated. What to do?

There are two way to choose:

1. Try to measure the average cycles in the best way we can and so write the machine code: listen then to the result and by try and error, modify the code until we can reach a good conversion. This is to considerate a temporary work that can give the opportunity to a sid fans to listen to his preferred Basic music into his sidplayer, even if it sounds quite different compared with the original. I say temporary as I hope that one day the sidplayers will manage Basic and Kernal roms, so there will be no reason for using this rip anymore.
2. Modify the code of a C64 emulator for giving the exact number of cycles that there are from two consecutive write operations into SID registers into the Basic program and then write a machine code that use this accurate cycles: we can have a definitive RSID rip with this, but a cost of lot of work.

I have used the first way (no, don' texpect I use the way 2: I should preferred going into rip of Marble Madness instead!) trying to calculate the cycles using this kind of measures:

```
10 FOR i=1 to 256
20 POKE 54272,10
30 POKE 54272,10
...
300 POKE 54272,10
310 NEXT i
```

I have run this program in a C64 emulator and measure its executing time and then calculate how many cycles a single POKE instruction will take in average.

As you can see in the code, I measure different kind of instructions and so many type of waste cycle routines are made. I think that the inserted comments are all you need for understand the rip.

RSID code

```
; A machine code port of the BASIC program Telengard
; Some measure was made to calculate the cycles to waste,
; however some adjustement were done by heard the result with
; the original. If one day a sid emulator will manage
; BASIC and KERNAL rom, this is a tune to re-rip.
; The timing is not so perfect: try to modify this source
; if you are able to fix it.

  processor 6502
  org 1923

 .byte "RSID"
 .word $0200         ; version 2
 .word $7C00         ; data offset
 .word $0000         ; load address in cbm format
 .byte $0108
 .byte $0000
 .byte $0000
 .byte $0000
 .word $0300         ; 3 song
 .word $0100         ; default song 2
 .word $0000
 .word $0000
```

```
 .byte "Telengard",0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
 .byte "<?>",0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
 .byte "1983 Avalon Hill",0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
 .word $0000
 .word $0000
 .word $0000

 .byte $01
 .byte $08

; From time measurement one poke of the form f+x,y
; takes about 0,0053sec -> 5200 cycles

; for qq=1 to 10: poke fz,rnd(1)*8+200: next
; takes about 0,1402sec -> 138000 cycles

; for qq=1 to 600: next
; takes about 0,682sec -> 672000 cycles

; for i=1 to 500: next
; takes about 0,590sec ->580000 cycles

; 19000 routine:
; takes about 1,572sec -> 1548000 cycles

; if ti<1 then goto
; takes about 0.22/12sec -> 216000/12=18000 cycles

; 50225 instr:
; takes about 8 sec -> 7882000

; 50240 instr:
; takes about 9 sec ->

        cmp  #$00
        bne  nn1

        jmp  titlepage

nn1:
        cmp  #$01
        bne  nn2
        jmp  throme

nn2:
        lda  #1
        sta  $8E
        jmp  chime



; 19000 ti$="000000"
; 19002 if ti<90then 19002
; 19003 return
r19000:
        jsr  C1548000
        rts

chime:
; 50100 rem *chime i times*
; 50105 poke f1+1,11:
        lda  #11
        sta  $D401
        jsr  C2600

;       poke f1,0:
        lda  #0
        sta  $D400
        jsr  C2600

;       poke f+5,43:
        lda  #43
        sta  $D405
        jsr  C2600

;       poke f+6,0:
        lda  #0
        sta  $D406
        jsr  C2600

;       poke f3+1,5:
        lda  #5
        sta  $D40F
        jsr  C2600

;       poke f3,0:
        lda  #0
        sta  $D40E
        jsr  C2600
```

```
;       poke fv,15
        lda  #15
        sta  $D418
        jsr  C2600

; 50110 for q=1 to i:
        lda  $8E
        jsr  C5200

nextq:

; poke f4,20:
        lda  #20
        sta  $D404
        jsr  C5200
        jsr  C5200  ; add some more delay
        jsr  C5200  ; add some more delay
        jsr  C5200  ; add some more delay
        jsr  C5200  ; add some more delay
        jsr  C5200  ; add some more delay
        jsr  C5200  ; add some more delay

; poke f4,21:
        lda  #21
        sta  $D404
        jsr  C5200

; for qq=1 to 10: poke fz,rnd(1)*8+200: next
        jsr  C138000

; 50112 for qq=1 to 600
; 50120 next:
        jsr  C672000

; next:
        jsr  C5200
        dec  $8E
        lda  $8E
        bne  nextq

; poke f4,20:
        lda  #20
        sta  $D404
        jsr  C2600

; gosub 19000:
        jsr  r19000

; poke fv,0:
        lda  #00
        sta  $D418
        jsr  C2600

; poke fz,200:
        jsr  C5200

; return
        rts


titlepage:
;50200 rem *titlepage music*
;50205 poke f+5,144: poke f+6,217: poke f+12,251: poke f+13,27: poke f+19,251: poke f+20,27
        lda  #144
        sta  $D405
        jsr  C2600

        lda  #217
        sta  $D406
        jsr  C2600

        lda  #251
        sta  $D40C
        jsr  C2600

        lda  #27
        sta  $D40D
        jsr  C2600

        lda  #251
        sta  $D413
        jsr  C2600

        lda  #27
        sta  $D414
        jsr  C2600

;50210 poke f3,70: poke f3+1,6: poke f5,0
        lda  #70
        sta  $D40E
```

```
        jsr  C2600

        lda  #6
        sta  $D40F
        jsr  C2600

        lda  #0
        sta  $D40B
        jsr  C2600


;50215 poke fv,8: for j=1 to 500: next: poke f6,129: ti$="000000"
        lda  #8
        sta  $D418
        jsr  C2600

        jsr  C580000

        lda  #129
        sta  $D412
        jsr  C5200

;50225 poke fz,rnd(1)*4+200: if ti<480 then 50225

        jsr  C1548000
        jsr  C1548000
        jsr  C1548000
        jsr  C1548000
        jsr  C1548000


;50230 poke f2,251: poke f2+1,9: poke f5,129: ti$="000000"
        lda  #251
        sta  $D407
        jsr  C2600

        lda  #9
        sta  $D408
        jsr  C2600

        lda  #129
        sta  $D40b
        jsr  C5200

;50240 poke fz,rnd(1)*4+200: if ti<540 then 50240
        jsr  C1548000
        jsr  C1548000
        jsr  C1548000
        jsr  C1548000
        jsr  C1548000
        jsr  C580000
        ;jsr  C580000


;50245 poke f6,0:poke f+19,16: poke f+20,215: f(0)=17: f(1)=129: f(2)=17: poke fv,15
        lda  #0
        sta  $D412
        jsr  C2600

        lda  #16
        sta  $D413
        jsr  C2600

        lda  #215
        sta  $D414
        jsr  C2600

        lda  #17
        sta  val
        jsr  C2600

        lda  #129
        sta  val+7
        jsr  C2600

        lda  #17
        sta  val+14
        jsr  C2600

        lda  #15
        sta  $D418
        jsr  C2600

;50247 poke sz,200: poke f+5,16: poke f+6,215
        jsr  C2600

        lda  #16
        sta  $D405
        jsr  C2600
```

```
        lda  #215
        sta  $D406
        jsr  C2600

;50250 ff=ft: gosub 50500
        lda  #<tuneA
        sta  $8B
        lda  #>tuneA
        sta  $8C
        jsr  C5200

        jsr  sequencer


;50255 i=5: gosub 50100
        lda  #5
        sta  $8E
        jsr  C2600
        jsr  chime

;50295 q=aq+1: poke f4,0: poke f5,0: poke f6,0: for q=0 to 24: poke f+q,0: next: return
        lda  #0
        sta  $D404
        jsr  C5200

        lda  #0
        sta  $D40b
        jsr  C2600

        lda  #0
        sta  $D412
        jsr  C2600

        ldx  #0
loop77:
        lda  #0
        sta  $D400,x
        jsr  C5200
        inx
        txa
        cmp  #25
        bne  loop77
        rts

        rts

sequencer:
; 50500 rem *music sequencer*
; 50502 k=f1+1
        ldy  #$00
        jsr  C2600

seq_next:
; 50505 ti$="000000": fs=peek(ff)*3: if fs=0 then return
        jsr  C5200
        lda  ($8B),y
        sta  $8D
        clc
        adc  $8D
        adc  $8D
        sta  $8D
        cmp  #00
        bne  cont22
        rts

cont22:

; 50515 ff=ff+1: for i=0 to 2: q=peek(ff): if q=0 then 50520
        iny
        jsr  C2600

        ldx  #$00
        jsr  C2600

nextSeq:
        jsr  C2600
        lda  ($8B),y
        beq  nextVoice


; 50516 if q=255 then poke f4+i*7,0: goto 50520
        jsr  C2600
        cmp  #255
        bne  continue

        lda  #00
        sta  $D404,x
        jsr  C5200
        jmp  nextVoice
```

```
continue:
; 50517 poke f1+i*7,nt%(q) and th: poke k+i*7,nt%(q)/tf: poke f4+i*7,f(i)
        sta  $8E
        tya
        pha
        ldy  $8E

        lda  low,y
        sta  $D400,x
        jsr  C5200

        lda  hi,y
        sta  $D401,x
        jsr  C5200

        lda  val,x
        sta  $D404,x
        jsr  C5200
        pla
        tay

nextVoice:
; 50520 ff=ff+1: next i
        jsr  C2600
        iny
        txa
        clc
        adc  #$07
        tax
        cmp  #$15
        bne  nextSeq

; 50535 if ti<fs then 50535
        lda  $8D
        sbc  #$03   ; some cycles are already wastle
loop2:
        jsr  C18000
        sec
        sbc  #$01
        bne  loop2


; 50540 goto 50505
        jsr  C2600
        jmp  seq_next

throme:
; 50600 rem *throne music*
; 50605 for fq=0 to 2: poke f+5+fq*7,18: poke f+6+fq*7,244: f(fq)=17: next: ff=38545
        lda  #18
        sta  $D405
        jsr  C5200

        lda  #244
        sta  $D406
        jsr  C5200

        lda  #17
        sta  val
        jsr  C5200

        lda  #18
        sta  $D40C
        jsr  C5200

        lda  #244
        sta  $D40D
        jsr  C5200

        lda  #17
        sta  val+7
        jsr  C5200

        lda  #18
        sta  $D413
        jsr  C5200

        lda  #244
        sta  $D414
        jsr  C5200

        lda  #17
        sta  val+14
        jsr  C5200

        lda  #<tuneB
        sta  $8B
        lda  #>tuneB
        sta  $8C
        jsr  C5200
```

```
; 50610 poke fv,10:
        lda  #10
        sta  $D418
        jsr  C5200

; gosub 50500:
        jsr  sequencer

; goto 50750
; 50750 rem *turn sound off*
; 50755 for fq=0 to 24: poke f+fq,0: next :return

        ldx  #0
loop1:
        lda  #0
        sta  $D400,x
        jsr  C5200
        inx
        txa
        cmp  #25
        bne  loop1
        rts

; waste 5200 about cycles
C5200:
        pha
        txa
        pha
        tya
        pha
        ldx  #$00             ; 2
rep2:
        lda  ($33,x)          ; 6
        lda  ($34,x)          ; 6
        lda  $3334            ; 4
        dex                   ; 2
        bne  rep2             ; 2
        pla
        tay
        pla
        tax
        pla
        rts                   ; 6

 ; waste 2600 about cycles
C2600:
        pha
        txa
        pha
        tya
        pha
        ldx  #129             ; 2
rep20:
        lda  ($33,x)          ; 6
        lda  ($34,x)          ; 6
        lda  $3334            ; 4
        dex                   ; 2
        bne  rep20            ; 2
        pla
        tay
        pla
        tax
        pla
        rts                   ; 6

; waste 138000 about cycles
; add some nops as too fast
C138000:
        pha
        txa
        pha
        tya
        pha
        ldx  #$00             ; 2
rep3:
        ldy  #134             ; 2
rep1:
        nop ;
        nop ;
        nop ;
        nop ;
        dey                   ; 2
        bne  rep1             ; 2
        dex                   ; 2
        bne  rep3             ; 2
        pla
        tay
        pla
        tax
        pla
```

```
        rts                     ; 6

; waste 18000 about cycles
; fix: 18000 are too many: from #17 -> #12
C18000:
        pha
        txa
        pha
        tya
        pha
        ldx  #$00               ; 2
rep33:
        ldy  #12                ; 2
rep11:
        dey                     ; 2
        bne  rep11              ; 2
        dex                     ; 2
        bne  rep33              ; 2
        pla
        tay
        pla
        tax
        pla
        rts                     ; 6

; waste 672000 about cycles
C672000:
        pha
        txa
        pha
        tya
        pha
        ldx  #$00               ; 2
rep4:
        ldy  #219               ; 2
rep5:
        lda  ($33,x)            ; 6
        nop                     ; 2
        dey                     ; 2
        bne  rep5               ; 2
        lda  $33                ; 3
        dex                     ; 2
        bne  rep4               ; 2
        pla
        tay
        pla
        tax
        pla
        rts                     ; 6

; waste 580000 about cycles
C580000:
        pha
        txa
        pha
        tya
        pha
        ldx  #$00               ; 2
rep44:
        ldy  #189               ; 2
rep55:
        lda  ($33,x)            ; 6
        nop                     ; 2
        dey                     ; 2
        bne  rep55              ; 2
        lda  $33                ; 3
        dex                     ; 2
        bne  rep44              ; 2
        pla
        tay
        pla
        tax
        pla
        rts                     ; 6

; waste 1548000 about cycles
C1548000:
        pha
        txa
        pha
        tya
        pha
        ldx  #$00               ; 2
rep6:
        ldy  #216               ; 2
rep7:
        lda  ($33,x)            ; 6
        lda  ($33,x)            ; 6
        lda  ($33,x)            ; 6
        lda  ($33,x)            ; 6
```

```
        dey                     ; 2
        bne  rep7               ; 2
        lda  ($33,x)            ; 6
        lda  ($33,x)            ; 6
        lda  ($33,x)            ; 6
        nop                     ; 2
        dex                     ; 2
        bne  rep6               ; 2
        pla
        tay
        pla
        tax
        pla
        rts                     ; 6

low:
  .byte  12,  27,  44,  62,  81, 101, 123, 145
  .byte 194, 221, 250,  24,  55,  89, 125, 163
  .byte 203, 246,  35,  83, 133, 187, 244,  48
  .byte 112, 179, 251,  71, 151, 236,  70, 166
  .byte  11, 118, 232,  96, 224, 103, 246, 142
  .byte  46, 217, 141,  76,  23, 237, 208, 193
  .byte 192, 206, 237,  28,  93, 178,  27, 153
  .byte  46, 219, 162

hi:
  .byte   1,   1,   1,   1,   1,   1,   1,   1
  .byte   1,   1,   1,   2,   2,   2,   2,   2
  .byte   2,   2,   3,   3,   3,   3,   3,   4
  .byte   4,   4,   4,   5,   5,   5,   6,   6
  .byte   7,   7,   7,   8,   8,   9,   9,  10
  .byte  11,  11,  12,  13,  14,  14,  15,  16
  .byte  17,  18,  19,  21,  22,  23,  25,  26
  .byte  28,  29,  31

tuneA:
  .byte $20, $24, $00, $1f,  $20, $27, $00, $24
  .byte $10, $26, $00, $1a,  $08, $24, $00, $00
  .byte $08, $22, $00, $00,  $10, $24, $00, $00
  .byte $10, $1f, $00, $00,  $20, $24, $00, $1f
  .byte $20, $27, $00, $24,  $08, $29, $00, $26
  .byte $08, $27, $00, $24,  $08, $26, $00, $22
  .byte $08, $24, $00, $21,  $20, $26, $00, $22
  .byte $04, $ff, $00, $ff,  $10, $24, $00, $30
  .byte $10, $00, $00, $2b,  $10, $27, $00, $2b
  .byte $08, $00, $00, $2e,  $08, $00, $00, $30
  .byte $10, $26, $00, $32,  $08, $24, $00, $2b
  .byte $08, $22, $00, $00,  $10, $24, $00, $2b
  .byte $10, $1f, $00, $32,  $10, $24, $00, $30
  .byte $10, $00, $00, $2b,  $10, $27, $00, $2b
  .byte $08, $00, $00, $2e,  $08, $00, $00, $30
  .byte $08, $29, $00, $2e,  $08, $27, $00, $00
  .byte $08, $26, $00, $00,  $08, $24, $00, $00
  .byte $20, $26, $1b, $00,  $30, $ff, $00, $ff
  .byte $00

tuneB:
  .byte $04, $2f, $2b, $23,  $04, $31, $2d, $25
  .byte $04, $32, $2f, $26,  $04, $34, $31, $28
  .byte $04, $36, $32, $2a,  $04, $32, $2f, $26
  .byte $0a, $36, $32, $2a,  $04, $35, $30, $29
  .byte $04, $31, $2d, $25,  $0a, $35, $30, $29
  .byte $04, $34, $31, $28,  $04, $30, $2c, $24
  .byte $0a, $34, $31, $28,  $04, $2f, $2b, $23
  .byte $04, $31, $2d, $25,  $04, $32, $2f, $26
  .byte $04, $34, $31, $28,  $04, $36, $32, $2a
  .byte $04, $32, $2f, $26,  $04, $36, $32, $2a
  .byte $04, $3b, $37, $2f,  $04, $39, $36, $2d
  .byte $04, $36, $32, $2a,  $04, $32, $2f, $26
  .byte $04, $34, $31, $28,  $0a, $36, $32, $2a
  .byte $00

val:
  ; some bytes are used here
  .byte $00
```

Conclusion

RSID rip can be a valid solution for BASIC music. I was able even to convert BASIC music that use RND random number, using some random tables of lookup values.

But maybe this can be a good material for other articles...

When I had choose a Martin tune for obtaining his music engine, Arkanoid was the most perfect candidate as the tune contains the new sample music system in it. We all known the story about the first tune released and produced that contains sample, but it is very interesting to see in Galway implementation a self modified code to mask the use of volume register for sample, and the use of 6502 JMP/JSR opcode as music pattern (we will see this later): I think that understand the sample routine at that time using the available tools was very difficult due to the use of all this kind of masked code.

Another point that appears me at the first approach with the Galway code is that it seems not so good coded: each voice has its code that are perfectly equal each other, so it may be coded using some indexed addressing mode for reducing size and having only one peace of code to maintain, instead of 3 equals parts. However, as all effects are coded into tables, this made the Galway engine very powerful: maybe having left each voice separate could be a Martin choice for not complicating more the engine.

Songs

As usual the engines is based onto songs:

```
tune1:
  .byte <tune1_voice1, >tune1_voice1
  .byte <tune1_voice2, >tune1_voice2
  .byte <tune1_voice3, >tune1_voice3
  .byte $09
```

Each songs have their tracks address for each voice, and a byte ($09 in the examples) that is the minim duration used for a note. If we increase this value, than one note takes more longer his duration (this is true only for the kind of notes that use table duration as we will see later).

Tracks/Patterns

In Martin engine there are not the division in tracks and patterns, but they are to be considerate together, as there are other methods to take advantage of the use of patterns: subroutine call.

Well, I have see this method in old games, where the music flow is programmed by pseudo instructions to be performed, but I think that almost today music engine did not use this kind of approach any more as it very powerful if it is compiled by hand, but too complicated by an editor program (even if this not means that it is impossible to implement, as I have already seen very complicated editor around).

However the first point that pop up from source looking is that instructions are not present for all voices, but some instructions are available only for a particular voice: this seems me that Martin had added some instruction when needed: maybe other version of the engine has other implemented instructions. Otherwise, he removed the code not used by one voice for reducing memory size.

Instructions/Notes

Well, programming a track is like creating a music program using a high level language: I have try to give a name to each instruction that can be useful, maybe Martin should had used directly the hex value instead.

You can see in the instruction table that there are instructions for controlling the program flow (subroutine call, jump, for/next cycle) and instruction for controlling the music parameters (like instruments table, ...)

HEX	*MNEMONIC*	*DESCRIPTION*	*VOICES*
00..5F nn	Note / dur_tab	Play note with duration nn given by a table of durations (5F=rest)	1,2,3
60..BF nn	Note/dur	Play note xx-60 with duration nn (BF=rest)	1,2,3
C0	RTS	Return to the caller instruction. Also used for terminating a track.	1,2,3
C2 lo hi	JSR	Execute the subroutine at the given address (low/high)	1,2,3
C4 lo hi	JMP	Jump to the new address location (low/high)	1,2,3
C6 ht lo hi	JSRT	Execute the subroutine at the given address (low/high), but perform the task using ht halftones added	1,3
CA id va	SET	Set a value (va) in the instrument table at given index (id): all offset of the table can be used	1,2,3
CC nn	FOR	Repeat nn times (For like instruction)	1,2,3
CE	NEXT	Point on where repeat (Next like instruction)	1,2,3
D2 nn lo hi	SETNI	Set the fisrt nn item of the instrument table form the given address low/high	1,2,3
D4 lo hi	INSTR	set 5 instrument parameters from the given address location (low/high) – control/ADSR	1,2,3
D6 nn va	SETCI	Set the current instrument value (va) at the given index (nn)	1,2,3
D8 lo hi	EXCT	Execute a peace of machine language code located at low/high address	3
DC id v1 v2	SET2I	Set 2 values (v1, v2) of instruction table at the given index (id): all offset are allowed	2,3
DE id v1 v2	SET2CI	Set 2 values (v1, v2) of the current instrument table at the given address index (id)	1,2,3
E0	LF3	Set low filter on voice 3 with max resonance	3
E2 lo hi	FILTA	Set the filter table with the values at address low/high	3
F0 lo hi	SETFI	Set instrument frequency effect using value at address (low/high): the 13 values in the table are copied onto instrument table (offset 00..0D)	1,2,3

Note and Duration

As we can see from the instructions table, a note is specify by a number from 00h to 5Fh, or from 60h to BFh. The difference is that the followed byte duration is used in two different ways:

- 00-5F: the duration is taken by a custom built table
- 60-BF: the duration is the specified in the byte

So, for example the note 12h is equivalent to note 72h.

The custom built table is based onto the minim byte duration that are insert into the tunes tracks declaration.

Note that the 5Fh and BFh are rest notes.

We can so see that the Martin engine used something like 6 octaves for sound, instead of full 8 octaves the sid can manage: so the max notes to use is 5Eh (BEh).

Instrument Table

In the Martin engine an instrument is completely described by a table of values. Selecting an instrument is performed by filling the instrument's 29 bytes table, by using some of the given SETNI, INSTR instructions and by changing some values with other minor variants: SET and SET2I. A second table contains the actual instrument 34 bytes used for making the timbre of the instrument. All this values can be changed by using the SETCI and SET2CI.

But now take a look at the instrument table:

Pos.	*Description*
00h	Freq. low to add in each cycle of phase 1
01h	Freq. high to add in each cycle of phase 1
02h	Freq. low to add in each cycle of phase 2
03h	Freq. high to add in each cycle of phase 2
04h	Freq. low to add in each cycle of phase 3
05h	Freq. high to add in each cycle of phase 3
06h	Freq. low to add in each cycle of phase 4
07h	Freq. high to add in each cycle of phase 4
08h	Freq.: number of cycles of phase 1
09h	Freq.: number of cycles of phase 2
0Ah	Freq.: number of cycles of phase 3
0Bh	Freq.: number of cycles of phase 4
0Ch	Freq.: number of initial cycles of delay before phase 1

Pos.	*Description*
0Dh	Freq.: effect flag bits: xyyy yyyyz x=1 -> reload the freq. cycles with freq. value of instrument table again z=1 -> reload the freq. cycles but use the actual frequency value y=1 -> continue with actual freq., no more cycle
0Eh	Wave (pulsation amplitude): number of cycles of phase 1
0Fh	Wave (pulsation amplitude): number of cycles of phase 2
10h	Wave (pulsation amplitude): number of initial cycles of delay before phase 1
11h	Wave (pulsation amplitude): effect flag bits: xyyy yyyyz x=1 -> reload the wave cycles with wave value of instrument table again z=1 -> reload the wave cycles but use the actual wave value y=1 -> continue with actual wave, no more cycle
12h	Wave (pulsation amplitude) low to add in each cycle of phase 1
13h	Wave (pulsation amplitude) high to add in each cycle of phase 1
14h	Wave (pulsation amplitude) low to add in each cycle of phase 2
15h	Wave (pulsation amplitude) high to add in each cycle of phase 2
16h	Wave (pulsation amplitude) low value
17h	Wave (pulsation amplitude) high value
18h	Control register of the voice
19h	Attack/Decay value
1Ah	Sustain/Release value
1Bh	Duration before apply release ($FF means not apply release) [?]
1Ch	Duration before an hard restart (after the end of event of previous 1Bh register) 0 means no effect to apply [?]

And now the current instrument table:

Pos.	*Description*
00h-01h	Freq. Low/high to add in each cycle of phase 1
02h-03h	Freq. Low/high to add in each cycle of phase 2
04h-05h	Freq. Low/high to add in each cycle of phase 3
06h-07h	Freq. Low/high to add in each cycle of phase 4
08h	Freq.: number of cycles (to reload) of phase 1
09h	Freq.: number of cycles (to reload) of phase 2
0Ah	Freq.: number of cycles (to reload) of phase 3

Pos.	*Description*
0Bh	Freq.: number of cycles (to reload) of phase 4
0Ch	Freq.: number of initial cycles of delay before phase 1
0Dh	Freq. effect flag
0Eh	Wave (pulsation amplitude): number of cycles (to reload) of phase 1
0Fh	Wave (pulsation amplitude): number of cycles (to reload) of phase 2
10h	Wave (pulsation amplitude): number of initial cycles of delay before phase 1
11h	Wave (pulsation amplitude): effect flag
12h-13h	Wave (pulsation amplitude) low/high to add in each cycle of phase 1
14h-15h	Wave (pulsation amplitude) low/high to add in each cycle of phase 2
16h-17h	Wave (pulsation amplitude) low/high
18h-19h	Freq. low/high
1Ah	Control register of voice
1Bh	Duration before apply release ($FF means not apply release) [?]
1Ch	Duration before an hard restart (after the end of event of previous 1Bh register) 0 means no effect to apply [?]
1Dh	Freq.: number of cycles of phase 1
1Eh	Freq.: number of cycles of phase 2
1Fh	Freq.: number of cycles of phase 3
20h	Freq.: number of cycles of phase 4
21h	Wave (pulsation amplitude): number of cycles of phase 1
22h	Wave (pulsation amplitude): number of cycles of phase 2

The main part of the instruments table is the AD/SR (19h-1Ah offsets), control (18h offset), and wave pitch for rectangular waveform (16h-17h offsets) values (as usual). All the other values are for pitch control as we see in the followed part.

Frequencies Timbre

Effects like vibrato and portamento can be produced by using the frequencies values of the instrument table.

There are 4 phases used by the timbre routine that manage frequencies: in each phase the actual value of the frequency that is produced by the sid is added to the one specified by the low/high values in the table (00h-01h offsets for phase 1) for a number of cycles specified by a given byte (08h offset for phase 1). When the cycles are over, a new phase will start.

The important point is that the added value are in two complement logic, so you can produce even a subtraction.

A special value (0Ch offset) is used for delaying the initial call to the first phase, for the specified number of cycles.

Finally the effect flag (0Dh) control what are to be executed after the ending of the for phase (it must be different from 0 for having the frequency timbre engaged):

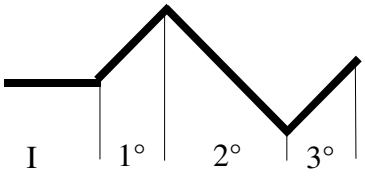
- If the bit 7 is 1, the frequency to use is taken from the value of current instrument table, then the cycles and adding value are taken again from the stored (into instrument table) and so a new sequence of phases restart.
- If the bit 1 is 1, the frequency used is the last one, but no other phases will begin.
- If all other bits are not zero: the frequency to use are the actual that the Sid is generating, then the phases restart with the other values as in the first case.

Let me show some examples:

- Vibrato:

```
.byte $14, $00                  ; freq low/high add 1
.byte $EC, $FF                  ; freq low/high add 2
.byte $14, $00                  ; freq low/high add 3
.byte $00, $00                  ; freq low/high add 4
.byte $03, $06                  ; freq cycle 1/2
.byte $03, $00                  ; freq cycle 3/4
.byte $1E, $05                  ; freq delay initial/freq effect flag
```

In this case, after an initial delay of 1Eh cycles, there are 3 cycles where 14h is added to currently frequencies, then for 6 cycles -14h is added (and so subtracted), finally for 3 cycles 14h is added. Now, the cycles can restart (05h) continuing with the actual frequencies.



This figure shows why cycle 2° is double longer (this is necessary for having the correct up/down sequence).

- Portamento:

```
.byte $01, $00                  ; freq low/high add 1
.byte $00, $00                  ; freq low/high add 2
.byte $00, $00                  ; freq low/high add 3
.byte $00, $00                  ; freq low/high add 4
.byte $35, $00                  ; freq cycle 1/2
.byte $00, $00                  ; freq cycle 3/4
.byte $00, $08                  ; freq delay initial/freq effect flag
```

This make a long up portamento with little frequency increment.

- Other effects:

Just changing the phases, you can reach very complex frequencies tasks: look at the source for more examples of real Martin use of the phases.

Rectangular Wave Timbre

If we use rectangular waveform, we can control the real time pitch (pulse amplitude) generation by using the same methods like frequencies: in this case, however, we only have two phases instead of 4. It useful to see how now the wave flag is intended for:

- If the bit 7 is 1, the pitch to use is taken from the value of current instrument table, then the cycles and adding value are taken again from the stored (into instrument table) and so a new sequence of phases restart.
- If the bit 1 is 1, the pitch used is the last one, but no other phases will begin.
- If all other bits are not zero: the pitch to use are the actual that the Sid is generating, then the

phases restart with the other values as in the first case.

For examples, a classical up/down pulse amplitude effect is achieved by:

```
.byte $32, $32                  ; wave cycle 1/2
.byte $14, $05                  ; wave delay initial/wave effect flag
.byte $0A, $00                  ; wave low/high add 1
.byte $F6, $FF                  ; wave low/high add 2
```

Filter Table

The Martin engine have a very complex filter manipulation, that are achieved using this filter table parameters:

Pos.	*Description*
00h-01h	Filter low/high value to add in each cycle of phase 1
02h-03h	Filter low/high value to add in each cycle of phase 2
04h-05h	Filter low/high value to add in each cycle of phase 3
06h-07h	Filter low/high value to add in each cycle of phase 4
08h	Filter: number of cycles of phase 1
09h	Filter: number of cycles of phase 2
0Ah	Filter: number of cycles of phase 3
0Bh	Filter: number of cycles of phase 4
0Ch	Filter: number of initial cycles of delay before phase 1
0Dh	Filter: effect flag bits: xyyy yyyyz x=1 -> reload the filter cycles with filter value of instrument table again z=1 -> reload the filter cycles but use the actual filter value y=1 -> continue with actual filter value, no more cycle
0Eh-0Fh	Filter low(8)/high(3) value

The effect you can generate is analogue to what we can achieve with frequencies table as even here there are four phases.

Here, for examples, there is a table used into the music of Arkanoid:

```
.byte $4D           ; 0h: add filter low value 1
.byte $01           ; 1h: add filter high value 1
.byte $D3           ; 2h: add filter low value 2
.byte $FF           ; 3h: add filter high value 2
.byte $FB           ; 4h: add filter low value 3
.byte $FF           ; 5h: add filter high value 3
.byte $FF           ; 6h: add filter low value 4
.byte $FF           ; 7h: add filter high value 4
.byte $03           ; 8h: filter cycle 1
.byte $14           ; 9h: filter cycle 2
.byte $0A           ; Ah: filter cycle 3
.byte $32           ; Bh: filter cycle 4
.byte $00           ; Ch: filter initial delay
.byte $04           ; Dh: filter effect flag
```

```
.byte $01           ; Eh: filter low value (8 bit)
.byte $00           ; Fh: filter high value (3 bit)
```

The instruction that is to be used for setting the filter table is the FILTA, but even the FL3 (that set the resonance of filter to max) and some calling to EXCT of custom code are used for better controlling the filter generation.

Other Instrument Effects

At this time we not have shown the meaning of the 1Bh and 1Ch bytes into the instruments table.

I can say that not all the meaning of this flags I was able to understand, however they could be grouped like something related to the restart (may be even an hard restart) of a note.

Looking in my note, the meaning of these control register are:

1Bh: specify a duration before a release are apply (if $FF no release are to be done)

1Ch: specify a duration before apply an hard restart (this event occurs after having executed the 1Bh effect). A $FF value means no hard restart to perform.

But a complication appears as the control byte can have even the test bit set: in this case it is tested if 1Bh table value is below current duration before apply a release (even if test bit is selected, the effective control putted into sid register is the one without the test bit).

As you can see from the source code, I did not experiment so much these table values for understand completely they meanings, so try to found by yourself the answers :(

For terminating the instrument viewing, in voice 2 there is a extra effect for frequency parameter that seems added only for Arkanoid: if bit 3 is 1, other task are performed: very important task for the timbre as it heavy used into tune 1.

Game Sound Effects

In Arkanoid lot of sound effects are used: they are generated using the same engine, but, for being executed quickly, no track/pattern command are used. Instead, a sound effect is activated by compiling an effect instrument table, and then activating the same methods (*makeTimbreVx*) used by the engine.

The sound instruments table is a perfect copy of the instrument table, but with two extra bytes at the end: the low/high value of the frequency to use. As no note/duration are used, the value of the note is taken by these two extra bytes, and the duration is taken according by the timbre effect the instrument table is executing.

The important thing is that all the 3 voices can be used for generating the sound effect (and else, this is the way used in the game).

Sample

Well, now we are arrived to the most interesting point: the sample generation.

If you are thinking that sample generation used by Martin is the “reproduction of sample” like in the common way used, you are not in the right way. If you were a ripper, maybe you know the right solution, because you should know the PSID specific extension that were introduced for managing this kind of “sample”.

The Martin approach to sample were to generate some sounds with volume variation by some procedures: each of these (6 in the Arkanoid) will reproduce a particular timbre, then by a flow instruction control, the procedures were called according to the music flow.

So, using sample in Martin' sengine is like programming the normal Sid sound voices, but using another syntax:

Hex	*Mnemonic*	*Description*
81 xx	Sample 1	Play sample type 1 for *xx* duration
82 xx	Sample 2	Play sample type 2 for *xx* duration
83 xx	Sample 3	Play sample type 3 for *xx* duration
84 xx	Sample 4	Play sample type 4 for *xx* duration
85 xx	Sample 5	Play sample type 5 for *xx* duration
86 xx	Sample 6	Play sample type 6 for *xx* duration
87 xx	Nothing	Play nothing for *xx* duration
20 lo hi	JSR	Executer a subroutine at given address
40	NEXT	Next like instruction
60	RTS	Return from subroutine instruction
49 xx	FOR	For (repeat *xx* times) like instruction
4C lo hi	JMP	Jump to the given address

As you can see in this table the hex used by JSR, RTS and JMP instruction are exactly the same opcodes as 6510 instructions: if you see the source, the flow of sample program could be interpreted by a real processor flow. This can be a coincidence, or maybe a way to make the part related to sample generation a bit harder to understand (a disassembly of this part will show code very similar to real 6510 instructions). Also, the pointers low and high that point to the pseudo instructions (used by the sample routine flow method) are located in two area very far, maybe this is for making hard to find the implementation of the routines into memory, but this can also be due to a memory restriction of the game.

However, if we see the 6 samples routines (they are located into very sparse memory area) that contain self modified code, maybe some suspicious of a way to hidden the sample generation seems to be present into Martin engine.

Here, for convenience, I show the code of one sample generation routine:

```
;================================
; Play Sample 1 routine
;================================
PSample1:
      ldy  #$05
```

```
      lda  #$94
      clc
      adc  #$40
      sta  Vol1+2                ; unmask the code

nextDelayP1:
      ldx  #$0C                  ; set repeating value

againP1:
      lda  delayTabP1-1,y

delayP1:                         ; waste some times
      sec
      sbc  #$01
      bne  delayP1

      lda  $DE                   ; increase up volume sequence
      clc
      adc  #$01
      sta  $DE
      and  #$0F
Vol1:
      sta  $DD18                 ; play sample
      dex
      bne  againP1

      dey                        ; select next delay in table
      bpl  nextDelayP1

      lda  #$DD
      sta  Vol1+2                ; mask the code again
      rts
```

Each call to the Psample1 routine will generate a sample sound witch timbre is governed by two parameters: a repeat value and a table of delays.

Essentially, for a given delay of one element in the table, no volume is changed for this delay time, after the volume is increased by some values and this will occurs until the repeating value is all counted. This has so generated an up volume sequences (that restart from low level when the max level is reached) that has a duration of one volume value given by the delay specify in the table.

The tables will then contains 5 or more delay values for giving the complete timbre of the instrument.

Now you probably should be able to understand why the Arkanoid tune will play so different into a sidplayer (with extended sid register) instead of the real machine (or today RSID rip). PSID extended register try to emulated this kind of sound generation that is quite an unusual way for common samples sound.

Source

```
; Arkanoid reverse enginnering source
; all copyright for this code remains to Martin Galway
  processor 6502
  org 1923

 .byte "RSID"
 .word $0200         ; version 2
 .word $7C00         ; data offset
 .word $0000         ; load address in cbm format
 .byte $08
 .byte $01
 .byte $0000
 .byte $0000
 .word $1400         ; 20 song
 .word $0100         ; default song 1
 .word $0000
```

```
 .word $0000
 .byte "Arkanoid",0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
 .byte "Martin Galway",0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
 .byte "1986 Imagine",0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
 .word $0000
 .word $0000
 .word $0000

 .byte $01
 .byte $08


; 31..+7   low addr into stack voice 1
; 39..+7   high addr into stack voice 1
; 40..+7   number of repeat voice 1

; A9..+7   low addr into stack voice 2
; B1..+7   high addr into stack voice 2
; B9..+7   number of repeat voice 2

; 61..+7   low addr into stack voice 3
; 69..+7   high addr into stack voice 3
; 71..+7   number of repeat voice 3

; 78..+7   low addr into stack sample
; 80..+7   high addr into stack sample
; 89..+7   number of repeat sample

; DA  low addr sample pattern index
; DB  high addr sample pattern index
; DC  duration of this sample
; DD  stack index for sample
; DE
; DF  sample generator routine index
; E0  low pattern addr. voice 1
; E1  high pattern addr. voice 1
; E2  low pattern addr. voice 2
; E3  high pattern addr. voice 2
; E4  low pattern addr. voice 3
; E5  high pattern addr. voice 3
; E6  actual note duration (length) voice 1
; E7  actual note duration (length) voice 2
; E8  actual note duration (length) voice 3
; E9  stack index for voice 1
; EA  stack index for voice 2
; EB  stack index for voice 3
; EC  halftone to add to current note voice 1
; ED  halftone to add to current note voice 2
; EE  halftone to add to current note voice 3
; EF
; F0
; F1  wave low v1
; F2  wave high v1
; F3  wave low v2
; F4  wave high v2
; F5  wave low v3
; F6  wave high v3
; F7  freq. low voice 1
; F8  freq. high voice 1
; F9  freq. low voice 2
; FA  freq. high voice 2
; FB  freq. low voice 3
; FC  freq. high voice 3
; FD  tmp track offset / low address of table
; FE  high address of table
; FF  current note to play

; pattern format:
; xx nn:
;   xx=00..5F play note xx for time from table(nn)
;   xx=60..BF play note xx-60h for time nn
; C0          RTS    instr.
; C2 lo hi    JSR    instr.
; C4 lo hi    JMP    instr.
; C6 ht lo hi JSRT   instr. (1,3)
; CA id va    SET    instr.
; CC nn       FOR    instr.
; CE          NEXT   instr.
; D2 nn lo hi SETNI  instr.
; D4 lo hi    INSTR  instr.
; D6 nn va    SETCI  instr.
; D8 lo hi    EXCT   instr. (3)
; DC id v1 v2 SET2I  instr. (2,3)
; DE id v1 v2 SET2CI instr.
; E0          LF3    instr. (3)
; E2 lo hi    FILTA  instr. (3)
; F0 lo hi    SETFI  instr.


; sample pattern format:
```

```
; 81 xx        : Sample 1
; 82 xx        : Sample 2
; 83 xx        : Sample 3
; 84 xx        : Sample 4
; 85 xx        : Sample 5
; 86 xx        : Sample 6
; 87 xx        : Nothing
; 20 lo hi     : JSR yyxx
; 40           : NEXT
; 60           : RTS
; 49 xx        : FOR
; 4C lo hi     : JMP yyxx

TEMP = $5FFF   ; 3FFF in the original code

      sta  $1FFF
      sei
      LDA  #$35
      STA  $01                  ; 6510 I/O register
      ;JSR  $1FC0
      lda  #$00
      sta  $DC0E                ; Control register A of CIA #1
      lda  #<IRQ
      sta  $FFFE                ; Masckerable Interrupt (IRQ) vector
      lda  #>IRQ
      sta  $FFFF
      jsr  initEngine
      jsr  initSample
      ldx  $1FFF
      cpx  #$02
      bcs  notSample

      lda  #$F0
      sta  $DC04                ; Timer A #1: Lo Byte
      lda  #$49
      sta  $DC05                ; Timer A #1: Hi Byte
      cpx  #$00
      bne  isTune2

      lda  #$02                 ; sample duration
      ldx  #<Sample_Tune1       ; low address
      ldy  #>Sample_Tune1       ; high address
      jsr  setSample
      jmp  nextP

isTune2:
      lda  #$02                 ; sample duration
      ldx  #<Sample_Tune2       ; low address
      ldy  #>Sample_Tune2       ; high address
      jsr  setSample
      jmp  nextP

notSample:
      lda  #$F8
      sta  $DC04                ; Timer A #1: Lo Byte
      lda  #$24
      sta  $DC05                ; Timer A #1: Hi Byte

nextP:
      ldx  $1FFF
      cpx  #$08
      bcc  normalTune
      jmp  calcAddress          ; calcolate and set address for effects

normalTune:
      ldy  offsetTracks,x
      jsr  setTracks

setInterrupt:
      lda  #$81
      sta  $DC0D                ; Interrupt control register CIA #1
      lda  #$01
      sta  $DC0E                ; Control register A of CIA #1
      cli

      lda  $1FFF                ; tune to play
      cmp  #$02                 ; first two tunes use sample
      bcc  SLoop
      rts

SLoop:
      jsr  SampleGeneration
      jmp  SLoop

offsetTracks:
  .byte $3D, $28, $05, $0C, $13, $21, $2F, $36
  .byte $36, $44, $4B, $52, $59, $60, $67, $6E

IRQ:
      pha
```

```
      tya
      pha
      txa
      pha
      lda  $1FFF                ; number of tune to play
      cmp  #$02
      bcs  skipSample
      jsr  playSample           ; play sample for the first 2 tunes

skipSample:
      jsr  makeFilterEff
      jsr  executePatternV1
      jsr  executePatternV2
      jsr  executePatternV3
      jsr  makeTimbreV1         ; voice 1
      jsr  makeTimbreV2         ; voice 2
      jsr  makeTimbreV3         ; voice 3
      lda  $DC0D                ; Interrupt control register CIA #1
      pla
      tax
      pla
      tay
      pla
      rti

  org $2000
  .byte $3A, $20                ; freq low/high add 1
  .byte $00, $00                ; freq low/high add 2
  .byte $C6, $DF                ; freq low/high add 3
  .byte $00, $00                ; freq low/high add 4
  .byte $01, $02                ; freq (to reload) cycle 1/2
  .byte $01, $01                ; freq (to reload) cycle 3/4
  .byte $01                     ; freq initial delay
  .byte $05                     ; freq effect flag

  .byte $00, $00                ; wave (to reload) cycle 1/2
  .byte $00                     ; wave initial delay
  .byte $00                     ; wave effect flag
  .byte $00, $00                ; wave low/high add 1
  .byte $00, $00                ; wave low/high add 2

  .byte $00, $00                ; wave low/high
  .byte $21                     ; control
  .byte $03, $8A                ; AD/SR
  .byte $0F, $5A
  .byte $3A, $20                ; freq. low/high

  org $201F
  .byte $C6, $DF                ; freq low/high add 1
  .byte $00, $00                ; freq low/high add 2
  .byte $3A, $20                ; freq low/high add 3
  .byte $00, $00                ; freq low/high add 4
  .byte $01, $01                ; freq (to reload) cycle 1/2
  .byte $01, $02                ; freq (to reload) cycle 3/4
  .byte $01                     ; freq initial delay
  .byte $05                     ; freq effect flag

  .byte $00, $00                ; wave (to reload) cycle 1/2
  .byte $00                     ; wave initial delay
  .byte $00                     ; wave effect flag
  .byte $00, $00                ; wave low/high add 1
  .byte $00, $00                ; wave low/high add 2

  .byte $00, $00                ; wave low/high
  .byte $21                     ; control
  .byte $03, $8A                ; AD/SR
  .byte $0F, $5A
  .byte $D8, $40                ; freq. low/high

  org $203E
  .byte $9E, $20                ; freq low/high add 1
  .byte $00, $00                ; freq low/high add 2
  .byte $62, $DF                ; freq low/high add 3
  .byte $00, $00                ; freq low/high add 4
  .byte $01, $01                ; freq (to reload) cycle 1/2
  .byte $01, $01                ; freq (to reload) cycle 3/4
  .byte $02                     ; freq initial delay
  .byte $05                     ; freq effect flag

  .byte $00, $00                ; wave (to reload) cycle 1/2
  .byte $00                     ; wave initial delay
  .byte $00                     ; wave effect flag
  .byte $00, $00                ; wave low/high add 1
  .byte $00, $00                ; wave low/high add 2
  .byte $00, $00                ; wave low/high
  .byte $21                     ; control
  .byte $03, $8A                ; AD/SR
  .byte $0F, $5A
  .byte $3C, $41                ; freq. low/high
```

```
org $205D
.byte $1F, $18                  ; freq low/high add 1
.byte $00, $00                  ; freq low/high add 2
.byte $E1, $E7                  ; freq low/high add 3
.byte $00, $00                  ; freq low/high add 4
.byte $01, $02                  ; freq (to reload) cycle 1/2
.byte $01, $01                  ; freq (to reload) cycle 3/4
.byte $01                       ; freq initial delay
.byte $05                       ; freq effect flag

.byte $00, $00                  ; wave (to reload) cycle 1/2
.byte $00                       ; wave initial delay
.byte $00                       ; wave effect flag
.byte $00, $00                  ; wave low/high add 1
.byte $00, $00                  ; wave low/high add 2

.byte $00, $00                  ; wave low/high
.byte $21                       ; control
.byte $03, $8A                  ; AD/SR
.byte $0F, $5A
.byte $3E, $30                  ; freq. low/high

org $207C
.byte $5E, $CF                  ; freq low/high add 1
.byte $00, $00                  ; freq low/high add 2
.byte $A2, $30                  ; freq low/high add 3
.byte $00, $00                  ; freq low/high add 4
.byte $01, $01                  ; freq (to reload) cycle 1/2
.byte $01, $02                  ; freq (to reload) cycle 3/4
.byte $01                       ; freq initial delay
.byte $05                       ; freq effect flag

.byte $00, $00                  ; wave (to reload) cycle 1/2
.byte $00                       ; wave initial delay
.byte $00                       ; wave effect flag
.byte $00, $00                  ; wave low/high add 1
.byte $00, $00                  ; wave low/high add 2

.byte $00, $00                  ; wave low/high
.byte $21                       ; control
.byte $03, $8A                  ; AD/SR
.byte $0F, $5A
.byte $44, $61                  ; freq. low/high

org $209B
.byte $06, $31                  ; freq low/high add 1
.byte $00, $00                  ; freq low/high add 2
.byte $FA, $CE                  ; freq low/high add 3
.byte $00, $00                  ; freq low/high add 4
.byte $01, $01                  ; freq (to reload) cycle 1/2
.byte $01, $01                  ; freq (to reload) cycle 3/4
.byte $02                       ; freq initial delay
.byte $05                       ; freq effect flag

.byte $00, $00                  ; wave (to reload) cycle 1/2
.byte $00                       ; wave initial delay
.byte $00                       ; wave effect flag
.byte $00, $00                  ; wave low/high add 1
.byte $00, $00                  ; wave low/high add 2

.byte $00, $00                  ; wave low/high
.byte $21                       ; control
.byte $03, $8A                  ; AD/SR
.byte $0F, $5A
.byte $0C, $62                  ; freq. low/high

org $20BA
.byte $04, $00                  ; freq low/high add 1
.byte $00, $00                  ; freq low/high add 2
.byte $00, $00                  ; freq low/high add 3
.byte $00, $00                  ; freq low/high add 4
.byte $FF, $00                  ; freq (to reload) cycle 1/2
.byte $00, $00                  ; freq (to reload) cycle 3/4
.byte $00                       ; freq initial delay
.byte $04                       ; freq effect flag

.byte $00, $00                  ; wave (to reload) cycle 1/2
.byte $00                       ; wave initial delay
.byte $00                       ; wave effect flag
.byte $00, $00                  ; wave low/high add 1
.byte $00, $00                  ; wave low/high add 2

.byte $00, $08                  ; wave low/high
.byte $41                       ; control
.byte $00, $F0                  ; AD/SR
.byte $32, $01
.byte $20, $03                  ; freq. low/high

org $20D9
.byte $04, $00                  ; freq low/high add 1
```

```
.byte $00, $00                  ; freq low/high add 2
.byte $00, $00                  ; freq low/high add 3
.byte $00, $00                  ; freq low/high add 4
.byte $FF, $00                  ; freq (to reload) cycle 1/2
.byte $00, $00                  ; freq (to reload) cycle 3/4
.byte $00                       ; freq initial delay
.byte $04                       ; freq effect flag

.byte $00, $00                  ; wave (to reload) cycle 1/2
.byte $00                       ; wave initial delay
.byte $00                       ; wave effect flag
.byte $00, $00                  ; wave low/high add 1
.byte $00, $00                  ; wave low/high add 2

.byte $00, $08                  ; wave low/high
.byte $41                       ; control
.byte $00, $F0                  ; AD/SR
.byte $32, $01
.byte $2A, $03                  ; freq. low/high

org $20F8
.byte $04, $00                  ; freq low/high add 1
.byte $00, $00                  ; freq low/high add 2
.byte $00, $00                  ; freq low/high add 3
.byte $00, $00                  ; freq low/high add 4
.byte $FF, $00                  ; freq (to reload) cycle 1/2
.byte $00, $00                  ; freq (to reload) cycle 3/4
.byte $00                       ; freq initial delay
.byte $04                       ; freq effect flag

.byte $00, $00                  ; wave (to reload) cycle 1/2
.byte $00                       ; wave initial delay
.byte $00                       ; wave effect flag
.byte $00, $00                  ; wave low/high add 1
.byte $00, $00                  ; wave low/high add 2

.byte $00, $08                  ; wave low/high
.byte $41                       ; control
.byte $00, $F0                  ; AD/SR
.byte $32, $01
.byte $34, $03                  ; freq. low/high

org $2117
.byte $F0, $D8                  ; freq low/high add 1
.byte $18, $FC                  ; freq low/high add 2
.byte $9C, $FF                  ; freq low/high add 3
.byte $00, $00                  ; freq low/high add 4
.byte $04, $05                  ; freq (to reload) cycle 1/2
.byte $0A, $00                  ; freq (to reload) cycle 3/4
.byte $00                       ; freq initial delay
.byte $85                       ; freq effect flag

.byte $00, $00                  ; wave (to reload) cycle 1/2
.byte $00                       ; wave initial delay
.byte $00                       ; wave effect flag
.byte $00, $00                  ; wave low/high add 1
.byte $00, $00                  ; wave low/high add 2

.byte $00, $08                  ; wave low/high
.byte $41                       ; control
.byte $09, $B9                  ; AD/SR
.byte $1E, $32
.byte $50, $C3                  ; freq. low/high

org $2136
.byte $F0, $D8                  ; freq low/high add 1
.byte $18, $FC                  ; freq low/high add 2
.byte $9C, $FF                  ; freq low/high add 3
.byte $7C, $15                  ; freq low/high add 4
.byte $04, $05                  ; freq (to reload) cycle 1/2
.byte $0A, $00                  ; freq (to reload) cycle 3/4
.byte $0A                       ; freq initial delay
.byte $87                       ; freq effect flag

.byte $00, $00                  ; wave (to reload) cycle 1/2
.byte $00                       ; wave initial delay
.byte $00                       ; wave effect flag
.byte $00, $00                  ; wave low/high add 1
.byte $00, $00                  ; wave low/high add 2

.byte $00, $08                  ; wave low/high
.byte $43                       ; control
.byte $09, $B9                  ; AD/SR
.byte $1E, $32
.byte $01, $00                  ; freq. low/high

org $2155
.byte $F0, $D8                  ; freq low/high add 1
.byte $18, $FC                  ; freq low/high add 2
.byte $9C, $FF                  ; freq low/high add 3
```

```
.byte $B8, $0B                  ; freq low/high add 4
.byte $04, $05                  ; freq (to reload) cycle 1/2
.byte $0A, $00                  ; freq (to reload) cycle 3/4
.byte $14                       ; freq initial delay
.byte $87                       ; freq effect flag

.byte $00, $00                  ; wave (to reload) cycle 1/2
.byte $00                       ; wave initial delay
.byte $00                       ; wave effect flag
.byte $00, $00                  ; wave low/high add 1
.byte $00, $00                  ; wave low/high add 2

.byte $00, $08                  ; wave low/high
.byte $41                       ; control
.byte $09, $B9                  ; AD/SR
.byte $1E, $32
.byte $01, $00                  ; freq. low/high

org $2174
.byte $74, $40                  ; freq low/high add 1
.byte $00, $00                  ; freq low/high add 2
.byte $8C, $BF                  ; freq low/high add 3
.byte $00, $00                  ; freq low/high add 4
.byte $01, $02                  ; freq (to reload) cycle 1/2
.byte $01, $01                  ; freq (to reload) cycle 3/4
.byte $01                       ; freq initial delay
.byte $05                       ; freq effect flag

.byte $00, $00                  ; wave (to reload) cycle 1/2
.byte $00                       ; wave initial delay
.byte $00                       ; wave effect flag
.byte $00, $00                  ; wave low/high add 1
.byte $00, $00                  ; wave low/high add 2

.byte $00, $00                  ; wave low/high
.byte $21                       ; control
.byte $03, $8A                  ; AD/SR
.byte $0F, $5A
.byte $74, $40                  ; freq. low/high

org $2193
.byte $8C, $BF                  ; freq low/high add 1
.byte $00, $00                  ; freq low/high add 2
.byte $74, $40                  ; freq low/high add 3
.byte $00, $00                  ; freq low/high add 4
.byte $01, $01                  ; freq (to reload) cycle 1/2
.byte $01, $02                  ; freq (to reload) cycle 3/4
.byte $01                       ; freq initial delay
.byte $05                       ; freq effect flag

.byte $00, $00                  ; wave (to reload) cycle 1/2
.byte $00                       ; wave initial delay
.byte $00                       ; wave effect flag
.byte $00, $00                  ; wave low/high add 1
.byte $00, $00                  ; wave low/high add 2

.byte $00, $00                  ; wave low/high
.byte $21                       ; control
.byte $03, $8A                  ; AD/SR
.byte $0F, $5A
.byte $D8, $40                  ; freq. low/high

org $21B2
.byte $3C, $41                  ; freq low/high add 1
.byte $00, $00                  ; freq low/high add 2
.byte $C4, $BE                  ; freq low/high add 3
.byte $00, $00                  ; freq low/high add 4
.byte $01, $01                  ; freq (to reload) cycle 1/2
.byte $01, $01                  ; freq (to reload) cycle 3/4
.byte $02                       ; freq initial delay
.byte $05                       ; freq effect flag

.byte $00, $00                  ; wave (to reload) cycle 1/2
.byte $00                       ; wave initial delay
.byte $00                       ; wave effect flag
.byte $00, $00                  ; wave low/high add 1
.byte $00, $00                  ; wave low/high add 2

.byte $00, $00                  ; wave low/high
.byte $21                       ; control
.byte $03, $8A                  ; AD/SR
.byte $0F, $5A
.byte $78, $82                  ; freq. low/high

org $21D1
.byte $E2, $04                  ; freq low/high add 1
.byte $38, $CD                  ; freq low/high add 2
.byte $00, $00                  ; freq low/high add 3
.byte $00, $00                  ; freq low/high add 4
.byte $09, $01                  ; freq (to reload) cycle 1/2
```

```
.byte $00, $00                  ; freq (to reload) cycle 3/4
.byte $00                       ; freq initial delay
.byte $05                       ; freq effect flag

.byte $00, $00                  ; wave (to reload) cycle 1/2
.byte $00                       ; wave initial delay
.byte $00                       ; wave effect flag
.byte $00, $00                  ; wave low/high add 1
.byte $00, $00                  ; wave low/high add 2

.byte $00, $00                  ; wave low/high
.byte $21                       ; control
.byte $15, $9B                  ; AD/SR
.byte $14, $46
.byte $3A, $20                  ; freq. low/high

org $21F0
.byte $36, $F7                  ; freq low/high add 1
.byte $38, $4A                  ; freq low/high add 2
.byte $00, $00                  ; freq low/high add 3
.byte $00, $00                  ; freq low/high add 4
.byte $09, $01                  ; freq (to reload) cycle 1/2
.byte $01, $00                  ; freq (to reload) cycle 3/4
.byte $00                       ; freq initial delay
.byte $05                       ; freq effect flag

.byte $00, $00                  ; wave (to reload) cycle 1/2
.byte $00                       ; wave initial delay
.byte $00                       ; wave effect flag
.byte $00, $00                  ; wave low/high add 1
.byte $00, $00                  ; wave low/high add 2

.byte $00, $00                  ; wave low/high
.byte $21                       ; control
.byte $15, $9B                  ; AD/SR
.byte $14, $46
.byte $D8, $40                  ; freq. low/high

org $220F
.byte $16, $0D                  ; freq low/high add 1
.byte $88, $96                  ; freq low/high add 2
.byte $00, $00                  ; freq low/high add 3
.byte $00, $00                  ; freq low/high add 4
.byte $07, $01                  ; freq (to reload) cycle 1/2
.byte $00, $00                  ; freq (to reload) cycle 3/4
.byte $00                       ; freq initial delay
.byte $05                       ; freq effect flag

.byte $00, $00                  ; wave (to reload) cycle 1/2
.byte $00                       ; wave initial delay
.byte $00                       ; wave effect flag
.byte $00, $00                  ; wave low/high add 1
.byte $00, $00                  ; wave low/high add 2

.byte $00, $00                  ; wave low/high
.byte $21                       ; control
.byte $15, $9B                  ; AD/SR
.byte $14, $46
.byte $3C, $41                  ; freq. low/high

org $222E
.byte $90, $E8                  ; freq low/high add 1
.byte $DD, $FF                  ; freq low/high add 2
.byte $0A, $00                  ; freq low/high add 3
.byte $00, $00                  ; freq low/high add 4
.byte $0A, $50                  ; freq (to reload) cycle 1/2
.byte $FF, $00                  ; freq (to reload) cycle 3/4
.byte $00                       ; freq initial delay
.byte $04                       ; freq effect flag

.byte $FF, $00                  ; wave (to reload) cycle 1/2
.byte $00                       ; wave initial delay
.byte $00                       ; wave effect flag
.byte $08, $00                  ; wave low/high add 1
.byte $00, $00                  ; wave low/high add 2

.byte $00, $08                  ; wave low/high
.byte $41                       ; control
.byte $19, $BB                  ; AD/SR
.byte $28, $C8
.byte $0C, $F8                  ; freq. low/high

org $224D
.byte $90, $E8                  ; freq low/high add 1
.byte $DD, $FF                  ; freq low/high add 2
.byte $0A, $00                  ; freq low/high add 3
.byte $00, $00                  ; freq low/high add 4
.byte $0A, $50                  ; freq (to reload) cycle 1/2
.byte $FF, $00                  ; freq (to reload) cycle 3/4
.byte $04                       ; freq initial delay
```

```
.byte $04                       ; freq effect flag

.byte $FF, $00                  ; wave (to reload) cycle 1/2
.byte $00                       ; wave initial delay
.byte $00                       ; wave effect flag
.byte $08, $00                  ; wave low/high add 1
.byte $00, $00                  ; wave low/high add 2

.byte $00, $08                  ; wave low/high
.byte $41                       ; control
.byte $19, $BB                  ; AD/SR
.byte $28, $C8
.byte $00, $FA                  ; freq. low/high

org $226C
.byte $90, $E8                  ; freq low/high add 1
.byte $DD, $FF                  ; freq low/high add 2
.byte $0A, $00                  ; freq low/high add 3
.byte $00, $00                  ; freq low/high add 4
.byte $0A, $50                  ; freq (to reload) cycle 1/2
.byte $FF, $00                  ; freq (to reload) cycle 3/4
.byte $08                       ; freq initial delay
.byte $04                       ; freq effect flag

.byte $FF, $00                  ; wave (to reload) cycle 1/2
.byte $00                       ; wave initial delay
.byte $00                       ; wave effect flag
.byte $08, $00                  ; wave low/high add 1
.byte $00, $00                  ; wave low/high add 2

.byte $00, $08                  ; wave low/high
.byte $41                       ; control
.byte $19, $BB                  ; AD/SR
.byte $28, $C8
.byte $F4, $FB                  ; freq. low/high

org $228B
.byte $3A, $20                  ; freq low/high add 1
.byte $00, $00                  ; freq low/high add 2
.byte $C6, $DF                  ; freq low/high add 3
.byte $00, $00                  ; freq low/high add 4
.byte $01, $02                  ; freq (to reload) cycle 1/2
.byte $01, $01                  ; freq (to reload) cycle 3/4
.byte $01                       ; freq initial delay
.byte $05                       ; freq effect flag

.byte $00, $00                  ; wave (to reload) cycle 1/2
.byte $00                       ; wave initial delay
.byte $00                       ; wave effect flag
.byte $00, $00                  ; wave low/high add 1
.byte $00, $00                  ; wave low/high add 2

.byte $00, $00                  ; wave low/high
.byte $21                       ; control
.byte $02, $A6                  ; AD/SR
.byte $05, $08
.byte $3A, $20                  ; freq. low/high

org $22AA
.byte $C6, $DF                  ; freq low/high add 1
.byte $00, $00                  ; freq low/high add 2
.byte $3A, $20                  ; freq low/high add 3
.byte $00, $00                  ; freq low/high add 4
.byte $01, $01                  ; freq (to reload) cycle 1/2
.byte $01, $02                  ; freq (to reload) cycle 3/4
.byte $01                       ; freq initial delay
.byte $05                       ; freq effect flag

.byte $00, $00                  ; wave (to reload) cycle 1/2
.byte $00                       ; wave initial delay
.byte $00                       ; wave effect flag
.byte $00, $00                  ; wave low/high add 1
.byte $00, $00                  ; wave low/high add 2

.byte $00, $00                  ; wave low/high
.byte $21                       ; control
.byte $02, $A6                  ; AD/SR
.byte $05, $08
.byte $D8, $40                  ; freq. low/high

org $22C9
.byte $9E, $20                  ; freq low/high add 1
.byte $00, $00                  ; freq low/high add 2
.byte $62, $DF                  ; freq low/high add 3
.byte $00, $00                  ; freq low/high add 4
.byte $01, $01                  ; freq (to reload) cycle 1/2
.byte $01, $01                  ; freq (to reload) cycle 3/4
.byte $02                       ; freq initial delay
.byte $05                       ; freq effect flag
```

```
.byte $00, $00                  ; wave (to reload) cycle 1/2
.byte $00                       ; wave initial delay
.byte $00                       ; wave effect flag
.byte $00, $00                  ; wave low/high add 1
.byte $00, $00                  ; wave low/high add 2

.byte $00, $00                  ; wave low/high
.byte $21                       ; control
.byte $02, $A6                  ; AD/SR
.byte $05, $08
.byte $3C, $41                  ; freq. low/high

org $22E8
.byte $E2, $04                  ; freq low/high add 1
.byte $08, $D5                  ; freq low/high add 2
.byte $00, $00                  ; freq low/high add 3
.byte $00, $00                  ; freq low/high add 4
.byte $0A, $01                  ; freq (to reload) cycle 1/2
.byte $00, $00                  ; freq (to reload) cycle 3/4
.byte $01                       ; freq initial delay
.byte $05                       ; freq effect flag

.byte $00, $00                  ; wave (to reload) cycle 1/2
.byte $00                       ; wave initial delay
.byte $00                       ; wave effect flag
.byte $00, $00                  ; wave low/high add 1
.byte $00, $00                  ; wave low/high add 2

.byte $00, $00                  ; wave low/high
.byte $21                       ; control
.byte $15, $9B                  ; AD/SR
.byte $14, $8C
.byte $3A, $20                  ; freq. low/high

org $2307
.byte $36, $F7                  ; freq low/high add 1
.byte $08, $52                  ; freq low/high add 2
.byte $00, $00                  ; freq low/high add 3
.byte $00, $00                  ; freq low/high add 4
.byte $09, $01                  ; freq (to reload) cycle 1/2
.byte $00, $00                  ; freq (to reload) cycle 3/4
.byte $02                       ; freq initial delay
.byte $05                       ; freq effect flag

.byte $00, $00                  ; wave (to reload) cycle 1/2
.byte $00                       ; wave initial delay
.byte $00                       ; wave effect flag
.byte $00, $00                  ; wave low/high add 1
.byte $00, $00                  ; wave low/high add 2

.byte $00, $00                  ; wave low/high
.byte $21                       ; control
.byte $15, $9B                  ; AD/SR
.byte $14, $8C
.byte $D8, $40                  ; freq. low/high

org $2326
.byte $16, $0D                  ; freq low/high add 1
.byte $70, $9A                  ; freq low/high add 2
.byte $00, $00                  ; freq low/high add 3
.byte $00, $00                  ; freq low/high add 4
.byte $08, $01                  ; freq (to reload) cycle 1/2
.byte $00, $00                  ; freq (to reload) cycle 3/4
.byte $03                       ; freq initial delay
.byte $05                       ; freq effect flag

.byte $00, $00                  ; wave (to reload) cycle 1/2
.byte $00                       ; wave initial delay
.byte $00                       ; wave effect flag
.byte $00, $00                  ; wave low/high add 1
.byte $00, $00                  ; wave low/high add 2

.byte $00, $00                  ; wave low/high
.byte $21                       ; control
.byte $15, $9B                  ; AD/SR
.byte $14, $8C
.byte $3C, $41                  ; freq. low/high

org $2345
.byte $90, $E8                  ; freq low/high add 1
.byte $CE, $FF                  ; freq low/high add 2
.byte $30, $F2                  ; freq low/high add 3
.byte $00, $00                  ; freq low/high add 4
.byte $0A, $28                  ; freq (to reload) cycle 1/2
.byte $01, $00                  ; freq (to reload) cycle 3/4
.byte $00                       ; freq initial delay
.byte $05                       ; freq effect flag

.byte $FF, $00                  ; wave (to reload) cycle 1/2
.byte $00                       ; wave initial delay
```

```
.byte $00                       ; wave effect flag
.byte $08, $00                  ; wave low/high add 1
.byte $00, $00                  ; wave low/high add 2

.byte $00, $08                  ; wave low/high
.byte $41                       ; control
.byte $19, $BB                  ; AD/SR
.byte $28, $8C
.byte $31, $F2                  ; freq. low/high

org $2364
.byte $90, $E8                  ; freq low/high add 1
.byte $CE, $FF                  ; freq low/high add 2
.byte $30, $F2                  ; freq low/high add 3
.byte $00, $00                  ; freq low/high add 4
.byte $0A, $28                  ; freq (to reload) cycle 1/2
.byte $01, $00                  ; freq (to reload) cycle 3/4
.byte $02                       ; freq initial delay
.byte $05                       ; freq effect flag

.byte $FF, $00                  ; wave (to reload) cycle 1/2
.byte $00                       ; wave initial delay
.byte $00                       ; wave effect flag
.byte $08, $00                  ; wave low/high add 1
.byte $00, $00                  ; wave low/high add 2

.byte $00, $08                  ; wave low/high
.byte $41                       ; control
.byte $19, $BB                  ; AD/SR
.byte $28, $8C
.byte $19, $F6                  ; freq. low/high

org $2383
.byte $90, $E8                  ; freq low/high add 1
.byte $CE, $FF                  ; freq low/high add 2
.byte $30, $F2                  ; freq low/high add 3
.byte $00, $00                  ; freq low/high add 4
.byte $0A, $28                  ; freq (to reload) cycle 1/2
.byte $01, $00                  ; freq (to reload) cycle 3/4
.byte $04                       ; freq initial delay
.byte $05                       ; freq effect flag

.byte $FF, $00                  ; wave (to reload) cycle 1/2
.byte $00                       ; wave initial delay
.byte $00                       ; wave effect flag
.byte $08, $00                  ; wave low/high add 1
.byte $00, $00                  ; wave low/high add 2

.byte $00, $08                  ; wave low/high
.byte $41                       ; control
.byte $19, $BB                  ; AD/SR
.byte $28, $8C
.byte $01, $FA                  ; freq. low/high

org $23A2
.byte $E2, $04                  ; freq low/high add 1
.byte $38, $CD                  ; freq low/high add 2
.byte $00, $00                  ; freq low/high add 3
.byte $00, $00                  ; freq low/high add 4
.byte $09, $02                  ; freq (to reload) cycle 1/2
.byte $01, $00                  ; freq (to reload) cycle 3/4
.byte $00                       ; freq initial delay
.byte $05                       ; freq effect flag

.byte $00, $00                  ; wave (to reload) cycle 1/2
.byte $00                       ; wave initial delay
.byte $00                       ; wave effect flag
.byte $00, $00                  ; wave low/high add 1
.byte $00, $00                  ; wave low/high add 2

.byte $00, $00                  ; wave low/high
.byte $15                       ; control
.byte $15, $E9                  ; AD/SR
.byte $0A, $5A
.byte $3A, $20                  ; freq. low/high

org $23C1
.byte $36, $F7                  ; freq low/high add 1
.byte $38, $4A                  ; freq low/high add 2
.byte $00, $00                  ; freq low/high add 3
.byte $00, $00                  ; freq low/high add 4
.byte $01, $02                  ; freq (to reload) cycle 1/2
.byte $09, $00                  ; freq (to reload) cycle 3/4
.byte $05                       ; freq initial delay
.byte $05                       ; freq effect flag

.byte $00, $00                  ; wave (to reload) cycle 1/2
.byte $00                       ; wave initial delay
.byte $00                       ; wave effect flag
.byte $00, $00                  ; wave low/high add 1
```

```
.byte $00, $00                  ; wave low/high add 2

.byte $00, $00                  ; wave low/high
.byte $15                       ; control
.byte $15, $99                  ; AD/SR
.byte $0A, $5A
.byte $D8, $40                  ; freq. low/high

org $23E0
.byte $16, $0D                  ; freq low/high add 1
.byte $88, $96                  ; freq low/high add 2
.byte $00, $00                  ; freq low/high add 3
.byte $00, $00                  ; freq low/high add 4
.byte $02, $07                  ; freq (to reload) cycle 1/2
.byte $01, $00                  ; freq (to reload) cycle 3/4
.byte $00                       ; freq initial delay
.byte $05                       ; freq effect flag

.byte $00, $00                  ; wave (to reload) cycle 1/2
.byte $00                       ; wave initial delay
.byte $00                       ; wave effect flag
.byte $00, $00                  ; wave low/high add 1
.byte $00, $00                  ; wave low/high add 2

.byte $00, $00                  ; wave low/high
.byte $15                       ; control
.byte $15, $99                  ; AD/SR
.byte $0A, $5A
.byte $3C, $41                  ; freq. low/high

org $23FF
.byte $1E, $FB                  ; freq low/high add 1
.byte $F8, $2A                  ; freq low/high add 2
.byte $E8, $03                  ; freq low/high add 3
.byte $08, $D5                  ; freq low/high add 4
.byte $0A, $01                  ; freq (to reload) cycle 1/2
.byte $0A, $01                  ; freq (to reload) cycle 3/4
.byte $01                       ; freq initial delay
.byte $05                       ; freq effect flag

.byte $00, $00                  ; wave (to reload) cycle 1/2
.byte $00                       ; wave initial delay
.byte $00                       ; wave effect flag
.byte $00, $00                  ; wave low/high add 1
.byte $00, $00                  ; wave low/high add 2

.byte $00, $08                  ; wave low/high
.byte $41                       ; control
.byte $CC, $FC                  ; AD/SR
.byte $FE, $FE
.byte $10, $27                  ; freq. low/high

org $241E
.byte $CA, $08                  ; freq low/high add 1
.byte $F8, $AD                  ; freq low/high add 2
.byte $00, $00                  ; freq low/high add 3
.byte $00, $00                  ; freq low/high add 4
.byte $09, $01                  ; freq (to reload) cycle 1/2
.byte $00, $00                  ; freq (to reload) cycle 3/4
.byte $02                       ; freq initial delay
.byte $05                       ; freq effect flag

.byte $00, $00                  ; wave (to reload) cycle 1/2
.byte $00                       ; wave initial delay
.byte $00                       ; wave effect flag
.byte $00, $00                  ; wave low/high add 1
.byte $00, $00                  ; wave low/high add 2

.byte $00, $08                  ; wave low/high
.byte $41                       ; control
.byte $CC, $FC                  ; AD/SR
.byte $FE, $FE
.byte $CE, $56                  ; freq. low/high

org $243D
.byte $EA, $F2                  ; freq low/high add 1
.byte $90, $65                  ; freq low/high add 2
.byte $00, $00                  ; freq low/high add 3
.byte $00, $00                  ; freq low/high add 4
.byte $08, $01                  ; freq (to reload) cycle 1/2
.byte $00, $00                  ; freq (to reload) cycle 3/4
.byte $03                       ; freq initial delay
.byte $05                       ; freq effect flag

.byte $00, $00                  ; wave (to reload) cycle 1/2
.byte $00                       ; wave initial delay
.byte $00                       ; wave effect flag
.byte $00, $00                  ; wave low/high add 1
.byte $00, $00                  ; wave low/high add 2
```

```
  .byte $00, $08                ; wave low/high
  .byte $41                     ; control
  .byte $CC, $FC                ; AD/SR
  .byte $FE, $FE
  .byte $07, $87                ; freq. low/high


;================================
; Instrument table voice 1
;================================
InstTableV1:
  .byte $23                     ; 00h: freq low add 1 voice 1
  .byte $00                     ; 01h: freq high add 1 voice 1
  .byte $DD                     ; 02h: freq low add 2 voice 1
  .byte $FF                     ; 03h: freq high add 2 voice 1
  .byte $23                     ; 04h: freq low add 3 voice 1
  .byte $00                     ; 05h: freq high add 3 voice 1
  .byte $00                     ; 06h: freq low add 4 voice 1
  .byte $00                     ; 07h: freq high add 4 voice 1
  .byte $03                     ; 08h: freq cycle 1 voice 1
  .byte $05                     ; 09h: freq cycle 2 voice 1
  .byte $02                     ; 0Ah: freq cycle 3 voice 1
  .byte $00                     ; 0Bh: freq cycle 4 voice 1
  .byte $0A                     ; 0Ch: freq delay initial voice 1
  .byte $05                     ; 0Dh: freq effect flag voice 1

  .byte $00                     ; 0Eh: wave cycle 1 voice 1
  .byte $00                     ; 0Fh: wave cycle 2 voice 1
  .byte $00                     ; 10h: wave delay initial voice 1
  .byte $00                     ; 11h: wave effect flag voice 1
  .byte $00                     ; 12h: wave low add 1 voice 1
  .byte $00                     ; 13h: wave high add 1 voice 1
  .byte $00                     ; 14h: wave low add 2 voice 1
  .byte $00                     ; 15h: wave high add 2 voice 1
  .byte $00                     ; 16h: wave low voice 1
  .byte $00                     ; 17h: wave high voice 1

  .byte $19                     ; 18h: control of voice 1
  .byte $A4                     ; 19h: AD voice 1
  .byte $F9                     ; 1Ah: SR voice 1
  .byte $14                     ; 1Bh
  .byte $FE                     ; 1Ch

;================================
; Instrument table voice 2
;================================
InstTableV2:
  .byte $00                     ; 00h: freq low add 1 voice 2
  .byte $00                     ; 01h: freq high add 1 voice 2
  .byte $00                     ; 02h: freq low add 2 voice 2
  .byte $00                     ; 03h: freq high add 2 voice 2
  .byte $01                     ; 04h: freq low add 3 voice 2
  .byte $00                     ; 05h: freq high add 3 voice 2
  .byte $01                     ; 06h: freq low add 4 voice 2
  .byte $00                     ; 07h: freq high add 4 voice 2
  .byte $13                     ; 08h: freq cycle 1 voice 2
  .byte $35                     ; 09h: freq cycle 2 voice 2
  .byte $00                     ; 0Ah: freq cycle 3 voice 2
  .byte $03                     ; 0Bh: freq cycle 4 voice 2
  .byte $00                     ; 0Ch: freq delay initial voice 2
  .byte $00                     ; 0Dh: freq effect flag voice 2

  .byte $32                     ; 0Eh: wave cycle 1 voice 2
  .byte $32                     ; 0Fh: wave cycle 2 voice 2
  .byte $14                     ; 10h: wave delay initial voice 2
  .byte $00                     ; 11h: wave effect flag voice 2
  .byte $0A                     ; 12h: wave low add 1 voice 2
  .byte $00                     ; 13h: wave high add 1 voice 2
  .byte $F6                     ; 14h: wave low add 2 voice 2
  .byte $FF                     ; 15h: wave high add 2 voice 2
  .byte $00                     ; 16h: wave loh voice 2
  .byte $08                     ; 17h: wave high voice 2

  .byte $41                     ; 18h: control of voice 2
  .byte $01                     ; 19h: AD voice 2
  .byte $F7                     ; 1Ah: SR voice 2
  .byte $04                     ; 1Bh
  .byte $14                     ; 1Ch

;================================
; Instrument table voice 3
;================================
InstTableV3:
  .byte $19                     ; 00h: freq low add 1 voice 3
  .byte $00                     ; 01h: freq high add 1 voice 3
  .byte $E7                     ; 02h: freq low add 2 voice 3
  .byte $FF                     ; 03h: freq high add 2 voice 3
  .byte $19                     ; 04h: freq low add 3 voice 3
  .byte $00                     ; 05h: freq high add 3 voice 3
  .byte $00                     ; 06h: freq low add 4 voice 3
```

```
  .byte $00                     ; 07h: freq high add 4 voice 3
  .byte $02                     ; 08h: freq cycle 1 voice 3
  .byte $04                     ; 09h: freq cycle 2 voice 3
  .byte $02                     ; 0Ah: freq cycle 3 voice 3
  .byte $00                     ; 0Bh: freq cycle 4 voice 3
  .byte $06                     ; 0Ch: freq delay initial voice 3
  .byte $05                     ; 0Dh: freq effect flag voice 3

  .byte $32                     ; 0Eh: wave cycle 1 voice 3
  .byte $32                     ; 0Fh: wave cycle 2 voice 3
  .byte $00                     ; 10h: wave delay initial voice 1
  .byte $05                     ; 11h: wave effect flag voice 1
  .byte $14                     ; 12h: wave low add 1 voice 3
  .byte $00                     ; 13h: wave high add 1 voice 3
  .byte $EC                     ; 14h: wave low add 2 voice 3
  .byte $FF                     ; 15h: wave high add 2 voice 3
  .byte $00                     ; 16h: wave lo voice 3
  .byte $06                     ; 17h: wave hi voice 3

  .byte $41                     ; 18h: control of voice 3
  .byte $14                     ; 19h: AD voice 3
  .byte $E8                     ; 1Ah: SR voice 3
  .byte $1E                     ; 1Bh
  .byte $28                     ; 1Ch

; main filter table
MainFilterTable:
  .byte $4D, $01, $D3, $FF, $FB, $FF, $FF, $FF
  .byte $03, $14, $0A, $32, $00, $04, $01, $00

; current filter table
CurFilterTable:
  .byte $4D           ; 0h: add filter low value 1
  .byte $01           ; 1h: add filter high value 1
  .byte $D3           ; 2h: add filter low value 2
  .byte $FF           ; 3h: add filter high value 2
  .byte $FB           ; 4h: add filter low value 3
  .byte $FF           ; 5h: add filter high value 3
  .byte $FF           ; 6h: add filter low value 4
  .byte $FF           ; 7h: add filter high value 4
  .byte $03           ; 8h: filter cycle 1
  .byte $14           ; 9h: filter cycle 2
  .byte $0A           ; Ah: filter cycle 3
  .byte $32           ; Bh: filter cycle 4
  .byte $00           ; Ch: filter initial delay
  .byte $00           ; Dh: filter effect flag
  .byte $01           ; Eh: filter low value (8 bit)
  .byte $00           ; Fh: filter high value (3 bit)

ActFilterTable:
  .byte $00           ; 0h: actual filter low value
  .byte $00           ; 1h: actual filter high value
  .byte $00           ; 2h: actual filter cycle 1
  .byte $00           ; 3h: actual filter cycle 2
  .byte $00           ; 4h: actual filter cycle 3
  .byte $00           ; 5h: actual filter cycle 4

;================================
;current istrument table voice 1
;================================
CurInstTableV1:
  .byte $23, $00                ; 00h: freq low/high add 1 voice 1
  .byte $DD, $FF                ; 02h: freq low/high add 2 voice 1
  .byte $23, $00                ; 04h: freq low/high add 3 voice 1
  .byte $00, $00                ; 06h: freq low/high add 4 voice 1
  .byte $03, $05                ; 08h: freq (to reload) cycle 1/2 voice 1
  .byte $02, $00                ; 0ah: freq (to reload) cycle 3/4 voice 1
  .byte $00                     ; 0ch: freq initial delay voice 1
  .byte $05                     ; 0dh: freq effect flag voice 1

  .byte $32, $32                ; 0eh: wave (to reload) cycle 1/2 voice 1
  .byte $00                     ; 10h: wave initial delay voice 1
  .byte $00                     ; 11h: wave effect flag voice 1
  .byte $0A, $00                ; 12h: wave low/high add 1 voice 1
  .byte $F6, $FF                ; 14h: wave low/high add 2 voice 1
  .byte $00, $08                ; 16h: wave low/high voice 1
  .byte $2C, $1A                ; 18h: freq low/high voice 1
  .byte $10                     ; 1Ah: control of voice 1
  .byte $00                     ; 1Bh:
  .byte $00                     ; 1Ch:
  .byte $00, $00                ; 1Dh: freq cycle 1/2 voice 1
  .byte $01, $00                ; 1Fh: freq cycle 3/4 voice 1
  .byte $00, $04                ; 21h: wave cycle 1/2 voice 1

;================================
;current istrument table voice 2
;================================
CurInstTableV2:
  .byte $00, $00                ; 00h: freq low/high add 1 voice 2
  .byte $00, $00                ; 02h: freq low/high add 2 voice 2
```

```
  .byte $01, $00                ; 04h: freq low/high add 3 voice 2
  .byte $01, $00                ; 06h: freq low/high add 4 voice 2
  .byte $13, $35                ; 08h: freq (to reload) cycle 1/2 voice 2
  .byte $37, $03                ; 0ah: freq (to reload) cycle 3/4 voice 2
  .byte $00                     ; 0ch: freq. initial delay voice 2
  .byte $00                     ; 0dh: freq. effect flag voice 2
  .byte $32, $32                ; 0eh: wave cycle 1/2 voice 2
  .byte $00                     ; 10h: wave initial delay voice 2
  .byte $00                     ; 11h: wave effect flag voice 2
  .byte $0A, $00                ; 12h: wave low/high add 1 voice 2
  .byte $F6, $FF                ; 14h: wave low/high add 2 voice 2
  .byte $00, $08                ; 16h: wave low/high voice 2
  .byte $2C, $1A                ; 18h: freq low/high voice 2
  .byte $41                     ; 1A: control of voice 2
  .byte $00                     ; 1B:
  .byte $00                     ; 1C:
  .byte $00, $00                ; 1D: freq cycle 1/2 voice 2
  .byte $02, $00                ; 1F: freq cycle 3/4 voice 2
  .byte $1E, $32                ; 21h: wave cycle 1/2 voice 1

;================================
;current istrument table voice 3
;================================
CurInstTableV3:
  .byte $19, $00               ; 00h: freq low/high add 1 voice 3
  .byte $E7, $FF               ; 02h: freq low/high add 2 voice 3
  .byte $19, $00               ; 04h: freq low/high add 3 voice 3
  .byte $00, $00               ; 06h: freq low/high add 4 voice 3
  .byte $02, $04               ; 08h: freq (to reload) cycle 1/2 voice 3
  .byte $02, $00               ; 0ah: freq (to reload) cycle 3/4 voice 3
  .byte $00                    ; 0ch: freq. initial delay voice 3
  .byte $05                    ; 0dh: freq. effect flag voice 3
  .byte $32, $32               ; 0eh: wave cycle 1/2 voice 3
  .byte $00                    ; 10h: wave initial delay voice 3
  .byte $05                    ; 11h: wave effect flag voice 3
  .byte $14, $00               ; 12h: wave low/high add 1 voice 3
  .byte $EC, $FF               ; 14h: wave low/high add 2 voice 3
  .byte $00, $06               ; 16h: wave low/high voice 3
  .byte $B0, $0E               ; 18h: freq low/high voice 3
  .byte $41                    ; 1Ah: control of voice 3
  .byte $00                    ; 1Bh:
  .byte $00                    ; 1Ch:
  .byte $00, $00               ; 1D: freq cycle 1/2 voice 2
  .byte $01, $00               ; 1F: freq cycle 3/4 voice 2
  .byte $00, $1F               ; 21h: wave cycle 1/2 voice 1

voiceNumber:
  .byte $00                    ; voice number (0,1,2) on where apply filter

tmpCutF:
  .byte $00

minDuration:
  .byte $04

; Play music flag: 0=no music, 1=music
playMusicV1:
  .byte $00
playMusicV2:
  .byte $00
playMusicV3:
  .byte $01

tmpVIndex:
  .byte $FF                     ; tmp voices index

tmpAVIndex:
  .byte $FE                     ; tmp address index voice

durationTable:
  .byte $04, $08, $0C, $10
  .byte $14, $18, $1C, $20
  .byte $24, $28, $2C, $30
  .byte $34, $38, $3C, $40
  .byte $44, $48, $4C, $50
  .byte $54, $58, $5C, $60
  .byte $64, $68, $6C, $70
  .byte $74, $78, $7C, $80

; offset for control register of each voice
offControlVoice:
  .byte $02                     ; $D404 control voice 1
  .byte $09                     ; $D40A control voice 2
  .byte $10                     ; $D411 control voice 3

; offset on one voice into instrument table
offInstrVoice:
  .byte $00, $1D, $3A

; offset for current instrument into music effect
```

```
offCurrEffect
  .byte $17, $3A, $5D

highFreq:
  .byte $17, $28, $3A, $4C, $60, $75, $8B, $A3
  .byte $BC, $D6, $F2, $10, $2F, $50, $73, $99
  .byte $C0, $EA, $16, $45, $77, $AC, $E4, $1F
  .byte $5E, $A0, $E7, $32, $80, $D4, $2D, $8B
  .byte $EE, $58, $C8, $3E, $BC, $41, $CD, $63
  .byte $01, $A8, $5A, $16, $DD, $B0, $90, $7C
  .byte $77, $81, $9B, $C5, $02, $51, $B4, $2C
  .byte $BA, $60, $1F, $F9, $EF, $03, $36, $8B
  .byte $03, $A1, $67, $57, $73, $C0, $3E, $F1
  .byte $DE, $05, $6C, $16, $06, $43, $CE, $AE
  .byte $00

lowFreq:
  .byte $01, $01, $01, $01, $01, $01, $01, $01
  .byte $01, $01, $01, $02, $02, $02, $02, $02
  .byte $02, $02, $03, $03, $03, $03, $03, $04
  .byte $04, $04, $04, $05, $05, $05, $06, $06
  .byte $06, $07, $07, $08, $08, $09, $09, $0A
  .byte $0B, $0B, $0C, $0D, $0D, $0E, $0F, $10
  .byte $11, $12, $13, $14, $16, $17, $18, $1A
  .byte $1B, $1D, $1F, $20, $22, $25, $27, $29
  .byte $2C, $2E, $31, $34, $37, $3A, $3E, $41
  .byte $45, $4A, $4E, $53, $58, $5D, $62, $68
  .byte $00


;2615  jsr  executePatternV1
;      jsr  executePatternV2
;      jsr  executePatternV3
;      jsr  makeTimbreV1
;      jsr  makeTimbreV2
;      jsr  makeTimbreV3
;      jsr  makeFilterEff
;      rts

;================================
; Set tracks at offset given by y
;================================
setTracks:
      lda  trackTable+1,y         ; read the minime note duration
      sta  minDuration
      and  #$0F
      sta  buildDurTable+1

      ldx  #$02
      stx  tmpVIndex              ; tmp voices index

      ldx  #$04
      stx  tmpAVIndex             ; tmp address voice index
      dey

nextVoice:
      lda  trackTable,y           ; test if low and high address are 0
      ora  trackTable+1,y
      beq  afterSetting           ; skip if address are 0

      ldx  tmpAVIndex             ; tmp address voice index
      lda  trackTable,y
      sta  $E0,x                  ; store low pattern addr. voice in x
      lda  trackTable+1,y
      sta  $E1,x                  ; store high pattern addr. voice in x

      sty  $FD                    ; tmp track offset
      ldx  tmpVIndex              ; tmp voices index
      ldy  offInstrVoice,x        ; offset of one voice into instr. table
      lda  #$00
      sta  $EC,x
      sta  InstTableV1+13,y       ; freq. effect voice from x
      sta  InstTableV1+17,y       ; wave effect voice from x
      lda  #$07
      sta  $E9,x                  ; init stack index for voice x
      lda  #$01
      sta  $E6,x                  ; store actual note duration (length) voice in x
      sta  playMusicV1,x          ; play music flag
      ldy  $FD                    ; tmp track offset

afterSetting:
      dey
      dey
      dec  tmpAVIndex             ; tmp address voice index
      dec  tmpAVIndex             ; tmp address voice index
      dec  tmpVIndex              ; tmp voices index
      bpl  nextVoice
      clc
      lda  #$00
```

```
buildDurTable:                  ; build the table of note duration
      adc  #$04                 ; add minime duration
      sta  durationTable,x
      inx
      cpx  #$20
      bcc  buildDurTable
      rts

;=================================
; Init the engine
;=================================
initEngine:
      ldx  #$16
loopInitE:
      lda  #$08
      sta  $D400,x              ; Voice 1: Frequency control (lo byte)
      lda  #$00
      sta  $D400,x              ; Voice 1: Frequency control (lo byte)
      dex
      bpl  loopInitE

      sta  CurInstTableV1+28
      sta  CurInstTableV2+28
      sta  CurInstTableV3+28
      sta  CurFilterTable+13    ; filter flag effect
      sta  playMusicV1          ; play music flag voice 1
      sta  playMusicV2          ; play music flag voice 2
      sta  playMusicV3          ; play music flag voice 3
      stx  voiceNumber          ; voice number (0,1,2) on where apply filter
      lda  #$F0
      sta  $D417                ; Filter resonance control/voice input control
      lda  #$0F
      sta  TEMP
      sta  $D418                ; Select volume and filter mode
      rts

;=================================
; Voice 1: Reload the cycle for
; Wave effect
;=================================
reloadWaveCycleV1:
      ldx  CurInstTableV1+22    ; read current wave low voice 1
      ldy  CurInstTableV1+23    ; read current wave high voice 1

reloadWCycleV1:
      stx  $F1                  ; wave low v1
      sty  $F2                  ; wave high v1
      lda  CurInstTableV1+14    ; read current (to reload) wave cycle 1 voice 1
      sta  CurInstTableV1+33    ; store current wave cycle 1 voice 1
      lda  CurInstTableV1+15    ; read current (to reload) wave cycle 2 voice 1
      sta  CurInstTableV1+34    ; store current wave cycle 2 voice 1
      rts

;=================================
; Voice 2: Reload the cycle for
; Wave effect
;=================================
reloadWaveCycleV2:
      ldx  CurInstTableV2+22    ; read current wave low voice 2
      ldy  CurInstTableV2+23    ; read current wave high voice 2
reloadWCycleV2:
      stx  $F3                  ; wave low v2
      sty  $F4                  ; wave high v2
      lda  CurInstTableV2+14    ; read current (to reload) wave cycle 1 voice 2
      sta  CurInstTableV2+33    ; store current wave cycle 1 voice 2
      lda  CurInstTableV2+15    ; read current (to reload) wave cycle 2 voice 2
      sta  CurInstTableV2+34    ; store current wave cycle 2 voice 2
      rts

;=================================
; Voice 3: Reload the cycle for
; Wave effect
;=================================
reloadWaveCycleV3:
      ldx  CurInstTableV3+22    ; read current wave low voice 3
      ldy  CurInstTableV3+23    ; read current wave high voice 3
reloadWCycleV3:
      stx  $F5                  ; wave low v3
      sty  $F6                  ; wave high v3
      lda  CurInstTableV3+14    ; read current (to reload) wave cycle 1 voice 3
      sta  CurInstTableV3+33    ; store current wave cycle 1 voice 3
      lda  CurInstTableV3+15    ; read current (to reload) wave cycle 2 voice 3
      sta  CurInstTableV3+34    ; store current wave cycle 2 voice 3
      rts

;=================================
; Reload  the filter cycle
;=================================
reloadFilterCycle:
      ldx  CurFilterTable+14    ; read filter low value (8 bit)
```

```
      ldy  CurFilterTable+15      ; read filter high value (3 bit)
      stx  ActFilterTable+0       ; store actual filter low value (8 bit)
      sty  ActFilterTable+1       ; store actual filter high value (3 bit)

reloadFCycle:
      lda  CurFilterTable+8       ; filter cycle 1
      sta  ActFilterTable+2       ; store actual filter cycle 1

      lda  CurFilterTable+9       ; filter cycle 2
      sta  ActFilterTable+3       ; store actual filter cycle 2

      lda  CurFilterTable+10      ; filter cycle 3
      sta  ActFilterTable+4       ; store actual filter cycle 3

      lda  CurFilterTable+11      ; filter cycle 4
      sta  ActFilterTable+5       ; store actual filter cycle 4
      rts

;================================
; Set Sound Effect for voice
; a/y address of table | x voice
;================================
setSoundEffect:
      sta  $FD
      sty  $FE
      stx  tmpVIndex              ; tmp voices index
      lda  #$00
      sta  $D417                  ; Filter resonance control/voice input control
      lda  #$0F
      sta  $D418                  ; Select volume and filter mode
      lda  offControlVoice,x
      sta  voiceOffset+1
      lda  #$08                   ; control: test bit on
      ldy  offControlVoice,x
      sta  $D402,y                ; Voice 1: Wave form pulsation amplitude (lo byte)

; set voice ADSR, control and wave hi/lo
      ldy  #$1A
      ldx  #$04

loopVoiceSet:
     lda  ($FD),y                 ; read value from table
voiceOffset:
      sta  $D410,x                ; store in sid register of given voice
      dey
      dex
      bpl  loopVoiceSet

      ldy  #$1D
      ldx  voiceOffset+1
      lda  ($FD),y
      sta  $D3FE,x                ; store freq. low voice x
      iny
      lda  ($FD),y
      sta  $D3FF,x                ; store freq. high voice x

      ldy  tmpVIndex              ; tmp voices index
      ldx  offCurrEffect,y
      ldy  #$1E
      sta  CurInstTableV1+2,x     ; current freq high voice from x
      dey
      lda  ($FD),y
      sta  CurInstTableV1+1,x     ; current freq low voice from x
      dey
      lda  ($FD),y
      sta  CurInstTableV1+5,x     ;
      dey
      lda  ($FD),y
      sta  CurInstTableV1+4,x     ;

      ldy  #$18
      lda  ($FD),y
      sta  CurInstTableV1+3,x     ; current control of voice from x

      ldy  #$17

copyLoop:
      lda  ($FD),y
      sta  CurInstTableV1+0,x     ; current instrument table voice 1
      dex
      dey
      bpl  copyLoop

      inx                         ; zero means voice 1
      bne  notVoice1

      lda  CurInstTableV1+17      ; read current wave effect flag voice 1
      beq  reloadFreqCycleV1
      jsr  reloadWaveCycleV1
```

```
;================================
; Voice 1: Reload the cycle for
; Freq effect
;================================
reloadFreqCycleV1:
      ldx  CurInstTableV1+24      ; read current freq. low voice 1
      ldy  CurInstTableV1+25      ; read current freq. high voice 1
      stx  $F7                    ; store freq. low voice 1
      sty  $F8                    ; store freq. high voice 1

reloadFCycleV1:
      lda  CurInstTableV1+11      ; read current (to reload) freq cycle 4 voice 1
      sta  CurInstTableV1+32      ; store current freq cycle 4 voice 1
      lda  CurInstTableV1+10      ; read current (to reload) freq cycle 3 voice 1
      sta  CurInstTableV1+31      ; store current freq cycle 3 voice 1
      lda  CurInstTableV1+9       ; read current (to reload) freq cycle 2 voice 1
      sta  CurInstTableV1+30      ; store current freq cycle 2 voice 1
      lda  CurInstTableV1+8       ; read current (to reload) freq cycle 1 voice 1
      sta  CurInstTableV1+29      ; store current freq cycle 1 voice 1
      rts

notVoice1:
      cpx  #$46                   ; $46 means voice 2
      beq  notVoice2

      lda  CurInstTableV2+17      ; read current wave effect flag voice 2
      beq  skipReloadWCV2
      jsr  reloadWaveCycleV2

skipReloadWCV2:
      lda  CurInstTableV2+13      ; read current freq. flag effect voice 2
      beq  exitRFCV2

;================================
; Voice 2: Reload the cycle for
; Freq effect
;================================
reloadFreqCycleV2:
      ldx  CurInstTableV2+24      ; read current freq. low voice 2
      ldy  CurInstTableV2+25      ; read current freq. high voice 2
      stx  $F9                    ; store freq. low voice 2
      sty  $FA                    ; store freq. high voice 2

reloadFCycleV2:
      lda  CurInstTableV2+11      ; read current (to reload) freq cycle 4 voice 2
      sta  CurInstTableV2+32      ; store current freq cycle 4 voice 2
      lda  CurInstTableV2+10      ; read current (to reload) freq cycle 3 voice 2
      sta  CurInstTableV2+31      ; store current freq cycle 3 voice 2
      lda  CurInstTableV2+9       ; read current (to reload) freq cycle 2 voice 2
      sta  CurInstTableV2+30      ; store current freq cycle 2 voice 2
      lda  CurInstTableV2+8       ; read current (to reload) freq cycle 1 voice 2
      sta  CurInstTableV2+29      ; store current freq cycle 1 voice 2
exitRFCV2:
      rts

notVoice2:
      lda  CurInstTableV3+17      ; read wave effect flag voice 3
      beq  skipReloadWCV3
      jsr  reloadWaveCycleV3

skipReloadWCV3:
      lda  CurInstTableV3+13      ; read freq. effect flag voice 3
      beq  exitRFCV2

;================================
; Voice 3: Reload the cycle for
; Freq effect
;================================
reloadFreqCycleV3:
      ldx  CurInstTableV3+24      ; read current freq. low voice 3
      ldy  CurInstTableV3+25      ; read current freq. high voice 3
      stx  $FB                    ; store freq. low voice 3
      sty  $FC                    ; store freq. high voice 3

reloadFCycleV3:
      lda  CurInstTableV3+11      ; read current (to reload) freq cycle 4 voice 3
      sta  CurInstTableV3+32      ; store current freq cycle 4 voice 3
      lda  CurInstTableV3+10      ; read current (to reload) freq cycle 3 voice 3
      sta  CurInstTableV3+31      ; store current freq cycle 3 voice 3
      lda  CurInstTableV3+9       ; read current (to reload) freq cycle 2 voice 3
      sta  CurInstTableV3+30      ; store current freq cycle 2 voice 3
      lda  CurInstTableV3+8       ; read current (to reload) freq cycle 1 voice 3
      sta  CurInstTableV3+29      ; store current freq cycle 1 voice 3
      rts

;==================================
; Voice 1: JSRT instruction
; execute a subroutine and set the
; number of halftone to transpose
; #1 halfnote to add (?)
```

```
; #2 low address
; #3 high address
;==================================
inst_C6_v1:
      lda  ($E0),y
      sta  $EC
      iny
      lda  #$04
      .byte $2C                    ; bit instruction (hide next instruction)

;==================================
; Voice 1: JSR instruction
; execute a subroutine pattern
; #1 low address
; #2 high address
;==================================
inst_C2_v1:
      lda  #$03
      ldx  $E9                     ; read stack index of voive 1
      clc
      adc  $E0                     ; fix for return
      sta  $30,x                   ; store return address low
      lda  #$00
      adc  $E1
      sta  $38,x                   ; store return address high
      dec  $E9                     ; dec stack index of voice 1
      lda  ($E0),y                 ; read next byte: low address
      tax
      iny
      lda  ($E0),y                 ; read next byte: high address
      stx  $E0                     ; change address
      sta  $E1
      jmp  readNextV1

;==================================
; Voice 1: SET2CI instruction
; Set 2 values of current instument
; #1 index
; #2 val1
; #3 val2
;==================================
inst_DE_v1:
      lda  ($E0),y                 ; read next byte: index
      tax
      iny
      lda  ($E0),y                 ; read next byte: first value
      sta  CurInstTableV1+0,x
      iny
      lda  ($E0),y                 ; read next byte: second value
      sta  CurInstTableV1+1,x
      lda  #$04
      jmp  adjustPatternV1

;==================================
; Voice 1: SETNI instruction
; Set first N items of Instrument
; table
; #1 number of items
; #2 low addr.
; #3 high addr.
;==================================
inst_D2_v1:
      lda  ($E0),y                 ; read next byte: number of items
      tax
      iny
      lda  ($E0),y                 ; read next byte: low addr
      sta  copyV1+1
      iny
      lda  ($E0),y                 ; read next byte: high addr
      sta  copyV1+2
copyV1:
      lda  ins02,x                 ; read istrument byte to copy
      sta  InstTableV1+0,x         ; store istrument byte to use
      dex
      bpl  copyV1
      lda  #$04
      jmp  adjustPatternV1

;==================================
; Voice 1: FOR instruction
; Repeat a block n times
; #1 the number of repeat to performe
;==================================
inst_CC_v1:
      ldx  $E9                     ; read stack index of voice 1
      lda  #$02
      clc
      adc  $E0                     ; fix for next inst.
      sta  $30,x                   ; store load addr. into stack
      lda  #$00
```

```
      adc  $E1
      sta  $38,x                  ; store high addr. into stack
      lda  ($E0),y                ; read next byte: number of repeat
      sta  $40,x
      dec  $E9                    ; dec stack index of voice 1
      lda  #$02
      jmp  adjustPatternV1

;=================================
; Voice 1: SETFI instruction
; Set instrument frequency effect
; by reading from the given table
; #1 low addr
; #1 high addr
;=================================
inst_F0_v1:
      lda  ($E0),y                ; read next byte: low addr
      sta  ftableV1+1
      iny
      lda  ($E0),y                ; read next byte: high addr
      sta  ftableV1+2
      ldx  #$0D
ftableV1:
      lda  insF03,x               ; read from table
      sta  InstTableV1+0,x        ; store istrument freq. values
      dex
      bpl  ftableV1
      jmp  adjust3PatternV1

;=================================
; Voice 1: JMP instruction
; jump to the given location
; #1 low address
; #2 high address
;=================================
inst_C4_v1:
      lda  ($E0),y               ; read next byte of pattern: low addr
      tax
      iny
      lda  ($E0),y               ; read next byte of pattern: high addr
      stx  $E0                   ; change address
      sta  $E1
      jmp  readNextV1

;=================================
; Voice 1: SET instruction
; set a value in the table at the
; given index
; #1 index
; #2 value
;=================================
inst_CA_v1:
      lda  ($E0),y               ; read next byte of pattern
      tax
      iny
      lda  ($E0),y               ; read next byte of pattern
      sta  InstTableV1+0,x
      jmp  adjust3PatternV1

;=================================
; Voice 1: NEXT instruction
; Execute the next cycle of the FOR
;=================================
inst_CE_v1:
      ldx  $E9                   ; read stack index of voice 1
      dec  $41,x                 ; number of repeat for voice 1
      beq  endCycle1             ; stop to cycle if 0
      ldy  $31,x                 ; read low addr into stack of voice 1
      lda  $39,x                 ; read high addr into stack of voice 1
      sty  $E0                   ; change address
      sta  $E1
      jmp  readNextV1

endCycle1:
      inc  $E9                   ; inc stack index of voice 1
      tya
      jmp  adjustPatternV1

;=================================
; Voice 1: RTS instruction
; return to the stored location
; in the stack
; If there is not an stored address,
; stop music
;=================================
inst_C0_v1:
      ldy  $E9                    ; read stack index of voice 1
      cpy  #$07
      beq  stop_v1
      inc  $E9                    ; inc stack index of voice 1
```

```
      ldx  $31,y                  ; read low address
      lda  $0039,y                ; read high address
      stx  $E0                    ; change address
      sta  $E1
      jmp  readNextV1

stop_v1:
      dec  playMusicV1            ; stop music
      rts

;================================
; Voice1: SETCI istruction
; Set the current instrument value
; at the given index
; #1 index
; #2 value
;================================
inst_D6_v1:
      lda  ($E0),y                ; read index of current instrument value
      tax
      iny
      lda  ($E0),y                ; read value to set
      sta  CurInstTableV1+0,x     ; store value in current instrument
      jmp  adjust3PatternV1

;================================
; Voice 1: INSTR instruction
; set 5 instrument parameters
; from the given location
; #1 low address
; #2 high address
;================================
inst_D4_v1:
      lda  ($E0),y               ; read next byte of pattern: low address
      sta  $EF
      iny
      lda  ($E0),y               ; read next byte of pattern: high address
      sta  $F0
      ldy  #$04
loopD4v1:
      lda  ($EF),y               ; read bytes from given address
      sta  InstTableV1+24,y      ; copy to instrument voice location
      dey
      bpl  loopD4v1
      jmp  adjust3PatternV1


;================================
; Voice 2: JSR instruction
; execute a subroutine pattern
; #1 low address
; #2 high address
;================================
inst_C2_v2:
      lda  #$03
      ldx  $EA                    ; read stack index of voice 2
      clc
      adc  $E2                    ; fix for return
      sta  $A8,x                  ; store return address low
      lda  #$00
      adc  $E3
      sta  $B0,x                  ; store return address high
      dec  $EA                    ; dec stack index of voice 2
      lda  ($E2),y                ; read next byte: low address
      tax
      iny
      lda  ($E2),y                ; read next byte: high address
      stx  $E2                    ; change address
      sta  $E3
      jmp  readNextV2

;================================
; Voice 2: SET2I instruction
; Set 2 values of instruction table
; at the given index
; #1 index
; #2 val1
; #3 val3
;================================
inst_DC_v2:
      lda  ($E2),y                ; read next byte: index
      tax
      iny
      lda  ($E2),y                ; read next byte: first value
      sta  InstTableV2+0,x
      iny
      lda  ($E2),y                ; read next byte: second value
      sta  InstTableV2+1,x
      lda  #$04
      jmp  adjustPatternV2
```

```
;================================
; Voice 2: SET2CI instruction
; Set 2 values of current instument
; #1 index
; #2 val1
; #3 val2
;================================
inst_DE_v2:
      lda  ($E2),y                ; read next byte: index
      tax
      iny
      lda  ($E2),y                ; read next byte: first value
      sta  CurInstTableV2+0,x
      iny
      lda  ($E2),y                ; read next byte: second value
      sta  CurInstTableV2+1,x
      lda  #$04
      jmp  adjustPatternV2

;================================
; Voice 2: SETNI instruction
; Set first N items of Istrument
; table
; #1 number of items
; #2 low addr.
; #3 high addr.
;================================
inst_D2_v2:
      lda  ($E2),y                ; read next byte: number of items
      tax
      iny
      lda  ($E2),y                ; read next byte: low addr
      sta  copyV2+1
      iny
      lda  ($E2),y                ; read next byte: high addr
      sta  copyV2+2
copyV2:
      lda  $3517,x                ; read istrument byte to copy
      sta  InstTableV2+0,x        ; store istrument byte to use
      dex
      bpl  copyV2
      lda  #$04
      jmp  adjustPatternV2

;================================
; Voice 2: FOR instruction
; Repeat a block n times
; #1 the number of repeat to performe
;================================
inst_CC_v2:
      ldx  $EA                    ; read stack index of voice 2
      lda  #$02
      clc
      adc  $E2                    ; fix for next inst.
      sta  $A8,x                  ; store load addr. into stack
      lda  #$00
      adc  $E3
      sta  $B0,x                  ; store high addr. into stack
      lda  ($E2),y                ; read next byte: number of repeat
      sta  $B8,x
      dec  $EA                    ; dec stack index of voice 2
      lda  #$02
      jmp  adjustPatternV2

;================================
; Voice 2: SETFI instruction
; Set instrument frequency effect
; by reading from the given table
; #1 low addr
; #1 high addr
;================================
inst_F0_v2:
      lda  ($E2),y                ; read next byte: low addr
      sta  ftableV2+1
      iny
      lda  ($E2),y                ; read next byte: high addr
      sta  ftableV2+2
      ldx  #$0D
ftableV2:
      lda  insF03,x               ; read from table
      sta  InstTableV2+0,x        ; store istrument freq. values
      dex
      bpl  ftableV2
      jmp  adjust3PatternV2

;================================
; Voice 2: JMP instruction
; jump to the given location
; #1 low address
```

```
; #2 high address
;================================
inst_C4_v2:
      lda  ($E2),y               ; read next byte of pattern: low addr
      tax
      iny
      lda  ($E2),y               ; read next byte of pattern: high addr
      stx  $E2                   ; change address
      sta  $E3
      jmp  readNextV2

;================================
; Voice 2: SET instruction
; set a value in the table at the
; given index
; #1 index
; #2 value
;================================
inst_CA_v2:
      lda  ($E2),y                ; read next byte of pattern
      tax
      iny
      lda  ($E2),y                ; read next byte of pattern
      sta  InstTableV2+0,x
      jmp  adjust3PatternV2

;================================
; Voice 2: NEXT instruction
; Execute the next cycle of the FOR
;================================
inst_CE_v2:
      ldx  $EA                    ; read stack index of voice 2
      dec  $B9,x                  ; number of repeat for voice 2
      beq  endCycle2              ; stop to cycle if 0
      ldy  $A9,x                  ; read low addr into stack of voice 2
      lda  $B1,x                  ; read high addr into stack of voice 2
      sty  $E2                    ; change address
      sta  $E3
      jmp  readNextV2

endCycle2:
      inc  $EA                    ; inc stack index of voice 2
      tya
      jmp  adjustPatternV2

;================================
; Voice 2: RTS instruction
; return to the stored location
; in the stack
;================================
inst_C0_v2:
      ldy  $EA                    ; read stack index of voice 2
      cpy  #$07
      beq  stop_v2
      inc  $EA                    ; inc stack index of voice 2
      ldx  $A9,Y                  ; read low address
      lda  $00B1,Y                ; read high address
      stx  $E2                    ; change address
      sta  $E3
      jmp  readNextV2

stop_v2:
      dec  playMusicV2            ; stop music
      rts

;================================
; Voice2:  istruction
; Set the current istrument value
; at the given index
; #1 index
; #2 value
;================================
inst_D6_v2:
      lda  ($E2),y                ; read index of current instrument value
      tax
      iny
      lda  ($E2),y                ; read value to set
      sta  CurInstTableV2+0,x     ; store value in current instrument
      jmp  adjust3PatternV2

;================================
; Voice 2: INSTR instruction
; set 5 instrument parameters
; from the given location
; #1 low address
; #2 high address
;================================
inst_D4_v2:
      lda  ($E2),y               ; read next byte of pattern: low address
      sta  $EF
```

```
      iny
      lda  ($E2),y               ; read next byte of pattern: high address
      sta  $F0
      ldy  #$04
loopD4v2:
      lda  ($EF),y               ; read bytes from given address
      sta  InstTableV2+24,y      ; copy to instrument voice location
      dey
      bpl  loopD4v2
      jmp  adjust3PatternV2


;==================================
; Voice 3: JSRT instruction
; execute a subroutine and set the
; number of halftone to transpose
; #1 halfnote to add (?)
; #2 low address
; #3 high address
;==================================
inst_C6_v3:
      lda  ($E4),y
      sta  $EE
      iny
      lda  #$04
      .byte $2C

;==================================
; Voice 3: JSR instruction
; execute a subroutine pattern
; #1 low address
; #2 high address
;==================================
inst_C2_v3:
      lda  #$03
      ldx  $EB                   ; read stack index of voice 3
      clc
      adc  $E4                   ; fix for return
      sta  $60,x                 ; store return address low
      lda  #$00
      adc  $E5
      sta  $68,x                 ; store return address high
      dec  $EB                   ; dec stack index of voice 3
      lda  ($E4),y               ; read next byte: low address
      tax
      iny
      lda  ($E4),y               ; read next byte: high address
      stx  $E4                   ; change address
      sta  $E5
      jmp  readNextV3

;==================================
; Voice 3:  EXCT instruction
; Excecute a code gave by paramethers
; #1 low addr.
; #2 high addr.
;==================================
inst_D8_v3:
      lda  #$2E                  ; return high point
      pha
      lda  #$D2                  ; return low point
      pha
      lda  ($E4),y               ; read low address
      sta  $EF
      iny
      lda  ($E4),y               ; read high address
      sta  $F0
      jmp  ($00EF)               ; execute code at read address

;==================================
; Voice 3: SET2I instruction
; Set 2 values of instruction table
; at the given index
; #1 index
; #2 val1
; #3 val3
;==================================
inst_DC_v3:
      lda  ($E4),y               ; read next byte: index
      tax
      iny
      lda  ($E4),y               ; read next byte: first value
      sta  InstTableV3+0,x
      iny
      lda  ($E4),y               ; read next byte: second value
      sta  InstTableV3+1,x
      lda  #$04
      jmp  adjustPatternV3

;==================================
```

```
; Voice 3: SET2CI instruction
; Set 2 values of current instument
; #1 index
; #2 val1
; #3 val2
;=================================
inst_DE_v3:
      lda  ($E4),y               ; read next byte of pattern: index
      tax
      iny
      lda  ($E4),y               ; read next byte of pattern: first value
      sta  CurInstTableV3+0,x
      iny
      lda  ($E4),y               ; read next byte of pattern: second value
      sta  CurInstTableV3+1,x
      lda  #$04
      jmp  adjustPatternV3

;=================================
; Voice 3: FILTA instruction
; Set filter table
; #1: low addr.
; #2: high addr.
;=================================
inst_E2_v3:
      lda  ($E4),y               ; read next byte of pattern
      sta  copyFil+1
      iny
      lda  ($E4),y               ; read next byte of pattern
      sta  copyFil+2
      ldx  #$0F

copyFil:
      lda  fil02,x               ; value to copy
      sta  MainFilterTable+0,x   ; copy in main filter table
      dex
      bpl  copyFil
      jmp  adjust3PatternV3

;=================================
; Voice 3: SETNI instruction
; Set first N items of Istrument
; table
; #1 number of items
; #2 low addr.
; #3 high addr.
;=================================
inst_D2_v3:
      lda  ($E4),y                ; read next byte: number of items
      tax
      iny
      lda  ($E4),y                ; read next byte: low addr
      sta  copyV3+1
      iny
      lda  ($E4),y                ; read next byte: high addr
      sta  copyV3+2
copyV3:
      lda  insF01,x               ; read istrument byte to copy
      sta  InstTableV3+0,x        ; store istrument byte to use
      dex
      bpl  copyV3
      lda  #$04
      jmp  adjustPatternV3

;=================================
; Voice 3: FOR instruction
; Repeat a block n times
; #1 the number of repeat to performe
;=================================
inst_CC_v3:
      ldx  $EB                    ; read stack index of voice 3
      lda  #$02
      clc
      adc  $E4                    ; fix for next inst.
      sta  $60,x                  ; store load addr. into stack
      lda  #$00
      adc  $E5
      sta  $68,x                  ; store high addr. into stack
      lda  ($E4),y                ; read next byte: number of repeat
      sta  $70,x
      dec  $EB                    ; dec stack index of voice 3
      lda  #$02
      jmp  adjustPatternV3

;=================================
; Voice 3: SETFI instruction
; Set instrument frequency effect
; by reading from the given table
; #1 low addr
; #1 high addr
```

```
;================================
inst_F0_v3:
      lda  ($E4),y                ; read next byte: low addr
      sta  ftableV3+1
      iny
      lda  ($E4),y                ; read next byte: high addr
      sta  ftableV3+2
      ldx  #$0D
ftableV3:
      lda  insF01,x               ; read from table
      sta  InstTableV3+0,x        ; store istrument freq. values
      dex
      bpl  ftableV3
      jmp  adjust3PatternV3

;================================
; Voice 3: JMP instruction
; jump to the given location
; #1 low address
; #2 high address
;================================
inst_C4_v3:
      lda  ($E4),y               ; read next byte of pattern: low addr
      tax
      iny
      lda  ($E4),y               ; read next byte of pattern: high addr
      stx  $E4                   ; change address
      sta  $E5
      jmp  readNextV3

setMax:
      ldy  #$F4                  ; max resonance/filter on voice 3
      sty  $D417                 ; Filter resonance control/voice input control
      stx  voiceNumber           ; voice number (0,1,2) on where apply filter
      ldx  #$1F                  ; low pass filter
      stx  $D418                 ; Select volume and filter mode
      lda  #$01
      jmp  adjustPatternV3

;================================
; Voice 3: LF3 instruction
; Set low filter on voice 3 with
; max resonance
;================================
inst_E0_v3:
      ldx  #$02
      bne  setMax

;================================
; Voice 3: SET instruction
; set a value in the table at the
; given index
; #1 index
; #2 value
;================================
inst_CA_v3:
      lda  ($E4),y               ; read next byte of pattern
      tax
      iny
      lda  ($E4),y               ; read next byte of pattern
      sta  InstTableV3+0,x
      jmp  adjust3PatternV3

;================================
; Voice 3: NEXT instruction
; Execute the next cycle of the FOR
;================================
inst_CE_v3:
      ldx  $EB                   ; read stack index of voice 3
      dec  $71,x                 ; number of repeat for voice 3
      beq  endCycle3             ; stop to cycle if 0
      ldy  $61,x                 ; read low addr into stack of voice 3
      lda  $69,x                 ; read high addr into stack of voice 3
      sty  $E4                   ; change address
      sta  $E5
      jmp  readNextV3

endCycle3:
      inc  $EB                   ; inc stack index of voice 3
      tya
      jmp  adjustPatternV3

;================================
; Voice 3: RTS instruction
; return to the stored location
; in the stack
;================================
inst_C0_v3:
      ldy  $EB                    ; read stack index of voice 3
      cpy  #$07
```

```
      beq  stop_v3
      inc  $EB                    ; inc stack index of voice 3
      ldx  $61,y                  ; read low address
      lda  $0069,y                ; read high address
      stx  $E4                    ; change address
      sta  $E5
      jmp  readNextV3

stop_v3:
      dec  playMusicV3            ; stop music
      rts

;================================
; Voice 3: INSTR instruction
; set 5 instrument parameters
; from the given location
; #1 low address
; #2 high address
;================================
inst_D4_v3:
      lda  ($E4),y               ; read next byte of pattern: low address
      sta  $EF
      iny
      lda  ($E4),y               ; read next byte of pattern: high address
      sta  $F0
      ldy  #$04
loopD4v3:
      lda  ($EF),y               ; read bytes from given address
      sta  InstTableV3+24,y      ; copy to instrument voice location
      dey
      bpl  loopD4v3
      jmp  adjust3PatternV3


;================================
; Make the dinamic filter effect
; If the filter flag has bit 1=1,
; during initial delay it is
; performed a cycle4 adding
;================================
makeFilterEff:
      lda  CurFilterTable+13     ; read filter flag effect
      beq  exitMFE               ; no filter if 0
      ldx  ActFilterTable+0      ; read actual filter low value (8 bit)
      ldy  ActFilterTable+1      ; read actual filter high value (3 bit)
      clc
      lda  CurFilterTable+12     ; read initial filter delay
      beq  filterAdd1
      dec  CurFilterTable+12     ; dec filter delay
      lda  CurFilterTable+13     ; read filter flag effect
      and  #$02
      bne  makeFilterAdd4
exitMFE:
      rts


;================================
; Filter add cycle 1
;================================
filterAdd:
      clc
filterAdd1:
      lda  ActFilterTable+2      ; actual filter cycle 1
      beq  filterAdd2            ; jump if 0
      dec  ActFilterTable+2      ; dec actual filter cycle 4
      txa
      adc  CurFilterTable+0      ; add filter low value 1
      tax
      tya
      adc  CurFilterTable+1      ; add filter high value 1
      jmp  changeFilter

;================================
; Filter add cycle 2
;================================
filterAdd2:
      lda  ActFilterTable+3      ; actual filter cycle 2
      beq  filterAdd3            ; jump if 0
      dec  ActFilterTable+3      ; dec actual filter cycle 2
      txa
      adc  CurFilterTable+2      ; add filter low value 2
      tax
      tya
      adc  CurFilterTable+3      ; add filter high value 2
      jmp  changeFilter

;================================
; Filter add cycle 3
;================================
filterAdd3:
```

```
      lda  ActFilterTable+4       ; actual filter cycle 3
      beq  filterAdd4             ; jump if zero
      dec  ActFilterTable+4       ; dec actual filter cycle 3
      txa
      adc  CurFilterTable+4       ; add filter low value 3
      tax
      tya
      adc  CurFilterTable+5       ; add filter high value 3
      jmp  changeFilter

;=================================
; filter add cycle 4
;=================================
filterAdd4:
      lda  ActFilterTable+5       ; actual filter cycle 4
      beq  resetFilter            ; jump if zero
      dec  ActFilterTable+5       ; dec actual filter cycle 4

makeFilterAdd4:
      txa
      adc  CurFilterTable+6       ; add filter low value 4
      tax
      tya
      adc  CurFilterTable+7       ; add filter high value 4

changeFilter:
      tay
changeRFilter:
      stx  ActFilterTable+0       ; store actual filter low value (8 bit)
      sty  ActFilterTable+1       ; store actual filter high value (3 bit)
changeNoSFilter:                  ; change and not store
      txa
      and  #$07
      sta  $D415                  ; Filter cut frequency: lo byte (bit 2-0)
      tya                         ; shift high bits into filter register high
      stx  tmpCutF
      lsr
      ror  tmpCutF
      lsr
      ror  tmpCutF
      lsr
      lda  tmpCutF
      ror
      sta  $D416                  ; Filter cut frequency: hi byte
      rts

;=================================
; Reset Filter
; Flag meanings:
; xyyy yyyyz
; x=1 -> reload the filter cycle with filter value of istrument table again
; z=1 -> reload the filter cycle but use the actual filter value
; y=1 -> continue with actual filter, no more cycle
;=================================
resetFilter:
      lda  CurFilterTable+13      ; read filter effect flag
      and  #$81
      beq  changeRFilter
      bpl  useActualF

      jsr  reloadFilterCycle
      jmp  filterAdd

useActualF:
      jsr  reloadFCycle
      jmp  filterAdd


;=================================
; copy main filter table to current
; one and reload the cycle
;=================================
copyMainCurrentFilter:
      ldx  #$07
loopCopy:
      lda  MainFilterTable+0,x    ; copy main filter table part 1
      sta  CurFilterTable+0,x     ; to current filter table part 1
      lda  MainFilterTable+8,x    ; copy main filter table part 2
      sta  CurFilterTable+8,x     ; to current filter table part 2
      dex
      bpl  loopCopy
      jsr  reloadFilterCycle
      jmp  changeNoSFilter

;=================================
; Execute the pattern of note
; voice 1
;=================================
executePatternV1:
      lda  playMusicV1            ; play music flag voice 1
```

```
      beq  no_execv1             ; exit if no music
      dec  $E6                   ; dec actual duration of note voice 1
      beq  readNextV1
no_execv1:
      rts

fixHighV1:
      inc  $E1
      bne  readNextV1

adjust3PatternV1:                ; adjust the pattern index with 3
      lda  #$03

adjustPatternV1:                 ; adjust the pattern index with A reg.
      clc
      adc  $E0
      sta  $E0
      bcs  fixHighV1

;==================================
; read next byte from pattern of voice 1
;==================================
readNextV1:
      ldy  #$00
      lda  ($E0),y               ; read next byte of pattern
      cmp  #$C0                  ; is an instruction?
      bcc  isNoteV1
      iny
      adc  #$3F                  ; calcolate right offset to the inst. table
      sta  addressV1+1
addressV1:
      jmp  (InstrVoice1)         ; execute instruction read from the pattern

jmpSetDurationV1:
      jmp  setDurationV1

isNoteV1:
      sta  $FF                   ; store the read byte: note to play
      cmp  #$60                  ; <60 means use custom durations
      bcc  alreadyReducedV1
      sbc  #$60                  ; reduce to 0..5F

alreadyReducedV1:
      cmp  #$5F                  ; 5Fh means rest note
      beq  jmpSetDurationV1
      adc  $EC                   ; haftone to transpose
      tax
      lda  #$08
      sta  $D404                 ; Voice 1: Control registers
      lda  voiceNumber           ; voice number (0,1,2) on where apply filter
      bne  continueInitV1        ; skip if not 0

      stx  $EF
      jsr  copyMainCurrentFilter
      ldx  $EF

continueInitV1:
      ldy  lowFreq,x             ; read freq high from table
      lda  highFreq,x            ; read freq low from table
      sta  CurInstTableV1+24     ; store current freq low voice 1
      sty  CurInstTableV1+25     ; store current freq high voice 1
      sta  $D400                 ; Voice 1: Frequency control (lo byte)
      sty  $D401                 ; Voice 1: Frequency control (hi byte)

      ldx  InstTableV1+22        ; read wave low voice 1
      ldy  InstTableV1+23        ; read wave high voice 1
      stx  $D402                 ; Voice 1: Wave form pulsation amplitude (lo byte)
      sty  $D403                 ; Voice 1: Wave form pulsation amplitude (hi byte)

      lda  InstTableV1+25        ; read AD voice 1
      sta  $D405                 ; Generator 1: Attack/Decay
      lda  InstTableV1+26        ; read SR voice 1
      sta  $D406                 ; Generator 1: Sustain/Release

      lda  InstTableV1+24        ; read control registers voice 1
      sta  CurInstTableV1+26     ; store current control registers
      and  #$F7                  ; set test bit to 0
      sta  $D404                 ; Voice 1: Control registers

      lda  InstTableV1+17        ; read wave effect flag voice 1
      sta  CurInstTableV1+17     ; store current wave effect flag voice 1
      beq  freqInitV1            ; skip other wave init if 0

      stx  CurInstTableV1+22     ; store current wave low voice 1
      sty  CurInstTableV1+23     ; store current wave high voice 1
      stx  $F1                   ; wave low voice 1
      sty  $F2                   ; wave high voice 1

      lda  InstTableV1+21        ; read wave high add 2 voice 1
      sta  CurInstTableV1+21     ; store current wave high add 2 voice 1
```

```
      lda  InstTableV1+20         ; read wave low add 2 voice 1
      sta  CurInstTableV1+20      ; store current wave low add 2 voice 1

      lda  InstTableV1+19         ; read wave high add 1 voice 1
      sta  CurInstTableV1+19      ; store current high add 1 voice 1

      lda  InstTableV1+18         ; read wave low add 1 voice 1
      sta  CurInstTableV1+18      ; store current wave low add 1 voice 1

      lda  InstTableV1+16         ; read wave delay initial voice 1
      sta  CurInstTableV1+16      ; store current wave initial delay voice 1

      ldx  InstTableV1+14         ; read wave cycle 1 voice 1
      ldy  InstTableV1+15         ; read wave cycle 2 voice 1
      stx  CurInstTableV1+14      ; store current (to reload) wave cycle 1 voice 1
      stx  CurInstTableV1+33      ; store current wave cycle 1 voice 1
      sty  CurInstTableV1+15      ; store current (to reload) wave cycle 2 voice 1
      sty  CurInstTableV1+34      ; store current wave cycle 2 voice 1

freqInitV1:
      lda  InstTableV1+13         ; read freq. flag effect voice 1
      sta  CurInstTableV1+13      ; store current freq. flag effect voice 1
      beq  skipFreqSetV1

      ldx  InstTableV1+12         ; read freq delay initial voice 1
      stx  CurInstTableV1+12      ; store current freq. initial delay voice 1

      ldx  InstTableV1+11         ; read freq cycle 4 voice 1
      stx  CurInstTableV1+11      ; store current (to reload) freq cycle 4 voice 1

      ldx  InstTableV1+10         ; read freq cycle 3 voice 1
      stx  CurInstTableV1+10      ; store current (to reload) freq cycle 3 voice 1

      ldx  InstTableV1+9          ; read freq cycle 2 voice 1
      stx  CurInstTableV1+9       ; store current (to reload) freq cycle 2 voice 1

      ldx  InstTableV1+8          ; read freq cycle 1 voice 1
      stx  CurInstTableV1+8       ; store current (to reload) freq cycle 1 voice 1

      ldx  InstTableV1+7          ; read freq high add 4 voice 1
      stx  CurInstTableV1+7       ; store current freq high add 4 voice 1

      ldx  InstTableV1+6          ; read freq low add 4 voice 1
      stx  CurInstTableV1+6       ; store current freq low add 4 voice 1

      ldx  InstTableV1+5          ; read freq high add 3 voice 1
      stx  CurInstTableV1+5       ; store current freq high add 3 voice 1

      ldx  InstTableV1+4          ; read freq low add 3 voice 1
      stx  CurInstTableV1+4       ; store current freq low add 3 voice 1

      ldx  InstTableV1+3          ; read freq high add 2 voice 1
      stx  CurInstTableV1+3       ; store current freq high add 2 voice 1

      ldx  InstTableV1+2          ; read freq low add 2 voice 1
      stx  CurInstTableV1+2       ; store current freq low add 2 voice 1

      ldx  InstTableV1+1          ; read freq high add 1 voice 1
      stx  CurInstTableV1+1       ; store current freq high add 1 voice 1

      ldx  InstTableV1+0          ; read freq low add 1 voice 1
      stx  CurInstTableV1+0       ; store current freq low add 1 voice 1
      and  #$08
      beq  reloadFCV1

      lda  $FF                    ; read store note to play
      cmp  #$60
      bcc  alreadyReduced2V1
      sbc  #$5F

alreadyReduced2V1:
      adc  $EC                    ; half tone to transpose
      sta  CurInstTableV1+10      ; store current (to reload) freq cycle 3 voice 1
      bne  skipReloadFCV1

reloadFCV1:
      jsr  reloadFreqCycleV1

skipReloadFCV1:
skipFreqSetV1:
      ldx  InstTableV1+27
      ldy  InstTableV1+28
      stx  CurInstTableV1+27
      sty  CurInstTableV1+28

setDurationV1:
      ldy  #$01
      lda  ($E0),y                ; read next byte of pattern: duration index voice 1
      ldx  $FF                    ; read stored note to play
```

```
      cpx  #$60                  ; <60h means custom durations
      bcs  skipDurTableV1
      tax
      lda  durationTable-1,x     ; read duration from table

skipDurTableV1:
      sta  $E6                   ; store duration of note voice 1
      lda  #$02
      clc
      adc  $E0
      sta  $E0                   ; update pointer for next note
      bcs  fixHigh2V1
      rts

fixHigh2V1:
      inc  $E1
      rts

;================================
; Execute the pattern of note
; voice 2
;================================
executePatternV2:
      lda  playMusicV2           ; play music flag voice 2
      beq  no_execv2             ; exit if no music
      dec  $E7                   ; dec actual note duration voice 2
      beq  readNextV2
no_execv2
      rts

fixHighV2:
      inc  $E3
      bne  readNextV2

adjust3PatternV2:                ; adjust the pattern index with 3
      lda  #$03

adjustPatternV2:                 ; adjust the pattern index with A reg.
      clc
      adc  $E2
      sta  $E2
      bcs  fixHighV2

;================================
; read next byte from pattern of voice 2
;================================
readNextV2:
      ldy  #$00
      lda  ($E2),y               ; read next byte of pattern
      cmp  #$C0                  ; is an instruction?
      bcc  isNoteV2
      iny
      adc  #$71                  ; calcolate right offset to the inst. table
      sta  addressV2+1
addressV2:
      jmp  (InstrVoice2)         ; execute instruction read from the pattern

jmpSetDurationV2:
      jmp  setDurationV2

isNoteV2:
      sta  $FF                   ; store the read byte: note to play
      cmp  #$60                  ; <60 means use custom durations
      bcc  alreadyReducedV2
      sbc  #$60                  ; reduce to 0..5F

alreadyReducedV2:
      cmp  #$5F                  ; 5Fh means rest note
      beq  jmpSetDurationV2
      adc  $ED                   ; haftone to transpose
      tax
      lda  #$08
      sta  $D40B                 ; Voice 2: Control registers
      lda  voiceNumber           ; voice number (0,1,2) on where apply filter
      cmp  #$01
      bne  continueInitV2        ; skip if not 1

      stx  $EF
      jsr  copyMainCurrentFilter
      ldx  $EF

continueInitV2:
      ldy  lowFreq,x             ; read freq high from table
      lda  highFreq,x            ; read freq low from table
      sta  CurInstTableV2+24     ; store current freq low voice 2
      sty  CurInstTableV2+25     ; store current freq high voice 2
      sta  $D407                 ; Voice 2: Frequency control (lo byte)
      sty  $D408                 ; Voice 2: Frequency control (hi byte)

      ldx  InstTableV2+22        ; read wave low voice 2
```

```
        ldy  InstTableV2+23         ; read wave high voice 2
        stx  $D409                  ; Voice 2: Wave form pulsation amplitude (lo byte)
        sty  $D40A                  ; Voice 2: Wave form pulsation amplitude (hi byte)

        lda  InstTableV2+25         ; read AD voice 2
        sta  $D40C                  ; Generator 2: Attack/Decay
        lda  InstTableV2+26         ; read SR voice 1
        sta  $D40D                  ; Generator 2: Sustain/Release

        lda  InstTableV2+24         ; read control registers voice 2
        sta  CurInstTableV2+26      ; store current control registers voice 2
        and  #$F7                   ; set test bit to 0
        sta  $D40B                  ; Voice 2: Control registers

        lda  InstTableV2+17         ; read wave effect flag voice 2
        sta  CurInstTableV2+17      ; store current wave effect flag voice 2
        beq  freqInitV2             ; skip other wave init if 0

        stx  CurInstTableV2+22      ; store current wave low voice 2
        sty  CurInstTableV2+23      ; store current wave high voice 2
        stx  $F3                    ; wave low voice 2
        sty  $F4                    ; wave high voice 2

        lda  InstTableV2+21         ; read wave high add 2 voice 2
        sta  CurInstTableV2+21      ; store current wave high add 2 voice 2

        ldx  InstTableV2+20         ; read wave low add 2 voice 2
        stx  CurInstTableV2+20      ; store current wave low add 2 voice 2

        lda  InstTableV2+19         ; read wave high add 1 voice 2
        sta  CurInstTableV2+19      ; store current high add 1 voice 2

        lda  InstTableV2+18         ; read wave low add 1 voice 2
        sta  CurInstTableV2+18      ; store current wave low add 1 voice 2

        lda  InstTableV2+16         ; read wave delay initial voice 2
        sta  CurInstTableV2+16      ; store current wave initial delay voice 2

        ldx  InstTableV2+14         ; read wave cycle 1 voice 2
        ldy  InstTableV2+15         ; read wave cycle 2 voice 2
        stx  CurInstTableV2+14      ; store current (to reload) wave cycle 1 voice 2
        stx  CurInstTableV2+33      ; store current wave cycle 1 voice 2
        sty  CurInstTableV2+15      ; store current (to reload) wave cycle 2 voice 2
        sty  CurInstTableV2+34      ; store current wave cycle 1 voice 2

freqInitV2:
        lda  InstTableV2+13         ; read freq. flag effect voice 2
        sta  CurInstTableV2+13      ; store current freq. flag effect voice 2
        beq  skipFreqSetV2

        ldx  InstTableV2+12         ; read freq delay initial voice 2
        stx  CurInstTableV2+12      ; store current freq. initial delay voice 2

        ldx  InstTableV2+11         ; read freq cycle 4 voice 2
        stx  CurInstTableV2+11      ; store current (to reload) freq cycle 4 voice 2

        ldx  InstTableV2+10         ; read freq cycle 3 voice 2
        stx  CurInstTableV2+10      ; store current (to reload) freq cycle 3 voice 2

        ldx  InstTableV2+9          ; read freq cycle 2 voice 2
        stx  CurInstTableV2+9       ; store current (to reload) freq cycle 2 voice 2

        ldx  InstTableV2+8          ; read freq cycle 1 voice 2
        stx  CurInstTableV2+8       ; store current (to reload) freq cycle 1 voice 2

        ldx  InstTableV2+7          ; read freq high add 4 voice 2
        stx  CurInstTableV2+7       ; store current freq high add 4 voice 2

        ldx  InstTableV2+6          ; read freq low add 4 voice 2
        stx  CurInstTableV2+6       ; store current freq low add 4 voice 2

        ldx  InstTableV2+5          ; read freq high add 3 voice 2
        stx  CurInstTableV2+5       ; store current freq high add 3 voice 2

        ldx  InstTableV2+4          ; read freq low add 3 voice 2
        stx  CurInstTableV2+4       ; store current freq low add 3 voice 2

        ldx  InstTableV2+3          ; read freq high add 2 voice 2
        stx  CurInstTableV2+3       ; store current freq high add 2 voice 2

        ldx  InstTableV2+2          ; read freq low add 2 voice 2
        stx  CurInstTableV2+2       ; store current freq low add 2 voice 2

        ldx  InstTableV2+1          ; read freq high add 1 voice 2
        stx  CurInstTableV2+1       ; store current freq high add 1 voice 2

        ldx  InstTableV2+0          ; read freq low add 1 voice 2
        stx  CurInstTableV2+0       ; store current freq low add 1 voice 2
        and  #$08
        beq  reloadFCV2
```

```
      lda  $FF                    ; read store note to play
      cmp  #$60
      bcc  alreadyReduced2V2

      sbc  #$5F

alreadyReduced2V2:
      adc  $ED
      sta  CurInstTableV2+10      ; store current (to reload) freq cycle 3 voice 2
      bne  skipReloadFCV2

reloadFCV2:
      jsr  reloadFreqCycleV2

skipFreqSetV2:
skipReloadFCV2:
      ldx  InstTableV2+27
      ldy  InstTableV2+28
      stx  CurInstTableV2+27
      sty  CurInstTableV2+28

setDurationV2:
      ldy  #$01
      lda  ($E2),y                ; read next byte of pattern: duration index voice 2
      ldx  $FF                    ; read stored note to play
      cpx  #$60                   ; <60h means custom durations
      bcs  skipDurTableV2

      tax
      lda  durationTable-1,x      ; read duration from table

skipDurTableV2:
      sta  $E7                    ; store actual note duration voice 2
      lda  #$02
      clc
      adc  $E2
      sta  $E2                    ; update pointer for next note
      bcs  fixHigh2V2
      rts

fixHigh2V2:
      inc  $E3
      rts

;================================
; Execute the pattern of note
; voice 3
;================================
executePatternV3:
      lda  playMusicV3            ; play music flag voice 3
      beq  no_execv3              ; exit if no music
      dec  $E8                    ; dec actual note duration voice 3
      beq  readNextV3
no_execv3
      rts

fixHighV3:
      inc  $E5
      bne  readNextV3


adjust3PatternV3:                 ; adjust the pattern index with 3
      lda  #$03

adjustPatternV3:                  ; adjust the pattern index with A reg.
      clc
      adc  $E4
      sta  $E4
      bcs  fixHighV3

;================================
; read next byte from pattern of voice 3
;================================
readNextV3:
      ldy  #$00
      lda  ($E4),y                ; read next byte of pattern
      cmp  #$C0                   ; is an instruction
      bcc  isNoteV3
      iny
      adc  #$A3                   ; calcolate right offset to the inst. table
      sta  addressV3+1
addressV3:
      jmp  (InstrVoice3)          ; execute instruction read from the pattern

jmpSetDurationV3:
      jmp  setDurationV3

isNoteV3:
      sta  $FF                    ; store the read byte: note to play
```

```
      cmp  #$60                  ; <60 means use custom durations
      bcc  alreadyReducedV3
      sbc  #$60                  ; reduce to 0..5F

alreadyReducedV3:
      cmp  #$5F                  ; 5Fh means rest note
      beq  jmpSetDurationV3

      cmp  #$50
      beq  noTransposeV3

      adc  $EE                   ; haftone to transpose

noTransposeV3:
      tax
      lda  #$08
      sta  $D412                 ; Voice 3: Control registers
      lda  voiceNumber           ; voice number (0,1,2) on where apply filter
      cmp  #$02
      bne  continueInitV3        ; skip if not 2

      stx  $EF
      jsr  copyMainCurrentFilter
      ldx  $EF

continueInitV3:
      ldy  lowFreq,x             ; read freq high from table
      lda  highFreq,x            ; read freq low from table
      sta  CurInstTableV3+24     ; store current freq low voice 3
      sty  CurInstTableV3+25     ; store current freq high voice 3
      sta  $D40E                 ; Voice 3: Frequency control (lo byte)
      sty  $D40F                 ; Voice 3: Frequency control (hi byte)

      ldx  InstTableV3+22        ; read wave low voice 3
      ldy  InstTableV3+23        ; read wave high voice 3
      stx  $D410                 ; Voice 3: Wave form pulsation amplitude (lo byte)
      sty  $D411                 ; Voice 3: Wave form pulsation amplitude (hi byte)

      lda  InstTableV3+25        ; read AD voice 3
      sta  $D413                 ; Generator 3: Attack/Decay
      lda  InstTableV3+26        ; read SR voice 3
      sta  $D414                 ; Generator 3: Sustain/Release

      lda  InstTableV3+24        ; read control registers voice 3
      sta  CurInstTableV3+26     ; store current control register
      and  #$F7                  ; set test bit to 0
      sta  $D412                 ; Voice 3: Control registers

      lda  InstTableV3+17        ; read wave effect flag voice 3
      sta  CurInstTableV3+17     ; store current wave effect flag voice 3
      beq  freqInitV3            ; skip other wave init if 0

      stx  CurInstTableV3+22     ; store current wave low voice 3
      sty  CurInstTableV3+23     ; store current wave high voice 3
      stx  $F5                   ; wave low voice 3
      sty  $F6                   ; wave high voice 3

      lda  InstTableV3+21        ; read wave high add 2 voice 3
      sta  CurInstTableV3+21     ; store current wave high add 2 voice 3

      lda  InstTableV3+20        ; read wave low add 2 voice 3
      sta  CurInstTableV3+20     ; store current wave low add 2 voice 3

      lda  InstTableV3+19        ; read wave high add 1 voice 3
      sta  CurInstTableV3+19     ; store current high add 1 voice 3

      lda  InstTableV3+18        ; read wave low add 1 voice 3
      sta  CurInstTableV3+18     ; store current wave low add 1 voice 3

      lda  InstTableV3+16        ; read wave delay initial voice 3
      sta  CurInstTableV3+16     ; store current wave initial delay voice 3

      ldx  InstTableV3+14        ; read wave cycle 1 voice 3
      ldy  InstTableV3+15        ; read wave cycle 2 voice 3
      stx  CurInstTableV3+14     ; store current (to reload) wave cycle 1 voice 3
      stx  CurInstTableV3+33     ; store current wave cycle 1 voice 3
      sty  CurInstTableV3+15     ; store current (to reload) wave cycle 2 voice 3
      sty  CurInstTableV3+34     ; store current wave cycle 1 voice 3

freqInitV3:
      lda  InstTableV3+13        ; read freq. flag effect voice 3
      sta  CurInstTableV3+13     ; store current freq. flag effect voice 3
      beq  skipFreqSetV3

      ldx  InstTableV3+12        ; read freq delay initial voice 3
      stx  CurInstTableV3+12     ; store current freq. initial delay voice 3

      ldx  InstTableV3+11        ; read freq cycle 4 voice 3
      stx  CurInstTableV3+11     ; store current (to reload) freq cycle 4 voice 3
```

```
      ldx  InstTableV3+10         ; read freq cycle 3 voice 3
      stx  CurInstTableV3+10      ; store current (to reload) freq cycle 3 voice 3

      ldx  InstTableV3+9          ; read freq cycle 2 voice 3
      stx  CurInstTableV3+9       ; store current (to reload) freq cycle 2 voice 3

      ldx  InstTableV3+8          ; read freq cycle 1 voice 3
      stx  CurInstTableV3+8       ; store current (to reload) freq cycle 1 voice 3

      ldx  InstTableV3+7          ; read freq high add 4 voice 3
      stx  CurInstTableV3+7       ; store current freq high add 4 voice 3

      ldx  InstTableV3+6          ; read freq low add 4 voice 3
      stx  CurInstTableV3+6       ; store current freq low add 4 voice 3

      ldx  InstTableV3+5          ; read freq high add 3 voice 3
      stx  CurInstTableV3+5       ; store current freq high add 3 voice 3

      ldx  InstTableV3+4          ; read freq low add 3 voice 3
      stx  CurInstTableV3+4       ; store current freq low add 3 voice 3

      ldx  InstTableV3+3          ; read freq high add 2 voice 3
      stx  CurInstTableV3+3       ; store current freq high add 2 voice 3

      ldx  InstTableV3+2          ; read freq low add 2 voice 3
      stx  CurInstTableV3+2       ; store current freq low add 2 voice 3

      ldx  InstTableV3+1          ; read freq high add 1 voice 3
      stx  CurInstTableV3+1       ; store current freq high add 1 voice 3

      ldx  InstTableV3+0          ; read freq low add 1 voice 3
      stx  CurInstTableV3+0       ; store current freq low add 1 voice 3
      and  #$08
      beq  reloadFCV3

      lda  $FF                    ; read store note to play
      cmp  #$60
      bcc  alreadyReduced2V3
      sbc  #$5F

alreadyReduced2V3:
      adc  $EE
      sta  CurInstTableV3+10      ; store current (to reload) freq cycle 3 voice 3
      bne  skipReloadFCV3

reloadFCV3:
      jsr  reloadFreqCycleV3

skipFreqSetV3:
skipReloadFCV3:
      ldx  InstTableV3+27
      ldy  InstTableV3+28
      stx  CurInstTableV3+27
      sty  CurInstTableV3+28

setDurationV3:
      ldy  #$01
      lda  ($E4),y                ; read next byte of pattern: duration index voice 3
      ldx  $FF                    ; read stored note to play
      cpx  #$60                   ; <60h means custom durations
      bcs  skipDurTableV3

      tax
      lda  durationTable-1,x      ; read duration from table

skipDurTableV3:
      sta  $E8                    ; store actual note duration voice 3

      lda  #$02
      clc
      adc  $E4
      sta  $E4                    ; update pointer for next note
      bcs  fixHigh2V3
      rts

fixHigh2V3:
      inc  $E5

exitRTS:
      rts

;================================
; Make the timbre of voice 1
;================================
makeTimbreV1:
      ldx  CurInstTableV1+28
      beq  exitRTS                ; exit if 0

      lda  CurInstTableV1+26      ; read current Control registers voice 1
      and  #$08                   ; test bit
```

```
      beq  noTestB1

      lda  $E6                  ; read duration of note voice 1
      cmp  CurInstTableV1+27
      bcs  testIfWaveEffV1

      lda  #$00
      sta  CurInstTableV1+27

      lda  CurInstTableV1+26    ; read current control registers voice 1
      and  #$F6                 ; gate and test bit to 0
      sta  CurInstTableV1+26    ; store current control registers voice 1
      bne  outControl1

noTestB1:
      lda  CurInstTableV1+27
      bne  r3066

      ldy  CurInstTableV1+28
      iny
      beq  testIfWaveEffV1
      dec  CurInstTableV1+28
      bne  testIfWaveEffV1

; hard restart
      ldx  #$06
loopSid1:
      sta  $D400,x              ; Voice 1: Frequency control (lo byte)
      dex
      bpl  loopSid1

testFilterV1:
      cmp  voiceNumber          ; voice number (0,1,2) on where apply filter
      bne  exitRTS              ; exit if not equal

      inx
      stx  CurFilterTable+13    ; filter effect flag
      rts

r3066:
      ldy  CurInstTableV1+27
r3069:
      iny
      beq  testIfWaveEffV1
      dec  CurInstTableV1+27
      bne  testIfWaveEffV1

      lda  CurInstTableV1+26    ; read current control registers voice 1
      and  #$F6                 ; gate and test bit to 0

outControl1:
      sta  $D404                ; Voice 1: Control registers

testIfWaveEffV1:
      lda  CurInstTableV1+17    ; read current wave effect flag voice 1
      beq  testIfFreqEffV1      ; no wave if 0

      lda  CurInstTableV1+16    ; read current wave initial delay voice 1
      beq  waveAdd1V1
      dec  CurInstTableV1+16    ; dec delay voice 1
      jmp  testIfFreqEffV1

;================================
; wave add cycle 1 voice 1
;================================
waveAdd1V1:
      clc
      ldx  $F1                  ; wave low v1 voice 1
      ldy  $F2                  ; wave high v1 voice 1
      lda  CurInstTableV1+33    ; wave cycle 1 voice 1
      beq  waveAdd2V1           ; goto cycle 2 if 0
      dec  CurInstTableV1+33    ; decrement wave cycle 1 voice 1
      txa
      adc  CurInstTableV1+18    ; add current low wave value for cycle 1 voice 1
      tax
      tya
      adc  CurInstTableV1+19    ; add current high wave value for cycle 1 voice 1
      tay
      jmp  changeWave1

;================================
; wave add cycle 2 voice 1
;================================
waveAdd2V1:
      lda  CurInstTableV1+34    ; wave cycle 2 voice 1
      beq  resetWave1
      dec  CurInstTableV1+34    ; decrement wave cycle 2 voice 1
      txa
      adc  CurInstTableV1+20    ; add current low value for cycle 2 voice 1
      tax
```

```
      tya
      adc  CurInstTableV1+21      ; add current high value for cycle 2 voice 1
      tay
      jmp  changeWave1


;=================================
; Reset Wave voice 1
; Flag meanings:
; xyyy yyyyz
; x=1 -> reload the wave cycle with wave value of istrument table again
; z=1 -> reload the wave cycle but use the actual wave value
; y=1 -> continue with actual wave, no more cycle
;=================================
resetWave1:
      lda  CurInstTableV1+17      ; read current wave effect flag voice 1
      and  #$81
      beq  changeWave1            ; go to continue with actual wave
      bpl  useCycleActualV1

      jsr  reloadWaveCycleV1      ; reload wave cycle using wave in current ins. table
      jmp  waveAdd1V1

useCycleActualV1:
      jsr  reloadWCycleV1         ; reload wave cycle using wave in register
      jmp  waveAdd1V1

;=================================
; change wave setting of voice 1
;=================================
changeWave1:
      stx  $F1                    ; wave low v1
      sty  $F2                    ; wave high v1
      stx  $D402                  ; Voice 1: Wave form pulsation amplitude (lo byte)
      sty  $D403                  ; Voice 1: Wave form pulsation amplitude (hi byte)

testIfFreqEffV1:
      lda  CurInstTableV1+13      ; read current freq. flag effect voice 1
      beq  exitCW1                ; exit if zero

      ldx  $F7                    ; low of freq.
      ldy  $F8                    ; high of freq.
      clc
      lda  CurInstTableV1+12      ; read current freq. initial delay voice 1
      beq  freqAdd1V1_
      dec  CurInstTableV1+12      ; dec current freq. initial delay voice 1
      lda  CurInstTableV1+13      ; read current freq. flag effect voice 1
      and  #$02
      bne  addF4V1
exitCW1:
      rts

;=================================
; freq add cycle1 voice 1
;=================================
freqAdd1V1:
      clc
freqAdd1V1_:
      lda  CurInstTableV1+29      ; read current freq cycle 1 voice 1
      beq  freqAdd2V1
      dec  CurInstTableV1+29      ; dec current freq cycle 1 voice 1
      txa
      adc  CurInstTableV1+0       ; add current low freq value for cycle 1 voice 1
      tax
      tya
      adc  CurInstTableV1+1       ; add current high freq value for cycle 1 voice 1
      jmp  change_freq1

;=================================
; freq add cycle2 voice 1
;=================================
freqAdd2V1:
      lda  CurInstTableV1+30      ; read current freq cycle 2 voice 1
      beq  freqAdd3V1
      dec  CurInstTableV1+30      ; dec current freq cycle 2 voice 1
      txa
      adc  CurInstTableV1+2       ; add current freq low add 2 voice 1
      tax
      tya
      adc  CurInstTableV1+3       ; add current freq high add 2 voice 1
      jmp  change_freq1

;=================================
; freq add cycle3 voice 1
;=================================
freqAdd3V1:
      lda  CurInstTableV1+31      ; read current freq cycle 3 voice 1
      beq  freqAdd4V1
      dec  CurInstTableV1+31      ; dec current freq cycle 3 voice 1
      txa
```

```
      adc  CurInstTableV1+4       ; add current freq low add 3 voice 1
      tax
      tya
      adc  CurInstTableV1+5       ; add current freq high add 3 voice 1
      jmp  change_freq1

;=================================
; freq add cycle4 voice 1
;=================================
freqAdd4V1:
      lda  CurInstTableV1+32      ; read current freq cycle 4 voice 1
      beq  resetFreqV1
      dec  CurInstTableV1+32      ; dec current freq cycle 4 voice 1

addF4V1:
      txa
      adc  CurInstTableV1+6       ; add current freq low add 4 voice 1
      tax
      tya
      adc  CurInstTableV1+7       ; add current freq high add 4 voice 1

;=================================
; change frequecy: X=lo, A=hi voice 1
;=================================
change_freq1:
      tay

useCurrF1:                        ; use current frequency
      stx  $D400                  ; Voice 1: Frequency control (lo byte)
      sty  $D401                  ; Voice 1: Frequency control (hi byte)
      stx  $F7
      sty  $F8

exitUseCurrF1:
      rts

;=================================
; Reset Frequency Voice 1
; Flag meandings:
; xyyy yyyyz
; x=1 -> reload the freq. cycle with freq. value of istrument table again
; z=1 -> reload the freq. cycle but use the actual freq value
; y=1 -> continue with actual freq., no more cycle
;=================================
resetFreqV1:
      lda  CurInstTableV1+13      ; read current freq. flag effect voice 1
      and  #$81
      beq  useCurrF1              ; go to continue with actual freq.
      bpl  useCycleFActualV1

      jsr  reloadFreqCycleV1
      jmp  freqAdd1V1

useCycleFActualV1:
      jsr  reloadFCycleV1
      jmp  freqAdd1V1

;=================================
; Make the timbre of voice 2
;=================================
makeTimbreV2:
      ldx  CurInstTableV2+28
      beq  exitUseCurrF1

      lda  CurInstTableV2+26      ; read current control registers voice 2
      and  #$08
      beq  noTestB2

      lda  $E7                    ; read actual note duration voice 2
      cmp  CurInstTableV2+27
      bcs  testIfWaveEffV2

      lda  #$00
      sta  CurInstTableV2+27

      lda  CurInstTableV2+26      ; read current control registers voice 2
      and  #$F6
      sta  CurInstTableV2+26      ; store current control registers voice 2
      bne  outControl2

noTestB2:
      lda  CurInstTableV2+27
      bne  r319F

      ldy  CurInstTableV2+28
      iny
      beq  testIfWaveEffV2
      dec  CurInstTableV2+28
      bne  testIfWaveEffV2
```

```
; hard restart
      ldx  #$06
loopSid2:
      sta  $D407,x               ; Voice 2: Frequency control (lo byte)
      dex
      bpl  loopSid2

testFilterV2:
      lda  #$01
      jmp  testFilterV1

r319F:
      ldy  CurInstTableV2+27
      iny
      beq  testIfWaveEffV2
      dec  CurInstTableV2+27
      bne  testIfWaveEffV2

      lda  CurInstTableV2+26     ; read current control registers voice 2
      and  #$F6                  ; gate and test bit to 0

outControl2:
      sta  $D40B                 ; Voice 2: Control registers

testIfWaveEffV2:
      lda  CurInstTableV2+17     ; read current wave effect flag voice 2
      beq  testIfFreqEffV2       ; no wave if 0

      lda  CurInstTableV2+16     ; read current wave initial delay voice 2
      beq  waveAdd1V2
      dec  CurInstTableV2+16     ; dec current wave initial delay voice 2
      jmp  testIfFreqEffV2

;==================================
; wave add cycle 1 voice 2
;==================================
waveAdd1V2:
      clc
      ldx  $F3                   ; wave low v1 voice 2
      ldy  $F4                   ; wave high v1 voice 2
      lda  CurInstTableV2+33     ; wave cycle 1 voice 2
      beq  waveAdd2V2            ; goto cycle 2 if 0
      dec  CurInstTableV2+33     ; decrement wave cycle 1 voice 2
      txa
      adc  CurInstTableV2+18     ; add current low wave value for cycle 1 voice 2
      tax
      tya
      adc  CurInstTableV2+19     ; add current high wave value for cycle 1 voice 2
      tay
      jmp  changeWave2

;==================================
; wave add cycle 2 voice 2
;==================================
waveAdd2V2:
      lda  CurInstTableV2+34     ; wave cycle 2 voice 2
      beq  resetWave2
      dec  CurInstTableV2+34     ; decrement wave cycle 2 voice 2
      txa
      adc  CurInstTableV2+20     ; add current low value for cycle 2 voice 2
      tax
      tya
      adc  CurInstTableV2+21     ; add current high value for cycle 2 voice 2
      tay
      jmp  changeWave2

;==================================
; Reset Wave voice 2
; Flag meanings:
; xyyy yyyyz
; x=1 -> reload the wave cycle with wave value of istrument table again
; z=1 -> reload the wave cycle but use the actual wave value
; y=1 -> continue with actual wave, no more cycle
;==================================
resetWave2:
      lda  CurInstTableV2+17     ; read current wave effect flag voice 2
      and  #$81
      beq  changeWave2           ; go to continue with actual wave
      bpl  useCycleActualV2
      jsr  reloadWaveCycleV2     ; reload wave cycle using wave in current ins. table
      jmp  waveAdd1V2

useCycleActualV2:
      jsr  reloadWCycleV2        ; reload wave cycle using wave in register
      jmp  waveAdd1V2

;==================================
; change wave setting of voice 2
;==================================
changeWave2:
```

```
      stx  $F3                  ; wave low v2
      sty  $F4                  ; wave high v2
      stx  $D409                ; Voice 2: Wave form pulsation amplitude (lo byte)
      sty  $D40A                ; Voice 2: Wave form pulsation amplitude (hi byte)

testIfFreqEffV2:
      lda  CurInstTableV2+13    ; read current freq. flag effect voice 2
      beq  exitCW2              ; exit if zero

      and  #$08
      bne  r322E

      ldx  $F9
      ldy  $FA
      clc
      lda  CurInstTableV2+12    ; current freq. initial delay voice 2
      beq  freqAdd1V2_
      dec  CurInstTableV2+12    ; dec current freq. initial delay voice 2
      lda  CurInstTableV2+13    ; read current freq. flag effect voice 2
      and  #$02
      bne  addF4V2
exitCW2:
      rts

r322E:
      dec  CurInstTableV2+4     ; dec current freq low add 3 voice 2
      bne  exitCW2

      ldy  CurInstTableV2+6     ; read current freq low add 4 voice 2
      sty  CurInstTableV2+4     ; store current freq low add 3 voice 2
      ldy  CurInstTableV2+12    ; read current freq. initial delay voice 2
      bpl  r3241
      ldy  CurInstTableV2+11    ; read current (to reload) freq cycle 4 voice 2

r3241
      ldx  CurInstTableV2+8     ; read current (to reload) freq cycle 1 voice 2
      stx  $EF
      ldx  CurInstTableV2+9     ; read current (to reload) freq cycle 2 voice 2
      stx  $F0
      lda  CurInstTableV2+10    ; read current (to reload) freq cycle 3 voice 2
      clc
      adc  ($EF),y
      dey
      sty  CurInstTableV2+12    ; store current freq. initial delay voice 2
      tay
      ldx  highFreq,y           ; read freq low from table
      lda  lowFreq,y            ; read freq high from table
      stx  $D407                ; Voice 2: Frequency control (lo byte)
      sta  $D408                ; Voice 2: Frequency control (hi byte)
      rts

;==================================
; freq add cycle1 voice 2
;==================================
freqAdd1V2:
      clc
freqAdd1V2_:
      lda  CurInstTableV2+29    ; read current freq cycle 1 voice 2
      beq  freqAdd2V2
      dec  CurInstTableV2+29    ; dec current freq cycle 1 voice 2
      txa
      adc  CurInstTableV2+0     ; add current low freq value for cycle 1 voice 2
      tax
      tya
      adc  CurInstTableV2+1     ; add current high freq value for cycle 1 voice 2
      jmp  change_freq2

;==================================
; freq add cycle2 voice 2
;==================================
freqAdd2V2:
      lda  CurInstTableV2+30    ; read current freq cycle 2 voice 2
      beq  freqAdd3V2
      dec  CurInstTableV2+30    ; dec current freq cycle 2 voice 2
      txa
      adc  CurInstTableV2+2     ; add current freq low add 2 voice 2
      tax
      tya
      adc  CurInstTableV2+3     ; add current freq high add 2 voice 2
      jmp  change_freq2

;==================================
; freq add cycle3 voice 2
;==================================
freqAdd3V2:
      lda  CurInstTableV2+31    ; read current freq cycle 3 voice 2
      beq  freqAdd4V2
      dec  CurInstTableV2+31    ; dec current freq cycle 3 voice 2
      txa
      adc  CurInstTableV2+4     ; add current freq low add 3 voice 2
```

```
      tax
      tya
      adc  CurInstTableV2+5       ; add current freq high add 3 voice 2
      jmp  change_freq2

;=================================
; freq add cycle4 voice 2
;=================================
freqAdd4V2:
      lda  CurInstTableV2+32      ; read current freq cycle 4 voice 2
      beq  resetFreqV2
      dec  CurInstTableV2+32      ; dec current freq cycle 4 voice 2

addF4V2:
      txa
      adc  CurInstTableV2+6       ; add current freq low add 4 voice 2
      tax
      tya
      adc  CurInstTableV2+7       ; add current freq high add 4 voice 2

;=================================
; change frequecy: X=lo, A=hi voice 2
;=================================
change_freq2:
      tay

useCurrF2:                        ; use current frequency
      stx  $D407                  ; Voice 2: Frequency control (lo byte)
      sty  $D408                  ; Voice 2: Frequency control (hi byte)
      stx  $F9                    ; freq. low voice 2
      sty  $FA                    ; freq. high voice 2
exitCF2:
      rts

;=================================
; Reset Frequency voice 2
; Flag meandings:
; xyyy yyyyz
; x=1 -> reload the freq. cycle with freq. value of istrument table again
; z=1 -> reload the freq. cycle but use the actual freq value
; y=1 -> continue with actual freq., no more cycle
;=================================
resetFreqV2:
      lda  CurInstTableV2+13      ; read current freq. flag effect voice 2
      and  #$81
      beq  useCurrF2
      bpl  useCycleFActualV2

      jsr  reloadFreqCycleV2
      jmp  freqAdd1V2

useCycleFActualV2:
      jsr  reloadFCycleV2
      jmp  freqAdd1V2

;=================================
; Make the timbre of the voice 3
;=================================
makeTimbreV3:
      ldx  CurInstTableV3+28
      beq  exitCF2

      lda  CurInstTableV3+26      ; read current control registers voice 3
      and  #$08
      beq  noTestB3
      lda  $E8                    ; read actual note duration (length) voice 3
      cmp  CurInstTableV3+27
      bcs  testIfWaveEffV3
      lda  #$00
      sta  CurInstTableV3+27
      lda  CurInstTableV3+26      ; read current control registers voice 3
      and  #$F6
      sta  CurInstTableV3+26      ; store current control registers voice 3
      bne  outControl3

noTestB3:
      lda  CurInstTableV3+27
      bne  r3311
      ldy  CurInstTableV3+28
      iny
      beq  testIfWaveEffV3
      dec  CurInstTableV3+28
      bne  testIfWaveEffV3

; hard restart
      ldx  #$06
loopSid3:
      sta  $D40E,x                ; Voice 3: Frequency control (lo byte)
      dex
      bpl  loopSid3
```

```
testFilterV3:
      lda  #$02
      jmp  testFilterV1

r3311:
      ldy  CurInstTableV3+27
      iny
      beq  testIfWaveEffV3
      dec  CurInstTableV3+27
      bne  testIfWaveEffV3

      lda  CurInstTableV3+26      ; read current control registers voice 3
      and  #$F6                   ; gate and test bit to 0

outControl3:
      sta  $D412                  ; Voice 3: Control registers

testIfWaveEffV3:
      lda  CurInstTableV3+17      ; read wave effect flag voice 3
      beq  testIfFreqEffV3
      lda  $FB
      ora  $FC
      beq  testIfFreqEffV3
      lda  CurInstTableV3+16      ; read current wave initial delay voice 3
      beq  waveAdd1V3
      dec  CurInstTableV3+16      ; dec current wave initial delay voice 3
      jmp  testIfFreqEffV3

;=================================
; wave add cycle 1 voice 3
;=================================
waveAdd1V3:
      clc
      ldx  $F5                    ; read wave low v3
      ldy  $F6                    ; read wave high v3
      lda  CurInstTableV3+33      ; read current wave cycle 1 voice 3
      beq  waveAdd2V3
      dec  CurInstTableV3+33      ; dec current wave cycle 1 voice 3
      txa
      adc  CurInstTableV3+18      ; add current wave low add 1 voice 3
      tax
      tya
      adc  CurInstTableV3+19      ; add current high add 1 voice 3
      tay
      jmp  changeWave3

;=================================
; wave add cycle 2 voice 3
;=================================
waveAdd2V3:
      lda  CurInstTableV3+34      ; read current wave cycle 2 voice 3
      beq  resetWave3
      dec  CurInstTableV3+34      ; dec current wave cycle 2 voice 3
      txa
      adc  CurInstTableV3+20      ; add current wave low add 2 voice 3
      tax
      tya
      adc  CurInstTableV3+21      ; add current wave high add 2 voice 3
      tay
      jmp  changeWave3

;=================================
; Reset Wave voice 3
; Flag meanings:
; xyyy yyyyz
; x=1 -> reload the wave cycle with wave value of istrument table again
; z=1 -> reload the wave cycle but use the actual wave value
; y=1 -> continue with actual wave, no more cycle
;=================================
resetWave3:
      lda  CurInstTableV3+17      ; wave effect flag voice 3
      and  #$81
      beq  changeWave3            ; go to continue with actual wave
      bpl  useCycleActualV3
      jsr  reloadWaveCycleV3      ; reload wave cycle using wave in current ins. table
      jmp  waveAdd1V3

useCycleActualV3:
      jsr  reloadWCycleV3         ; reload wave cycle using wave in register
      jmp  waveAdd1V3

;=================================
; change wave setting of voice 3
;=================================
changeWave3:
      stx  $F5                    ; store wave low v3
      sty  $F6                    ; store wave high v3
      stx  $D410                  ; Voice 3: Wave form pulsation amplitude (lo byte)
      sty  $D411                  ; Voice 3: Wave form pulsation amplitude (hi byte)
```

```
testIfFreqEffV3:
      lda  CurInstTableV3+13      ; read freq. effect flag voice 3
      beq  exitCW3                ; exit if zero

      lda  $FB                    ; read freq. low voice 3
      ora  $FC                    ; freq. high voice 3
      beq  exitCW3

      ldx  $FB
      ldy  $FC
      clc
      lda  CurInstTableV3+12      ; read current freq. initial delay voice 3
      beq  freqAdd1V3_
      dec  CurInstTableV3+12      ; dec current freq. initial delay voice 3
      lda  CurInstTableV3+13      ; read freq. effect flag voice 3
      and  #$02
      bne  addF4V3
exitCW3:
      rts

;=================================
; freq add cycle1 voice 3
;=================================
freqAdd1V3:
      clc
freqAdd1V3_:
      lda  CurInstTableV3+29      ; read current freq cycle 1 voice 3
      beq  freqAdd2V3
      dec  CurInstTableV3+29      ; dec current freq cycle 1 voice 3
      txa
      adc  CurInstTableV3+0       ; add current low freq value for cycle 1 voice 3
      tax
      tya
      adc  CurInstTableV3+1       ; add current high freq value for cycle 1 voice 3
      jmp  change_freq3

;=================================
; freq add cycle2 voice 3
;=================================
freqAdd2V3:
      lda  CurInstTableV3+30      ; read current freq cycle 2 voice 3
      beq  freqAdd3V3
      dec  CurInstTableV3+30      ; dec current freq cycle 2 voice 3
      txa
      adc  CurInstTableV3+2       ; add current freq low add 2 voice 3
      tax
      tya
      adc  CurInstTableV3+3       ; add current freq high add 2 voice 3
      jmp  change_freq3

;=================================
; freq add cycle3 voice 3
;=================================
freqAdd3V3:
      lda  CurInstTableV3+31      ; read current freq cycle 3 voice 3
      beq  freqAdd4V3
      dec  CurInstTableV3+31      ; dec current freq cycle 3 voice 3
      txa
      adc  CurInstTableV3+4       ; add current freq low add 3 voice 3
      tax
      tya
      adc  CurInstTableV3+5       ; add current freq high add 3 voice 3
      jmp  change_freq3

;=================================
; freq add cycle4 voice 3
;=================================
freqAdd4V3:
      lda  CurInstTableV3+32      ; read current freq cycle 4 voice 3
      beq  resetFreqV3
      dec  CurInstTableV3+32      ; dec current freq cycle 4 voice 3

addF4V3:
      txa
      adc  CurInstTableV3+6       ; add current freq low add 4 voice 3
      tax
      tya
      adc  CurInstTableV3+7       ; add current freq high add 4 voice 3

;=================================
; change frequecy: X=lo, A=hi voice 3
;=================================
change_freq3:
      tay

useCurrF3:                        ; use current frequency
      stx  $D40E                  ; Voice 3: Frequency control (lo byte)
      sty  $D40F                  ; Voice 3: Frequency control (hi byte)
      stx  $FB
```

```
      sty  $FC
      rts

;================================
; Reset Frequency voice 3
; Flag meandings:
; xyyy yyyyz
; x=1 -> reload the freq. cycle with freq. value of istrument table again
; z=1 -> reload the freq. cycle but use the actual freq value
; y=1 -> continue with actual freq., no more cycle
;================================
resetFreqV3:
      lda  CurInstTableV3+13      ; read current freq. flag effect voice 3
      and  #$81
      beq  useCurrF3
      bpl  useCycleFActualV3

      jsr  reloadFreqCycleV3
      jmp  freqAdd1V3

useCycleFActualV3:
      jsr  reloadFCycleV3
      jmp  freqAdd1V3


;;3417
      lda  playMusicV1            ; play music flag voice 1
      ora  playMusicV2            ; play music flag voice 2
      ora  playMusicV3            ; play music flag voice 3
      ora  CurInstTableV1+28
      ora  CurInstTableV2+28
      ora  CurInstTableV3+28
      rts

;================================
; Define the table with the track
; offset: last byte is the minime
; duration of a note
; The offset used by setTrack is
; calculated from the second byte
; of the table
;================================
trackTable:

tune3:
  .byte <tune3_voice1, >tune3_voice1
  .byte <tune3_voice2, >tune3_voice2
  .byte <tune3_voice3, >tune3_voice3
  .byte $09                         ; min note duration

tune4:
  .byte <tune4_voice1, >tune4_voice1
  .byte <tune4_voice2, >tune4_voice2
  .byte <tune4_voice3, >tune4_voice3
  .byte $0B                         ; min note duration

tune5:
  .byte <tune5_voice1, >tune5_voice1
  .byte <tune5_voice2, >tune5_voice2
  .byte <tune5_voice3, >tune4_voice3
  .byte $09                         ; min note duration

;343F
  .byte $DD, $DD
  .byte $DD, $DD
  .byte $DD, $DD
  .byte $DD

tune6:
  .byte <tune6_voice1, >tune6_voice1
  .byte <tune6_voice2, >tune6_voice2
  .byte <tune6_voice3, >tune6_voice3
  .byte $0D                         ; min note duration

tune2:
  .byte <tune2_voice1, >tune2_voice1
  .byte <tune2_voice2, >tune2_voice2
  .byte <tune2_voice3, >tune2_voice3
  .byte $03                         ; min note duration

tune7:
  .byte <tune7_voice1, >tune7_voice1
  .byte <tune7_voice2, >tune7_voice2
  .byte <tune7_voice3, >tune7_voice3
  .byte $0B                         ; min note duration

tune8:
  .byte <tune8_voice1, >tune8_voice1
  .byte <tune8_voice2, >tune8_voice2
  .byte <tune8_voice3, >tune8_voice3
```

```
  .byte $07                         ; min note duration

tune1:
  .byte <tune1_voice1, >tune1_voice1
  .byte <tune1_voice2, >tune1_voice2
  .byte <tune1_voice3, >tune1_voice3
  .byte $04                         ; min note duration


; Instrument 01 definition
ins01:
  .byte $14, $00                    ; freq low/high add 1
  .byte $EC, $FF                    ; freq low/high add 2
  .byte $14, $00                    ; freq low/high add 3
  .byte $00, $00                    ; freq low/high add 4
  .byte $03, $06                    ; freq cycle 1/2
  .byte $03, $00                    ; freq cycle 3/4
  .byte $1E, $05                    ; freq delay initial/freq effect flag

  .byte $32, $32                    ; wave cycle 1/2
  .byte $0A, $05                    ; wave delay initial/wave effect flag
  .byte $0A, $00                    ; wave low/high add 1
  .byte $F6, $FF                    ; wave low/high add 2
  .byte $00, $08                    ; wave low/high

  .byte $41                         ; control to rectangular
  .byte $14, $C8                    ; AD/SR
  .byte $FF, $FA

; Instrument 02 definition
ins02:
  .byte $23, $00                    ; freq low/high add 1
  .byte $DD, $FF                    ; freq low/high add 2
  .byte $23, $00                    ; freq low/high add 3
  .byte $00, $00                    ; freq low/high add 4
  .byte $03, $05                    ; freq cycle 1/2
  .byte $02, $00                    ; freq cycle 3/4
  .byte $0A, $05                    ; freq delay initial/freq effect flag

  .byte $00, $00                    ; wave not used
  .byte $00, $00
  .byte $00, $00
  .byte $00, $00
  .byte $00, $00

  .byte $19                         ; control to triangular + test bit
  .byte $A4, $F9                    ; AD SR
  .byte $14, $FE

tune1_voice1:
  .byte $D2, $1C, <ins01, >ins01  ; SETNI instr: set all the instrument table
  .byte $5F, $20
  .byte $1D, $20
  .byte $CC, $03                    ; FOR instr: repeat 3h times
  .byte $5F, $20
  .byte $CE                         ; NEXT instr
  .byte $1F, $20
  .byte $CC, $03                    ; FOR instr: repeat 3h times
  .byte $5F, $20
  .byte $CE                         ; NEXT instr
  .byte $DE, $0C, $80, $07        ; SET2CI instr: delay initial | freq effect flag
  .byte $DE, $06, $3C, $00        ; SET2CI instr: freq low|high add 4
  .byte $C6, $F4, <sub01, >sub01  ; JSRT: execute subroutine with transpose
  .byte $CA, $1A, $8D             ; SET instr: set SR of instrument
  .byte $C6, $00, <sub_1, >sub_1  ; JSRT: execute subroutine with transpose
  .byte $D6, $1B, $01             ; SETCI instr (??????????????????)
  .byte $D2, $1C, <ins02, >ins02  ; SETNI instr: set all the instrument table
  .byte $CC, $04                    ; FOR instr: repeat 4h times
  .byte $5F, $20
  .byte $CE                         ; NEXT instr
  .byte $CC, $04                    ; FOR inst: repeat 4h times
  .byte $43, $10
  .byte $42, $10
  .byte $40, $10
  .byte $3E, $10
  .byte $3C, $10
  .byte $3B, $10
  .byte $39, $10
  .byte $37, $10
  .byte $CE                         ; NEXT instr
  .byte $C0                         ; RTS instruction

; instrument 03 definition
ins03:
  .byte $14, $00                    ; freq low/high add 1
  .byte $EC, $FF                    ; freq low/high add 2
  .byte $14, $00                    ; freq low/high add 3
  .byte $00, $00                    ; freq low/high add 4
  .byte $03, $06                    ; freq cycle 1/2
  .byte $03, $00                    ; freq cycle 3/4
```

```
  .byte $0A, $05                ; freq delay initial/freq effect flag

  .byte $32, $32                ; wave cycle 1/2
  .byte $14, $05                ; wave delay initial/wave effect flag
  .byte $0A, $00                ; wave low/high add 1
  .byte $F6, $FF                ; wave low/high add 2
  .byte $00, $08                ; wave low/high
  .byte $41                     ; control to rectangular
  .byte $14, $C4                ; AD/SR
  .byte $FF, $05

tab07:
  .byte $0C, $00, $03, $09

tab0b:
  .byte $0C, $00, $03, $07

tab0f:
  .byte $0C, $00, $05, $0B

tab13:
  .byte $0C, $00, $05, $09

; instrument 04 definition
ins04:
  .byte $00, $00                ; freq low/high add 1
  .byte $00, $00                ; freq low/high add 2
  .byte $01, $00                ; freq low/high add 3
  .byte $01, $00                ; freq low/high add 4
  .byte $07, $35                ; freq cycle 1/2
  .byte $00, $03                ; freq cycle 3/4
  .byte $00, $08                ; freq delay initial/freq effect flag

  .byte $32, $32                ; wave cycle 1/2
  .byte $14, $00                ; wave delay initial/wave effect flag
  .byte $0A, $00                ; wave low/high add 1
  .byte $F6, $FF                ; wave low/high add 2
  .byte $00, $08                ; wave low/high
  .byte $41                     ; control: rectangular
  .byte $01, $F7                ; AD/SR
  .byte $04, $14

tune1_voice2:
  .byte $D2, $1C, <ins03, >ins03  ; SETNI instr: set all the instrument table
  .byte $5F, $20
  .byte $1D, $20
  .byte $CC, $03                ; FOR inst: repeat 3h times
  .byte $5F, $20
  .byte $CE                     ; NEXT instr
  .byte $1F, $20
  .byte $CC, $03                ; FOR inst: repeat 3h times
  .byte $5F, $20
  .byte $CE                     ; NEXT instr
  .byte $DE, $0C, $80, $07      ; SET2CI instr: set freq delay initial/freq effect flag
  .byte $DE, $06, $28, $00      ; SET2CI instr: set freq low/high add 4
  .byte $D2, $1C, <ins04, >ins04  ; SETNI instr: set all the instrument table
  .byte $5F, $20
  .byte $CC, $02                ; FOR inst: repeat 2h times (level 1)
  .byte $DC, $08, <tab07, >tab07  ;  SET2I instr: freq cycle 1/2
  .byte $CC, $10                ;  FOR inst: repeat 10h times (level 2)
  .byte $39, $02
  .byte $CE                     ;  NEXT (level 2)
  .byte $DC, $08, <tab0b, >tab0b  ;  SET2I instr: freq cycle 1/2
  .byte $CC, $10                ;  FOR inst: repeat 10h times (level 2)
  .byte $39, $02
  .byte $CE                     ;  NEXT (level 2)
  .byte $DC, $08, <tab0f, >tab0f  ;  SET2I instr: freq cycle 1/2
  .byte $CC, $10                ;  FOR inst: repeat 10h times (level 2)
  .byte $37, $02
  .byte $CE                     ;  NEXT (level 2)
  .byte $DC, $08, <tab13, >tab13  ;  SET2I instr: freq cycle 1/2
  .byte $CC, $10                ;  FOR inst: repeat 10h times (level 2)
  .byte $37, $02
  .byte $CE                     ;  NEXT (level 2)
  .byte $CE                     ; NEXT (level 1)
  .byte $CC, $07                ; FOR inst: repeat 7h times (level 1)
  .byte $DC, $08, <tab07, >tab07  ;  SET2I instr: freq cycle 1/2
  .byte $CC, $03                ;  FOR inst: repeat 3h times (level 2)
  .byte $CA, $0D, $08           ;   SETI: set freq effect flag on
  .byte $39, $02
  .byte $CA, $0D, $00           ;   SETI: set freq effect flag off
  .byte $39, $02
  .byte $CE                     ;  NEXT (level 2)
  .byte $45, $02
  .byte $CA, $0D, $08           ;  SETI: set freq effect flag on
  .byte $39, $02
  .byte $CA, $0D, $00           ;  SETI: set freq effect flag off
  .byte $45, $02
  .byte $CA, $0D, $08           ;  SETI: set freq effect flag on
  .byte $39, $02
```

```
  .byte $CA, $0D, $00             ;  SETI: set freq effect flag off
  .byte $39, $02
  .byte $45, $02
  .byte $CA, $0D, $08             ;  SETI: set freq effect flag on
  .byte $39, $02
  .byte $CA, $0D, $00             ;  SETI: set freq effect flag off
  .byte $39, $02
  .byte $39, $02
  .byte $39, $02
  .byte $DC, $08, <tab0b, >tab0b  ;  SET2I instr: freq cycle 1/2
  .byte $CA, $0D, $08             ;  SETI: set freq effect flag on
  .byte $39, $02
  .byte $CA, $0D, $00             ;  SETI: set freq effect flag off
  .byte $39, $02
  .byte $CA, $0D, $08             ;  SETI: set freq effect flag on
  .byte $CC, $04                  ;  FOR inst: repeat 4h times (level 2)
  .byte $39, $02
  .byte $CE                       ;  NEXT (level 2)
  .byte $CC, $02                  ;  FOR inst: repeat 2h times (level 2)
  .byte $CA, $0D, $00             ;   SETI: set freq effect flag off
  .byte $45, $02
  .byte $CA, $0D, $08             ;   SETI: set freq effect flag on
  .byte $39, $02
  .byte $CE                       ;  NEXT (level 2)
  .byte $39, $02
  .byte $CA, $0D, $00             ;  SETI: set freq effect flag off
  .byte $45, $02
  .byte $CA, $0D, $08             ;  SETI: set freq effect flag on
  .byte $39, $02
  .byte $CA, $0D, $00             ;  SETI: set freq effect flag off
  .byte $45, $02
  .byte $39, $02
  .byte $39, $02
  .byte $DC, $08, <tab0f, >tab0f  ;  SET2I instr: freq cycle 1/2
  .byte $CC, $03                  ;  FOR inst: repeat 3h times (level 2)
  .byte $CA, $0D, $08             ;   SETI: set freq effect flag on
  .byte $37, $02
  .byte $CA, $0D, $00             ;   SETI: set freq effect flag off
  .byte $37, $02
  .byte $CE                       ;  NEXT (level 2)
  .byte $48, $02
  .byte $CA, $0D, $08             ;  SETI: set freq effect flag on
  .byte $37, $02
  .byte $CA, $0D, $00             ;  SETI: set freq effect flag off
  .byte $48, $02
  .byte $CA, $0D, $08             ;  SETI: set freq effect flag on
  .byte $37, $02
  .byte $CA, $0D, $00             ;  SETI: set freq effect flag off
  .byte $37, $02
  .byte $48, $02
  .byte $CA, $0D, $08             ;  SETI: set freq effect flag on
  .byte $37, $02
  .byte $CA, $0D, $00             ;  SETI: set freq effect flag off
  .byte $37, $02
  .byte $37, $02
  .byte $37, $02
  .byte $DC, $08, <tab13, >tab13  ;  SET2I instr: freq cycle 1/2
  .byte $CA, $0D, $08             ;  SETI: set freq effect flag on
  .byte $37, $02
  .byte $CA, $0D, $00             ;  SETI: set freq effect flag off
  .byte $37, $02
  .byte $CA, $0D, $08             ;  SETI: set freq effect flag on
  .byte $CC, $04                  ;  FOR inst: repeat 3h times (level 2)
  .byte $37, $02
  .byte $CE                       ;  NEXT (level 2)
  .byte $CC, $02                  ;  FOR inst: repeat 3h times (level 2)
  .byte $CA, $0D, $00             ;   SETI: set freq effect flag off
  .byte $48, $02
  .byte $CA, $0D, $08             ;   SETI: set freq effect flag on
  .byte $37, $02
  .byte $CE                       ;  NEXT (level 2)
  .byte $37, $02
  .byte $CA, $0D, $00             ;  SETI: set freq effect flag off
  .byte $48, $02
  .byte $CA, $0D, $08             ;  SETI: set freq effect flag on
  .byte $37, $02
  .byte $CA, $0D, $00             ;  SETI: set freq effect flag off
  .byte $48, $02
  .byte $37, $02
  .byte $37, $02
  .byte $CE                       ; NEXT (level 1)
  .byte $C0                       ; RTS instruction

; filter table 01 definition
fil01:
  .byte $4D, $01                  ; add filter low/high value 1
  .byte $E2, $FF                  ; add filter low/high value 2
  .byte $00, $00                  ; add filter low/high value 3
  .byte $00, $00                  ; add filter low/high value 4
  .byte $03                       ; filter cycle 1
```

```
  .byte $14                     ; filter cycle 2
  .byte $00                     ; filter cycle 3
  .byte $00                     ; filter cycle 4
  .byte $00                     ; filter initial delay
  .byte $04                     ; filter effct flag
  .byte $01                     ; filter low value (8 bit)
  .byte $00                     ; filter high value (3 bit)

; instrument 05 definition
ins05:
  .byte $14, $00                ; freq low/high add 1
  .byte $EC, $FF                ; freq low/high add 2
  .byte $14, $00                ; freq low/high add 3
  .byte $1C, $00                ; freq low/high add 4
  .byte $03, $06                ; freq cycle 1/2
  .byte $03, $00                ; freq cycle 3/4
  .byte $28, $07                ; freq delay initial/freq effect flag

  .byte $32, $32                ; wave cycle 1/2
  .byte $28, $05                ; wave delay initial/wave effect flag
  .byte $0A, $00                ; wave low/high add 1
  .byte $F6, $FF                ; wave low/high add 2
  .byte $00, $08                ; wave low/high
  .byte $41                     ; control: rectangular
  .byte $14, $C8                ; AD/SR
  .byte $FF
  .byte $32

; filter table 02 definition
fil02:
  .byte $4D, $01                ; add filter low/high value 1
  .byte $D3, $FF                ; add filter low/high value 2
  .byte $FB, $FF                ; add filter low/high value 3
  .byte $FF, $FF                ; add filter low/high value 4
  .byte $03, $14                ; filter cycle 1/2
  .byte $0A, $32                ; filter cycle 3/4
  .byte $00                     ; filter initial delay
  .byte $04                     ; filter effect flag
  .byte $01                     ; filter low value (8 bit)
  .byte $00                     ; filter high value (3 bit)

ins06:                          ; instrument 06 definition
insF01:                         ; instrument frequency table 1
  .byte $19, $00                ; freq low/high add 1
  .byte $E7, $FF                ; freq low/high add 2
  .byte $19, $00                ; freq low/high add 3
  .byte $00, $00                ; freq low/high add 4
  .byte $02, $04                ; freq cycle 1/2
  .byte $02, $00                ; freq cycle 3/4
  .byte $06, $05                ; freq delay initial/freq effect flag

  .byte $32, $32                ; wave cycle 1/2
  .byte $00, $05                ; wave delay initial/wave effect flag
  .byte $14, $00                ; wave low/high add 1
  .byte $EC, $FF                ; wave low/high add 2
  .byte $00, $06                ; wave low/high
  .byte $41                     ; control: rectangular
  .byte $14, $E8                ; AD/SR
  .byte $1E
  .byte $28

insF02:                         ; instrument frequency table 2
  .byte $20, $00                ; freq low/high add 1
  .byte $00, $00                ; freq low/high add 2
  .byte $00, $00                ; freq low/high add 3
  .byte $00, $00                ; freq low/high add 4
  .byte $FF, $00                ; freq cycle 1/2
  .byte $00, $00                ; freq cycle 3/4
  .byte $0A, $04                ; freq delay initial/freq effect flag

sub01:
  .byte $5F, $20
sub_1:
  .byte $21, $20
  .byte $5F, $1A
  .byte $23, $06
  .byte $24, $20
  .byte $5F, $1A
  .byte $26, $02
  .byte $28, $02
  .byte $24, $02
  .byte $21, $20
  .byte $5F, $1A
  .byte $23, $06
  .byte $24, $20
  .byte $5F, $20
  .byte $C0                     ; RTS instruction

sub02:
  .byte $21, $04
```

```
  .byte $21, $04
  .byte $2D, $01
  .byte $50, $01
  .byte $21, $04
  .byte $21, $08
  .byte $21, $02
  .byte $2D, $02
  .byte $1C, $02
  .byte $1F, $04
  .byte $21, $04
  .byte $21, $02
  .byte $21, $02
  .byte $2D, $01
  .byte $50, $01
  .byte $21, $04
  .byte $21, $04
  .byte $21, $04
  .byte $21, $02
  .byte $2D, $02
  .byte $21, $02
  .byte $23, $04
  .byte $24, $04
  .byte $24, $02
  .byte $24, $02
  .byte $30, $01
  .byte $50, $03
  .byte $24, $02
  .byte $24, $04
  .byte $24, $02
  .byte $24, $02
  .byte $24, $02
  .byte $30, $02
  .byte $F0, <insF02, >insF02     ; SETFI: set frequency effect for instrument
  .byte $1F, $06
  .byte $F0, <insF01, >insF01     ; SETFI: set frequency effect for instrument
  .byte $24, $02
  .byte $18, $02
  .byte $24, $02
  .byte $18, $02
  .byte $30, $04
  .byte $24, $02
  .byte $24, $04
  .byte $24, $02
  .byte $34, $02
  .byte $24, $02
  .byte $30, $02
  .byte $28, $02
  .byte $2B, $02
  .byte $2D, $02
  .byte $C0                       ; RTS instruction

exec01:
      lda  #$10
      sta  TEMP
      rts

exec02:
      lda  #$00
      sta  TEMP
      sta  $D417                  ; Filter resonance control/voice input control
      sta  voiceNumber            ; voice number (0,1,2) on where apply filter
      rts

tune1_voice3:
  .byte $E2, <fil02, >fil02       ; FILTA instr: set filter table
  .byte $D2, $1C, <ins05, >ins05  ; SETNI instr: set all the instrument table
  .byte $BF, $58
  .byte $65, $28
  .byte $CC, $04                  ; FOR inst: repeat 4h times
  .byte $5F, $20
  .byte $CE                       ; NEXT instr
  .byte $CA, $0D, $05             ; SETI: set freq effect flag on
  .byte $1F, $20
  .byte $CC, $03                  ; FOR inst: repeat 3h times
  .byte $5F, $20
  .byte $CE                       ; NEXT instr
  .byte $DE, $0C, $80, $07        ; SET2CI instr: delay initial | freq effect flag
  .byte $DE, $06, $14, $00        ; SET2CI instr: freq low|high add 4
  .byte $CA, $0D, $05             ; SETI: set freq effect flag on
  .byte $C2, <sub01, >sub01       ; JSR instruction
  .byte $D2, $1C, <ins06 >ins06   ; SETNI instr: set all the instrument table
  .byte $E0
  .byte $D8, <exec01, >exec01     ; EXCT: execute given address code
  .byte $C2, <sub02, >sub02       ; JSR instruction: execute subroutine
  .byte $C2, <sub02, >sub02       ; JSR instruction: execute subroutine
  .byte $D2, $1C, <ins02, >ins02  ; SETNI instr: set all the instrument table
  .byte $E2, <fil01, >fil01       ; FILTA instr: set filter table
  .byte $CC, $03                  ; FOR inst: repeat 3h times
  .byte $5F, $20
  .byte $CE                       ; NEXT instr
```

```
  .byte $CC, $02                ; FOR inst: repeat 2h times
  .byte $43, $10
  .byte $42, $10
  .byte $40, $10
  .byte $3E, $10
  .byte $3C, $10
  .byte $3B, $10
  .byte $39, $10
  .byte $37, $10
  .byte $CE                     ; NEXT instr
  .byte $43, $10
  .byte $A2, $17
  .byte $E2, <fil02, >fil02     ; FILTA instr: set filter table
  .byte $D2, $1C, <ins05, >ins05  ; SETNI instr: set all the instrument table
  .byte $BF, $01
  .byte $65, $28
  .byte $D2, $1C, <ins06, >ins06  ; SETNI instr: set all the instrument table
  .byte $C2, <sub02, >sub02     ; JSR instruction: execute subroutine
  .byte $C2, <sub02, >sub02     ; JSR instruction: execute subroutine
  .byte $D8, <exec02, >exec02   ; EXCT: execute given address code: filter to 0
  .byte $5F, $10
  .byte $CC, $0A                ; FOR inst: repeat Ah times
  .byte $5F, $20
  .byte $CE                     ; NEXT instr
  .byte $C0                     ; RTS instruction

; instrument 07 definition
ins07:
  .byte $1E, $00                ; freq low/high add 1
  .byte $E2, $FF                ; freq low/high add 2
  .byte $1E, $00                ; freq low/high add 3
  .byte $00, $00                ; freq low/high add 4
  .byte $03, $05                ; freq cycle 1/2
  .byte $02, $00                ; freq cycle 3/4
  .byte $08, $05                ; freq delay initial/freq effect flag

  .byte $00, $00                ; wave cycle 1/2
  .byte $00, $00                ; wave delay initial/wave effect flag
  .byte $00, $00                ; wave low/high add 1
  .byte $00, $00                ; wave low/high add 2
  .byte $00, $08                ; wave low/high
  .byte $49                     ; control: rectangular, test bit on
  .byte $06, $99                ; AD/SR
  .byte $04, $28

sub03:
  .byte $37, $04
  .byte $39, $02
  .byte $3A, $02
  .byte $39, $03
  .byte $5F, $01
  .byte $37, $03
  .byte $5F, $01
  .byte $C0                     ; RTS instruction

sub04:
  .byte $39, $03
  .byte $5F, $01
  .byte $3E, $03
  .byte $5F, $01
  .byte $39, $07
  .byte $5F, $01
  .byte $C0                     ; RTS instruction

sub05:
  .byte $C2, <sub03, >sub03     ; JSR instruction: execute subroutine
  .byte $C2, <sub04, >sub04     ; JSR instruction: execute subroutine
  .byte $C2, <sub03, >sub03     ; JSR instruction: execute subroutine
  .byte $37, $04
  .byte $36, $04
  .byte $37, $04
  .byte $39, $04
  .byte $C2, <sub03, >sub03     ; JSR instruction: execute subroutine
  .byte $C2, <sub04, >sub04     ; JSR instruction: execute subroutine
  .byte $3A, $04
  .byte $3C, $02
  .byte $3E, $02
  .byte $3C, $03
  .byte $5F, $01
  .byte $3A, $03
  .byte $5F, $01
  .byte $3C, $03
  .byte $5F, $01
  .byte $41, $03
  .byte $5F, $01
  .byte $3C, $07
  .byte $5F, $01
  .byte $CC, $02                ; FOR inst: repeat 2h times (level 1)
  .byte $CC, $03                ;  FOR inst: repeat 3h times (level 2)
  .byte $45, $02
```

```
  .byte $45, $02
  .byte $5F, $04
  .byte $CE                       ;  NEXT instr (level 2)
  .byte $45, $02
  .byte $43, $02
  .byte $41, $02
  .byte $3C, $02
  .byte $3E, $1C
  .byte $5F, $04
  .byte $CE                       ; NEXT instr (level 1)
  .byte $C0                       ; RTS instruction

tune2_voice1:
  .byte $D2, $1C, <ins07, >ins07  ; SETNI instr: set all the instrument table
  .byte $BF, $01
t2v1_:
  .byte $CC, $04                  ; FOR inst: repeat 4h times
  .byte $C2, <sub05, >sub05       ; JSR instruction: execute subroutine
  .byte $CE                       ; NEXT instr
  .byte $CC, $40                  ; FOR inst: repeat 40h times
  .byte $5F, $04                  ;  no sound (rest)
  .byte $CE                       ; NEXT instr
  .byte $C4, <t2v1_, >t2v1_       ; JMP instr.: jump to given address location

; instrument 08 definition
ins08:
  .byte $1E, $00                  ; freq low/high add 1
  .byte $E2, $FF                  ; freq low/high add 2
  .byte $1E, $00                  ; freq low/high add 3
  .byte $00, $00                  ; freq low/high add 4
  .byte $02, $04                  ; freq cycle 1/2
  .byte $02, $00                  ; freq cycle 3/4
  .byte $06, $05                  ; freq delay initial/freq effect flag

  .byte $00, $00                  ; wave cycle 1/2
  .byte $00, $00                  ; wave delay initial/wave effect flag
  .byte $00, $00                  ; wave low/high add 1
  .byte $00, $00                  ; wave low/high add 2
  .byte $00, $08                  ; wave low/high
  .byte $29                       ; control: sawtooth and test bit on
  .byte $06, $99                  ; AD/SR
  .byte $04
  .byte $28

tune2_voice2:
  .byte $D2, $1C, <ins08, >ins08  ; SETNI instr: set all the instrument table
  .byte $BF, $01
t2v2_:
  .byte $CC, $03                  ; FOR inst: repeat 3h times
  .byte $C2, <sub05, >sub05       ; JSR instruction: execute subroutine
  .byte $5F, $02
  .byte $CE                       ; NEXT instr
  .byte $CC, $FA                  ; FOR inst: repeat FAh times
  .byte $5F, $01
  .byte $CE                       ; NEXT instr
  .byte $C2, <sub05, >sub05       ; JSR instruction: execute subroutine
  .byte $C4, <t2v2_, >t2v2_       ; JMP inst.: jump to given address location

; instrument 09 definition
ins09:
  .byte $00, $00                  ; freq low/high add 1
  .byte $00, $00                  ; freq low/high add 2
  .byte $00, $00                  ; freq low/high add 3
  .byte $00, $00                  ; freq low/high add 4
  .byte $00, $00                  ; freq cycle 1/2
  .byte $00, $00                  ; freq cycle 3/4
  .byte $00, $00                  ; freq delay initial/freq effect flag

  .byte $FF, $00                  ; wave cycle 1/2
  .byte $01, $04                  ; wave delay initial/wave effect flag
  .byte $64, $00                  ; wave low/high add 1
  .byte $00, $00                  ; wave low/high add 2
  .byte $00, $02                  ; wave low/high
  .byte $41                       ; control: rectangular
  .byte $06, $48                  ; AD/SR
  .byte $0A
  .byte $14

sub06:
  .byte $CC, $02                  ; FOR inst: repeat 2h times
  .byte $00, $02
  .byte $00, $02
  .byte $0C, $02
  .byte $0C, $02
  .byte $CE                       ; NEXT inst
  .byte $C0                       ; RTS instruction

sub07:
  .byte $C6, $1F, <sub06, >sub06  ; JSRT: execute subroutine with transpose
  .byte $C6, $1D, <sub06, >sub06  ; JSRT: execute subroutine with transpose
```

```
  .byte $C6, $1B, <sub06, >sub06  ; JSRT: execute subroutine with transpose
  .byte $C6, $1A, <sub06, >sub06  ; JSRT: execute subroutine with transpose
  .byte $C0                       ; RTS instruction

tune2_voice3:
  .byte $D2, $1C, <ins09, >ins09  ; SETNI instr: set all the instrument table
  .byte $BF, $01
t2v3_:
  .byte $C2, <sub07, >sub07       ; JSR instruction: execute subroutine
  .byte $C6, $1F, <sub06, >sub06  ; JSRT: execute subroutine with transpose
  .byte $C6, $1D, <sub06, >sub06  ; JSRT: execute subroutine with transpose
  .byte $C6, $1B, <sub06, >sub06  ; JSRT: execute subroutine with transpose
  .byte $C6, $1D, <sub06, >sub06  ; JSRT: execute subroutine with transpose
  .byte $C2, <sub07, >sub07       ; JSR instruction: execute subroutine
  .byte $C2, <sub07, >sub07       ; JSR instruction: execute subroutine
  .byte $C4, <t2v3_, >t2v3_       ; JMP instr.

par01:
  .byte $11                       ; control: triangle
  .byte $23, $E4                  ; AD/SR
  .byte $14
  .byte $0A

tune7_voice1:
  .byte $D4, <par01, >par01       ; INSTR instruction: select instrument parameter
  .byte $BF, $15
  .byte $37, $01
  .byte $3D, $01
  .byte $3E, $01
  .byte $41, $01
  .byte $43, $01
  .byte $47, $01
  .byte $C0                       ; RTS instruction

tune7_voice2:
  .byte $D4, <par01, >par01       ; INSTR instruction: select instrument parameter
  .byte $BF, $0B
  .byte $37, $01
  .byte $3D, $01
  .byte $3E, $01
  .byte $41, $01
  .byte $43, $01
  .byte $47, $01
  .byte $4A, $01
  .byte $C0                       ; RTS instruction

par02:
  .byte $41                       ; control: rectangular
  .byte $24, $A4                  ; AD/SR
  .byte $14
  .byte $04

tune7_voice3:
  .byte $D4, <par02, >par02       ; INSTR instruction: select instrument parameter
  .byte $DC, $16, $00, $08        ; SET2I: set wave low/high of voice
  .byte $BF, $01
  .byte $37, $01
  .byte $3D, $01
  .byte $3E, $01
  .byte $41, $01
  .byte $43, $01
  .byte $47, $01
  .byte $4A, $01
  .byte $C0                       ; RTS instruction

; instrument frequency table 3
insF03:
  .byte $EB, $F2                  ; freq low/high add 1
  .byte $00, $00                  ; freq low/high add 2
  .byte $15, $0D                  ; freq low/high add 3
  .byte $00, $00                  ; freq low/high add 4
  .byte $01, $03                  ; freq cycle 1/2
  .byte $01, $09                  ; freq cycle 3/4
  .byte $09, $05                  ; freq delay initial/freq effect flag

par03:
  .byte $19                       ; control: triangle, test bit on
  .byte $13, $E4                  ; AD/SR
  .byte $02
  .byte $0A

tune6_voice1:
  .byte $D4, <par03, >par03       ; INSTR instruction: select instrument parameter
  .byte $BF, $1B
t6v1_:
  .byte $3A, $01
  .byte $35, $01
  .byte $37, $01
  .byte $32, $01
  .byte $35, $01
```

```
  .byte $30, $01
  .byte $32, $01
  .byte $2E, $01
  .byte $F0, <insF03, >insF03     ; SETFI: set frequency effect for instrument
  .byte $2B, $03
  .byte $CA, $0D, $00             ; SETI: set freq effect flag off
  .byte $2B, $02
  .byte $C0                       ; RTS instr.

tune6_voice2:
  .byte $D4, <par03, >par03       ; INSTR: select all instrument table
  .byte $BF, $04
  .byte $C4, <t6v1_, >t6v1_       ; JMP instr.: jump to given address location

par04:
  .byte $49                       ; control: rectangular + test
  .byte $24, $A4                  ; AD/SR
  .byte $02, $04

tune6_voice3:
  .byte $D4, <par04, >par04       ; INSTR inst.: select all instrument table
  .byte $DC, $16, $00, $08        ; SET2I: set wave low/high of voice
  .byte $BF, $01
  .byte $C4, <t6v1_, >t6v1_       ; JMP instr.: jump to given address location


; instrument 0A definition
ins0A:
  .byte $0E, $00                  ; freq low/high add 1
  .byte $F2, $FF                  ; freq low/high add 2
  .byte $0E, $00                  ; freq low/high add 3
  .byte $00, $00                  ; freq low/high add 4
  .byte $05, $0A                  ; freq cycle 1/2
  .byte $05, $00                  ; freq cycle 3/4
  .byte $14, $05                  ; freq delay initial/freq effect flag

  .byte $FF, $00                  ; wave cycle 1/2
  .byte $00, $04                  ; wave delay initial/wave effect flag
  .byte $0A, $00                  ; wave low/high add 1
  .byte $00, $00                  ; wave low/high add 2
  .byte $00, $08                  ; wave low/high
  .byte $49                       ; control: rectangular, test bit on
  .byte $06, $98                  ; AD/SR
  .byte $05
  .byte $1E

tune3_voice1:
  .byte $D2, $1C, <ins0A, >ins0A  ; SETNI instr: set all the instrument table
  .byte $30, $03
  .byte $30, $01
  .byte $37, $08
  .byte $96, $0C
  .byte $97, $0C
  .byte $99, $0C
  .byte $37, $10
  .byte $30, $03
  .byte $30, $01
  .byte $37, $08
  .byte $39, $01
  .byte $37, $01
  .byte $36, $01
  .byte $39, $01
  .byte $37, $08
  .byte $C0                       ; RTS instruction

tune3_voice2:
  .byte $D2, $1C, <ins0A, >ins0A  ; SETNI instr: set all the instrument table
  .byte $2B, $03
  .byte $2B, $01
  .byte $34, $08
  .byte $93, $0C
  .byte $94, $0C
  .byte $95, $0C
  .byte $34, $10
  .byte $2B, $03
  .byte $2B, $01
  .byte $34, $08
  .byte $35, $01
  .byte $34, $01
  .byte $33, $01
  .byte $35, $01
  .byte $34, $08
  .byte $C0                       ; RTS instruction

; filter table 03 definition
fil03:
  .byte $4D, $01                  ; add filter low/high value 1
  .byte $D3, $FF                  ; add filter low/high value 2
  .byte $14, $00                  ; add filter low/high value 3
  .byte $EC, $FF                  ; add filter low/high value 4
```

```
  .byte $03, $14                ; filter cycle 1/2
  .byte $32, $32                ; filter cycle 3/4
  .byte $00                     ; filter initial delay
  .byte $04                     ; filter effect flag
  .byte $01                     ; filter low value (8 bit)
  .byte $00                     ; filter high value (3 bit)

; instrument 0B definition
ins0B:
  .byte $0F, $00                ; freq low/high add 1
  .byte $F1, $FF                ; freq low/high add 2
  .byte $0F, $00                ; freq low/high add 3
  .byte $00, $00                ; freq low/high add 4
  .byte $04, $08                ; freq cycle 1/2
  .byte $04, $00                ; freq cycle 3/4
  .byte $0E, $05                ; freq delay initial/freq effect flag

  .byte $19, $00                ; wave cycle 1/2
  .byte $00, $04                ; wave delay initial/wave effect flag
  .byte $0A, $00                ; wave low/high add 1
  .byte $00, $00                ; wave low/high add 2
  .byte $00, $08                ; wave low/high
  .byte $49                     ; control: rectangular and test bit on
  .byte $06, $98                ; AD/SR
  .byte $05
  .byte $1E

tune3_voice3:
  .byte $D2, $1C, <ins0B, >ins0B  ; SETNI instr: set all the instrument table
  .byte $E0                     ; LF3: low filter (max resonance) on voice 3
  .byte $E2, <fil03, >fil03       ; FILTA instr: set filter table
  .byte $CC, $03                ; FOR inst: repeat 3h times
  .byte $24, $04
  .byte $1F, $04
  .byte $CE                     ; NEXT instr
  .byte $24, $02
  .byte $1F, $02
  .byte $21, $02
  .byte $23, $02
  .byte $CC, $02                ; FOR inst: repeat 2h times
  .byte $24, $04
  .byte $1F, $04
  .byte $CE                     ; NEXT instr
  .byte $24, $10
  .byte $5F, $08
  .byte $C0                     ; RTS instruction

tune4_voice3:
  .byte $D2, $1C, <ins0B, >ins0B  ; SETNI instr: set all the instrument table
  .byte $E0                     ; LF3: low filter (max resonance) on voice 3
  .byte $E2, <fil03, >fil03       ; FILTA instr: set filter table
  .byte $5F, $03
  .byte $1B, $01
  .byte $1B, $01
  .byte $1B, $01
  .byte $1B, $03
  .byte $1D, $03
  .byte $21, $03
  .byte $CC, $02                ; FOR inst: repeat 2h times
  .byte $84, $10
  .byte $7F, $11
  .byte $CE                     ; NEXT instr
  .byte $24, $06
  .byte $C0                     ; RTS instruction

tune4_voice2
  .byte $BF, $14
tune4_voice1:
  .byte $D2, $1C, <ins0A, >ins0A  ; SETNI instr: set all the instrument table
  .byte $37, $02
  .byte $37, $01
  .byte $3A, $06
  .byte $99, $10
  .byte $97, $11
  .byte $95, $10
  .byte $99, $11
  .byte $37, $06
  .byte $C0                     ; RTS instruction

sub08:
  .byte $42, $01
  .byte $3F, $01
  .byte $3C, $01
  .byte $3A, $01
  .byte $36, $01
  .byte $33, $01
  .byte $30, $01
  .byte $2E, $01
  .byte $2A, $01
  .byte $2E, $01
```

```
  .byte $30, $01
  .byte $33, $01
  .byte $36, $01
  .byte $3A, $01
  .byte $3C, $01
  .byte $C0                     ; RTS instruction

tune5_voice1:
  .byte $D2, $1C, <ins0B, >ins0B  ; SETNI instr: set all the instrument table
  .byte $37, $02
  .byte $36, $02
  .byte $33, $02
  .byte $35, $04
  .byte $36, $02
  .byte $37, $04
  .byte $CC, $03                ; FOR inst: repeat 3h times
  .byte $36, $02
  .byte $36, $04
  .byte $CE                     ; NEXT instr
  .byte $5F, $01
  .byte $C2, <sub08, >sub08     ; JSR instruction: execute subroutine
  .byte $3F, $01
  .byte $42, $01
  .byte $5F, $02
  .byte $36, $02
  .byte $C0                     ; RTS instruction

tune5_voice2:
  .byte $D2, $1C, <ins0B, >ins0B  ; SETNI instr: set all the instrument table
  .byte $30, $02
  .byte $30, $02
  .byte $CC, $05                ; FOR inst: repeat 5h times
  .byte $30, $02
  .byte $30, $04
  .byte $CE                     ; NEXT instr
  .byte $5F, $01
  .byte $42, $01
  .byte $C2, <sub08, >sub08     ; JSR instruction: execute subroutine
  .byte $3F, $01
  .byte $5F, $02
  .byte $30, $02
  .byte $C0                     ; RTS instruction

tune5_voice3:
  .byte $D2, $1C, <ins0B, >ins0B  ; SETNI instr: set all the instrument table
  .byte $E0                     ; LF3: low filter (max resonance) on voice 3
  .byte $E2, <fil03, >fil03     ; FILTA instr: set filter table
  .byte $CC, $04                ; FOR inst: repeat 4h times
  .byte $18, $02
  .byte $24, $02
  .byte $CE                     ; NEXT instr
  .byte $CC, $04                ; FOR inst: repeat 4h times
  .byte $1B, $02
  .byte $27, $02
  .byte $CE                     ; NEXT instr
  .byte $5F, $03
  .byte $3F, $01
  .byte $42, $01
  .byte $C2, <sub08, >sub08     ; JSR instruction: execute subroutine
  .byte $5F, $02
  .byte $1E, $02
  .byte $C0                     ; RTS instruction

par05:
  .byte $41                     ; control: rectangular
  .byte $06, $59                ; AD/SR
  .byte $19
  .byte $14

tune8_voice1:
  .byte $D2, $17, <ins0B, >ins0B  ; SETNI instr: set all the instrument table
  .byte $D4, <par05, >par05     ; INSTR instruction: select instrument parameter
  .byte $37, $01
  .byte $37, $01
  .byte $37, $04
  .byte $3C, $04
  .byte $3B, $04
  .byte $3E, $04
  .byte $3C, $04
  .byte $37, $0A
  .byte $37, $01
  .byte $37, $01
  .byte $37, $04
  .byte $3C, $04
  .byte $3B, $04
  .byte $3E, $04
  .byte $A0, $13
  .byte $9E, $09
  .byte $40, $01
  .byte $C0                     ; RTS instruction
```

```
tune8_voice2:
  .byte $D2, $17, <ins0B, >ins0B  ; SETNI instr: set all the instrument table
  .byte $D4, <par05, >par05       ; INSTR instruction: select instrument parameter
  .byte $43, $01
  .byte $43, $01
  .byte $43, $04
  .byte $48, $04
  .byte $47, $04
  .byte $4A, $04
  .byte $48, $04
  .byte $40, $01
  .byte $40, $01
  .byte $40, $01
  .byte $50, $01
  .byte $41, $01
  .byte $41, $01
  .byte $41, $01
  .byte $50, $01
  .byte $40, $01
  .byte $40, $01
  .byte $40, $01
  .byte $43, $01
  .byte $43, $04
  .byte $48, $04
  .byte $47, $04
  .byte $4A, $04
  .byte $9C, $13
  .byte $9B, $09
  .byte $3C, $01
  .byte $C0                       ; RTS instruction

tune8_voice3:
  .byte $D2, $1C, <ins0B, >ins0B  ; SETNI instr: set all the instrument table
  .byte $E0                       ; LF3: low filter (max resonance) on voice 3
  .byte $E2, <fil03, >fil03       ; FILTA instr: set filter table
  .byte $5F, $02
  .byte $CC, $06                  ; FOR inst: repeat 6h times
  .byte $24, $04
  .byte $1F, $04
  .byte $CE                       ; NEXT instr
  .byte $CA, $18, $41             ; SET instr: 18h set control to rectangular
  .byte $CA, $1B, $28             ; ??
  .byte $84, $13
  .byte $7F, $09
  .byte $24, $01
  .byte $C0                       ; RTS instruction

  org $3F00

InstrVoice1:
  .byte <inst_C0_v1, >inst_C0_v1  ; C0:  RTS
  .byte <inst_C2_v1, >inst_C2_v1  ; C2:  JSR
  .byte <inst_C4_v1, >inst_C4_v1  ; C4:  JMP
  .byte <inst_C6_v1, >inst_C6_v1  ; C6:  JSRT
  .byte $5A, $0A
  .byte <inst_CA_v1, >inst_CA_v1  ; CA:  SET
  .byte <inst_CC_v1, >inst_CC_v1  ; CC:  FOR
  .byte <inst_CE_v1, >inst_CE_v1  ; CE:  NEXT
  .byte $5A, $0A
  .byte <inst_D2_v1, >inst_D2_v1  ; D2:  SETNI
  .byte <inst_D4_v1, >inst_D4_v1  ; D4:  INSTR
  .byte <inst_D6_v1, >inst_D6_v1  ; D6:  SETCI
  .byte $5A, $0A
  .byte $5A, $0A
  .byte $5A, $0A
  .byte <inst_DE_v1, >inst_DE_v1  ; DE:  SET2CI
  .byte $5A, $0A
  .byte $5A, $0A
  .byte $5A, $0A
  .byte $5A, $0A
  .byte $5A, $0A
  .byte $5A, $0A
  .byte $5A, $0A
  .byte $5A, $0A
  .byte $5A, $0A
  .byte <inst_F0_v1, >inst_F0_v1  ; F0:  SETFI

InstrVoice2:
  .byte <inst_C0_v2, >inst_C0_v2  ; C0:  RTS
  .byte <inst_C2_v2, >inst_C2_v2  ; C2:  JSR
  .byte <inst_C4_v2, >inst_C4_v2  ; C4:  JMP
  .byte $5D, $0A
  .byte $5D, $0A
  .byte <inst_CA_v2, >inst_CA_v2  ; CA:  SET
  .byte <inst_CC_v2, >inst_CC_v2  ; CC:  FOR
  .byte <inst_CE_v2, >inst_CE_v2  ; CE:  NEXT
  .byte $5D, $0A
  .byte <inst_D2_v2, >inst_D2_v2  ; D2:  SETNI
  .byte <inst_D4_v2, >inst_D4_v2  ; D4:  INSTR
  .byte <inst_D6_v2, >inst_D6_v2  ; D6:  SETCI
```

```
  .byte $5D, $0A
  .byte $5D, $0A
  .byte <inst_DC_v2, >inst_DC_v2  ; DC:  SET2I
  .byte <inst_DE_v2, >inst_DE_v2  ; DE:  SET2CI
  .byte $5D, $0A
  .byte $5D, $0A
  .byte $5D, $0A
  .byte $5D, $0A
  .byte $5D, $0A
  .byte $5D, $0A
  .byte $5D, $0A
  .byte $5D, $0A
  .byte <inst_F0_v2, >inst_F0_v2  ; F0:  SETFI

InstrVoice3:
  .byte <inst_C0_v3, >inst_C0_v3  ; C0:  RTS
  .byte <inst_C2_v3, >inst_C2_v3  ; C2:  JSR
  .byte <inst_C4_v3, >inst_C4_v3  ; C4:  JMP
  .byte <inst_C6_v3, >inst_C6_v3  ; C6:  JSRT
  .byte $60, $0A
  .byte <inst_CA_v3, >inst_CA_v3  ; CA:  SET
  .byte <inst_CC_v3, >inst_CC_v3  ; CC:  FOR
  .byte <inst_CE_v3, >inst_CE_v3  ; CE:  NEXT
  .byte $60, $0A
  .byte <inst_D2_v3, >inst_D2_v3  ; D2:  SETNI
  .byte <inst_D4_v3, >inst_D4_v3  ; D4:  INSTR
  .byte $60, $0A
  .byte <inst_D8_v3, >inst_D8_v3  ; D8:  EXCT
  .byte $60, $0A
  .byte <inst_DC_v3, >inst_DC_v3  ; DC:  SET2I
  .byte <inst_DE_v3, >inst_DE_v3  ; DE:  SET2CI
  .byte <inst_E0_v3, >inst_E0_v3  ; E0:  LF3
  .byte <inst_E2_v3, >inst_E2_v3  ; E2:  FILTA
  .byte $60, $0A
  .byte $60, $0A
  .byte $60, $0A
  .byte $60, $0A
  .byte $60, $0A
  .byte $60, $0A
  .byte <inst_F0_v3, >inst_F0_v3  ; F0:  SETFI

;================================
; Calculate the right address
;================================
calcAddress:
      lda  $1FFF
      sec
      sbc  #$08
      tax
      lda  #$00
      sta  $FB                   ; low address
      sta  $FC                   ; high address
      cpx  #$00
      beq  skipCalc

iterate:
      lda  $FB
      clc
      adc  #$5D
      sta  $FB
      lda  $FC
      adc  #$00
      sta  $FC
      dex
      bne  iterate

skipCalc:
      lda  $FC
      ora  #$20                   ; this is according with org $2000
      sta  $FC

      ldx  #$00
      lda  $FB
      ldy  $FC
      jsr  setSoundEffect
      jsr  fixForNext

      ldx  #$01
      lda  $FB
      ldy  $FC
      jsr  setSoundEffect
      jsr  fixForNext

      ldx  #$02
      lda  $FB
      ldy  $FC
      jsr  setSoundEffect
      jmp  setInterrupt

fixForNext:
```

```
      lda  $FB
      clc
      adc  #$1F
      sta  $FB
      lda  $FC
      adc  #$00
      sta  $FC
      rts

;985E  AE FF 1F   LDX  $1FFF
;9861  BD 90 40   LDA  $4090,X
;9864  D0 03      BNE  $9869
;9866  4C 67 1F   JMP  setInterrupt

;9869  A9 F0      LDA  #$F0
;986B  8D 04 DC   STA  $DC04       Timer A #1: Lo Byte
;986E  A9 49      LDA  #$49
;9870  8D 05 DC   STA  $DC05       Timer A #1: Hi Byte
;9873  4C 67 1F   JMP  setInterrupt

rout13:
  .byte $84, $0C                  ; Play Sample 4
  .byte $86, $06                  ; Play Sample 6
  .byte $86, $06                  ; Play Sample 6
  .byte $82, $0C                  ; Play Sample 2
  .byte $86, $06                  ; Play Sample 6
  .byte $86, $06                  ; Play Sample 6
  .byte $84, $06                  ; Play Sample 4
  .byte $84, $06                  ; Play Sample 4
  .byte $86, $06                  ; Play Sample 6
  .byte $86, $06                  ; Play Sample 6
  .byte $82, $0C                  ; Play Sample 2
  .byte $86, $06                  ; Play Sample 6
  .byte $86, $06                  ; Play Sample 6
  .byte $84, $0C                  ; Play Sample 4
  .byte $86, $06                  ; Play Sample 6
  .byte $86, $06                  ; Play Sample 6
  .byte $82, $0C                  ; Play Sample 2
  .byte $86, $06                  ; Play Sample 6
  .byte $86, $06                  ; Play Sample 6
  .byte $84, $06                  ; Play Sample 4
  .byte $84, $06                  ; Play Sample 4
  .byte $86, $06                  ; Play Sample 6
  .byte $86, $06                  ; Play Sample 6
  .byte $60                       ; RTS

;=================================
; Set sample
; A=sample duration
; X=low address
; Y=high address
;=================================
setSample:
      sta  $DC                    ; store sample duration
      stx  $DA                    ; store low address of sample pattern
      sty  $DB                    ; store high address of sample pattern
      lda  #$07
      sta  $DD                    ; store stack index for sample
      rts

Sample_Tune1:
  .byte $20, <rout01, >rout01     ; JSR instr
  .byte $20, <rout02, >rout02     ; JSR instr
  .byte $20, <rout02, >rout02     ; JSR instr
  .byte $20, <rout01, >rout01     ; JSR instr
  .byte $49, $02                  ; FOR instr: repeat 02 times (level 1)
  .byte $20, <rout02, >rout02     ;  JSR instr
  .byte $20, <rout03, >rout03     ;  JSR instr
  .byte $49, $03                  ;  FOR instr: repeat 03 times (level 2)
  .byte $82, $01                  ;   Play Sample 2
  .byte $82, $01                  ;   Play Sample 2
  .byte $82, $06                  ;   Play Sample 2
  .byte $40                       ;  NEXT instr (level 2)
  .byte $40                       ; NEXT instr (level 1)
  .byte $20, <rout06, >rout06     ; JSR instr
  .byte $87, $78                  ; Play Nothing
  .byte $49, $07                  ; FOR instr: repeat 07 times
  .byte $87, $80                  ;  Play Nothing
  .byte $40                       ; NEXT instr
  .byte $20, <rout04, >rout04     ; JSR instr
  .byte $20, <rout05, >rout05     ; JSR instr
  .byte $20, <rout04, >rout04     ; JSR instr
  .byte $20, <rout07, >rout07     ; JSR instr
  .byte $20, <rout04, >rout04     ; JSR instr
  .byte $20, <rout05, >rout05     ; JSR instr
  .byte $20, <rout06, >rout06     ; JSR instr
  .byte $20, <rout08, >rout08     ; JSR instr
  .byte $20, <rout06, >rout06     ; JSR instr
  .byte $49, $04                  ; FOR instr: repeat 04 times
  .byte $20, <rout06, >rout06     ;  JSR instr
```

```
   .byte $20, <rout08, >rout08     ;  JSR instr
   .byte $40                       ; NEXT instr
   .byte $20, <rout06, >rout06     ; JSR instr
   .byte $20, <rout06, >rout06     ; JSR instr
   .byte $87, $18                  ;  Play Nothing
   .byte $20, <rout02, >rout02     ; JSR instr
   .byte $20, <rout02, >rout02     ; JSR instr
   .byte $49, $03                  ; FOR instr: repeat 03 times
   .byte $20, <rout09, >rout09     ;  JSR instr
   .byte $40                       ; NEXT instr
   .byte $20, <rout06, >rout06     ; JSR instr
   .byte $20, <rout08, >rout08     ; JSR instr
   .byte $84, $02                  ; Play Sample 4
   .byte $84, $06                  ; Play Sample 4
   .byte $20, <rout08, >rout08     ; JSR instr
   .byte $20, <rout10, >rout10     ; JSR instr
   .byte $20, <rout06, >rout06     ; JSR instr
   .byte $84, $02                  ; Play Sample 4
   .byte $84, $06                  ; Play Sample 4
   .byte $20, <rout06, >rout06     ; JSR instr
   .byte $49, $03                  ; FOR instr: repeat 03 times
   .byte $20, <rout09, >rout09     ;  JSR instr
   .byte $40                       ; NEXT instr
   .byte $20, <rout06, >rout06     ; JSR instr
   .byte $20, <rout08, >rout08     ; JSR instr
   .byte $84, $02                  ; Play Sample 4
   .byte $84, $06                  ; Play Sample 4
   .byte $85, $04                  ; Play Sample 5
   .byte $85, $04                  ; Play Sample 5
   .byte $20, <rout10, >rout10     ; JSR instr
   .byte $85, $02                  ; Play Sample 5
   .byte $85, $06                  ; Play Sample 5
   .byte $20, <rout06, >rout06     ; JSR instr
   .byte $49, $04                  ; FOR instr: repeat 04 times
   .byte $86, $02                  ;  Play Sample 6
   .byte $40                       ; NEXT instr
   .byte $49, $03                  ; FOR instr: repeat 03 times
   .byte $20, <rout11, >rout11     ;  JSR instr
   .byte $40                       ; NEXT instr
   .byte $84, $02                  ; Play Sample 4
   .byte $84, $06                  ; Play Sample 4
   .byte $20, <rout08, >rout08     ; JSR instr
   .byte $84, $02                  ; Play Sample 4
   .byte $84, $06                  ; Play Sample 4
   .byte $20, <rout08, >rout08     ; JSR instr
   .byte $20, <rout10, >rout10     ; JSR instr
   .byte $49, $03                  ; FOR instr: repeat 03 times
   .byte $84, $02                  ;  Play Sample 4
   .byte $84, $06                  ;  Play Sample 4
   .byte $40                       ; NEXT instr
   .byte $49, $03                  ; FOR instr: repeat 03 times
   .byte $20, <rout11, >rout11     ;  JSR instr
   .byte $40                       ; NEXT instr
   .byte $84, $02                  ; Play Sample 4
   .byte $84, $06                  ; Play Sample 4
   .byte $20, <rout08, >rout08     ; JSR instr
   .byte $84, $02                  ; Play Sample 4
   .byte $84, $06                  ; Play Sample 4
   .byte $85, $04                  ; Play Sample 5
   .byte $85, $04                  ; Play Sample 5
   .byte $20, <rout10, >rout10     ; JSR instr
   .byte $85, $02                  ; Play Sample 5
   .byte $85, $06                  ; Play Sample 5
   .byte $84, $02                  ; Play Sample 4
   .byte $84, $06                  ; Play Sample 4
   .byte $49, $04                  ; FOR instr: repeat 04 times
   .byte $86, $02                  ;  Play Sample 6
   .byte $40                       ; NEXT instr
   .byte $49, $02                  ; FOR instr: repeat 02 times
   .byte $84, $02                  ;  Play Sample 4
   .byte $84, $0E                  ;  Play Sample 4
   .byte $20, <rout08, >rout08     ;  JSR instr
   .byte $20, <rout08, >rout08     ;  JSR instr
   .byte $20, <rout10, >rout10     ;  JSR instr
   .byte $87, $08                  ;  Play Nothing
   .byte $20, <rout08, >rout08     ;  JSR instr
   .byte $20, <rout08, >rout08     ;  JSR instr
   .byte $40                       ; NEXT instr
   .byte $84, $02
   .byte $84, $0E
   .byte $20, <rout08, >rout08     ; JSR instr
   .byte $20, <rout08, >rout08     ; JSR instr
   .byte $20, <rout10, >rout10     ; JSR instr
   .byte $20, <rout06, >rout06     ; JSR instr
   .byte $20, <rout08, >rout08     ; JSR instr
   .byte $20, <rout08, >rout08     ; JSR instr
   .byte $20, <rout06, >rout06     ; JSR instr

noSample:
   .byte $87, $64                  ; Play Nothing
```

```
  .byte $4C, <noSample, >noSample

;================================
; Set the retun address for
; sample routine
;================================
set_return_sample:
      ldx  $DD                  ; read stack index for sample
      clc
      adc  $DA                  ; add to low addr sample pattern index
      sta  $78,x                ; store low addr sample pattern
      lda  $DB
      adc  #$00
      sta  $80,x                ; store high addr sample pattern
      dex
      stx  $DD                  ; store actual stack index for sample
      rts

;================================
; RTS istruction for sample
; return from subroutine
;================================
inst_60_sample:
      inc  $DD                  ; inc stack index for sample
      ldx  $DD

restoreSampleAddr:
      lda  $78,x                ; read low addr from stack
      sta  $DA                  ; store low addr sample pattern index
      lda  $80,x                ; read high addr from stack
      sta  $DB                  ; store high addr sample pattern index
      rts

;high address to play sample routine
highPSample:
  .byte >PSample1, >PSample2, >PSample3
  .byte >PSample4, >PSample5, >PSample6
  .byte >PNothing

;================================
; JSR sample instruction
; #1 low address
; #2 high address
;================================
Inst_20_sample:
      lda  #$03
      jsr  set_return_sample    ; store return pointer in stack

;================================
; JMP sample instruction
;================================
inst_4C_sample:
      iny
      lda  ($DA),y              ; read low address
      tax
      iny
      lda  ($DA),y              ; read high address
      stx  $DA                  ; set new low address
      sta  $DB                  ; set new high address
      rts

rout01:
  .byte $49, $08                ; FOR instr: repeat 08 times
  .byte $83, $01                ;  Play Sample 3
  .byte $83, $03                ;  Play Sample 3
  .byte $40                     ; NEXT instr
  .byte $49, $08                ; FOR instr: repeat 08 times
  .byte $85, $01                ;  Play Sample 5
  .byte $85, $03                ;  Play Sample 5
  .byte $40                     ; NEXT instr
  .byte $20, <rout12, >rout12   ; JSR instr
  .byte $20, <rout12, >rout12   ; JSR instr
  .byte $20, <rout08, >rout08   ; JSR instr
  .byte $20, <rout08, >rout08   ; JSR instr

rout12:
  .byte $86, $02                ; Play Sample 6
  .byte $86, $0E                ; Play Sample 6
  .byte $60                     ; RTS

rout04:
  .byte $20, <rout06, >rout06   ; JSR instr
  .byte $20, <rout08, >rout08   ; JSR instr
  .byte $20, <rout06, >rout06   ; JSR instr
  .byte $20, <rout08, >rout08   ; JSR instr
  .byte $20, <rout10, >rout10   ; JSR instr
  .byte $20, <rout08, >rout08   ; JSR instr
  .byte $20, <rout08, >rout08   ; JSR instr
  .byte $20, <rout06, >rout06   ; JSR instr
  .byte $20, <rout08, >rout08   ; JSR instr
```

```
  .byte $20, <rout06, >rout06     ; JSR instr
  .byte $20, <rout06, >rout06     ; JSR instr
  .byte $20, <rout08, >rout08     ; JSR instr

rout10:
  .byte $82, $01                  ; Play Sample 2
  .byte $82, $01                  ; Play Sample 2
  .byte $82, $06                  ; Play Sample 2
  .byte $60                       ; RTS

rout05:
  .byte $20, <rout08, >rout08     ; JSR  instr
  .byte $20, <rout06, >rout06     ; JSR  instr

rout08:
  .byte $86, $02                  ; Play Sample 6
  .byte $86, $06                  ; Play Sample 6
  .byte $60                       ; RTS

rout07:
  .byte $20, <rout06, >rout06     ; JSR instr
  .byte $20, <rout08, >rout08     ; JSR instr

rout06:
  .byte $81, $02                  ; Play Sample 1
  .byte $81, $06                  ; Play Sample 1
  .byte $60                       ; RTS

rout02:
  .byte $20, <rout03, >rout03     ; JSR instr
  .byte $4C, <rout02_, >rout02_   ; JMP instr

rout02_:
  .byte $20, <rout06, >rout06     ; JSR instr
  .byte $84, $02                  ; Play Sample 4
  .byte $84, $06                  ; Play Sample 4
  .byte $84, $02                  ; Play Sample 4
  .byte $84, $06                  ; Play Sample 4
  .byte $60                       ; RTS

;=================================
; FOR instr for sample
; #1 number of cycle
;=================================
inst_49_sample:
      lda  #$02
      jsr  set_return_sample      ; set return address for cycle
      pha
      iny
      lda  ($DA),y                ; read number of cycle
      sta  $89,x                  ; store the number of cycle for the for
      pla
      sta  $DB                    ; set low addr sample pattern index
      lda  $79,x
      sta  $DA                    ; set high addr sample pattern index
      rts

;=================================
; NEXT instr. for sample
;=================================
inst_40_sample:
      inc  $DD                    ; inc stack index for sample
      ldx  $DD
      dec  $88,x                  ; dec number of cycle
      beq  cycleEndSample         ; end of cycles?
      jsr  restoreSampleAddr
      dec  $DD
      rts

cycleEndSample:
      stx  $DD
      inc  $DA                    ; go to next instruction (low addr)
      bne  skipFixHigh
      inc  $DB                    ; fix high address
skipFixHigh:
      rts

rout03:
  .byte $20, <rout04, >rout04     ; JSR  instr
  .byte $20, <rout05, >rout05     ; JSR  instr
  .byte $20, <rout04, >rout04     ; JSR  instr
  .byte $20, <rout07, >rout07     ; JSR  instr
  .byte $20, <rout04, >rout04     ; JSR  instr
  .byte $20, <rout05, >rout05     ; JSR  instr
  .byte $4C, <rout04, >rout04     ; JMP  instr

rout09:
  .byte $20, <rout06, >rout06     ; JSR  instr
  .byte $20, <rout08, >rout08     ; JSR  instr
  .byte $83, $02                  ; Play Sample 3
```

```
  .byte $83, $06                ; Play Sample 3
  .byte $20, <rout08, >rout08   ; JSR  instr
  .byte $20, <rout10, >rout10   ; JSR  instr
  .byte $20, <rout08, >rout08   ; JSR  instr
  .byte $83, $02
  .byte $83, $06
  .byte $4C, <rout08, >rout08   ; JMP  instr

rout11:
  .byte $84, $02                ; play Sample 4
  .byte $84, $06                ; play Sample 4
  .byte $20, <rout08, >rout08   ; JSR instr
  .byte $83, $02                ; Play Sample 3
  .byte $83, $06                ; Play Sample 3
  .byte $20, <rout08, >rout08   ; JSR instr
  .byte $20, <rout10, >rout10   ; JSR instr
  .byte $20, <rout08, >rout08   ; JSR instr
  .byte $83, $02                ; Play Sample 3
  .byte $83, $06                ; Play Sample 3
  .byte $4C, <rout08, >rout08   ; JMP instr

Sample_Tune2:
  .byte $87, $01                ; Play Nothing

STune2_:
  .byte $49, $07                ; FOR instr: repeat 07 times
  .byte $85, $18                ;  Play Sample 5
  .byte $40                     ; NEXT instr
  .byte $85, $0C                ; Play Sample 5
  .byte $85, $06                ; Play Sample 5
  .byte $85, $06                ; Play Sample 5
  .byte $49, $06                ; FOR instr: repeat 06 times
  .byte $85, $02                ;  Play Sample 5
  .byte $85, $16                ;  Play Sample 5
  .byte $40                     ; NEXT instr
  .byte $82, $02                ; Play Sample 2
  .byte $82, $16                ; Play Sample 2
  .byte $49, $02                ; FOR instr: repeat 02 times
  .byte $83, $02                ;  Play Sample 3
  .byte $83, $04                ;  Play Sample 3
  .byte $40                     ; NEXT instr
  .byte $49, $02                ; FOR instr: repeat 02 times
  .byte $84, $02                ;  Play Sample 4
  .byte $84, $04                ;  Play Sample 4
  .byte $40                     ; NEXT instr
  .byte $49, $04                ; FOR instr: repeat 04 times (level 1)
  .byte $49, $03                ;  FOR instr: repeat 03 times (level 2)
  .byte $20, <rout13, >rout13   ;   JSR instr
  .byte $82, $06                ;   Play Sample 2
  .byte $83, $06                ;   Play Sample 3
  .byte $86, $06                ;   Play Sample 6
  .byte $86, $06                ;   Play Sample 6
  .byte $40                     ;  NEXT instr (level 2)
  .byte $20, <rout13, >rout13   ;  JSR instr
  .byte $82, $06                ; Play Sample 2
  .byte $83, $06                ;  Play Sample 3
  .byte $82, $06                ; Play Sample 2
  .byte $82, $06                ; Play Sample 2
  .byte $40                     ; NEXT instr (level 1)
  .byte $4C, <STune2_, >STune2_ ; JMP istr

;================================
; Play Sample 6 routine
;================================
PSample6:
      ldy  #$05
      lda  #$94
      clc
      adc  #$40                 ; $94+$40=$D4
      sta  Vol6+2               ; unmask D418 address

nextDelayP6:
      ldx  #$14

againP6:
      lda  delayTabP6,y         ; read duration

delayP6:                        ; waste some times
      sec
      sbc  #$01
      bne  delayP6

      lda  $DE                  ; increase up volume sequence
      clc
      adc  #$98
      sta  $DE
      and  #$0F

Vol6:
      sta  $9418                ; D418: play sample
```

```
      dex
      bne  againP6

      dey
      bpl  nextDelayP6

      lda  #$94
      sta  Vol6+2                ; mask again D418 address
      rts

;================================
; Play Sample 3 routine
;================================
PSample3:
      ldy  #$05
      lda  #$94
      clc
      adc  #$40
      sta  Vol3+2                ; unmask D418 address

nextDelayP3:
      ldx  #$19                  ; set repeating value

againP3:
      lda  delayTabP3,y

delayP3:                         ; waste some times
      sec
      sbc  #$01
      bne  delayP3

      lda  $DE                   ; increase up volume sequence
      clc
      adc  #$65
      sta  $DE
      and  #$0F

Vol3:
      sta  $9418                 ; D418: play sample
      dex
      bne  againP3

      dey                        ; select next delay in table
      bpl  nextDelayP3

      lda  #$94
      sta  Vol3+2                ; mask again D418 address
      rts

;low address to play sample routine
lowPSample:
  .byte <PSample1, <PSample2, <PSample3
  .byte <PSample4, <PSample5, <PSample6
  .byte <PNothing

;================================
; Play Sample 5 routine
;================================
PSample5:
      ldy  #$05
      lda  #$94
      clc
      adc  #$40
      sta  Vol5+2                ; unmask D418 address

nextDelayP5:
      ldx  #$19                  ; set repeating value

againP5:
      lda  delayTabP5,y

delayP5:                         ; waste some times
      sec
      sbc  #$01
      bne  delayP5

      lda  $DE                   ; increase up volume sequence
      clc
      adc  #$DD
      sta  $DE
      and  #$0F
Vol5:
      sta  $9418                 ; D418: play sample
      dex
      bne  againP5

      dey
      bpl  nextDelayP5

      lda  #$94
```

```
      sta  Vol5+2                ; mask again D418 address
      rts

; high pointers of operations
highOper:
  .byte >Inst_20_sample
  .byte >inst_40_sample
  .byte >inst_60_sample
  .byte >inst_49_sample
  .byte >inst_4C_sample


;================================
; Play Sample 4 routine
;================================
PSample4:
      ldy  #$05
      lda  #$94
      clc
      adc  #$40
      sta  Vol4+2                ; unmask D418 address

nextDelayP4:
      ldx  #$19                  ; set repeating value

againP4:
      lda  delayTabP4,y

delayP4:                         ; waste some times
      sec
      sbc  #$01
      bne  delayP4

      lda  $DE                   ; increase up volume sequence
      clc
      adc  #$DD
      sta  $DE
      and  #$0F

Vol4:
      sta  $9418                 ; D418: play sample
      dex
      bne  againP4

      dey
      bpl  nextDelayP4

      lda  #$94
      sta  Vol4+2                ; mask again D418 address
      rts

;================================
; Init sample routine
;================================
initSample:
      ldx  #$00
      stx  $DF                   ; reset sample generation index routine
      dex
      stx  $DC                   ; sample duration
      ldx  #<noSample
      ldy  #>noSample
      stx  $DA                   ; store low address of sample pattern
      sty  $DB                   ; store high address of sample pattern
      rts

;================================
; Play Sample 2 routine
;================================
PSample2:
      ldy  #$0D
      lda  #$94
      clc
      adc  #$40
      sta  Vol2+2                ; unmask D418 address

nextDelayP2:
      ldx  #$0C                  ; set repeating value

againP2:
      lda  delayTabP2,y

delayP2:                         ; waste some times
      sec
      sbc  #$01
      bne  delayP2

      lda  $DE                   ; increase up volume sequence
      clc
      adc  #$0E
      sta  $DE
```

```
      and  #$0F
Vol2:
      sta  $9418                ; D418: play sample
      dex
      bne  againP2

      dey                       ; select next delay in table
      bpl  nextDelayP2
      lda  #$94
      sta  Vol2+2               ; mask again D418 address
      rts

;================================
; Play Sample 1 routine
;================================
PSample1:
      ldy  #$05
      lda  #$94
      clc
      adc  #$40
      sta  Vol1+2               ; unmask D418 address

nextDelayP1:
      ldx  #$0C                 ; set repeating value

againP1:
      lda  delayTabP1,y

delayP1:                        ; waste some times
      sec
      sbc  #$01
      bne  delayP1

      lda  $DE                  ; increase up volume sequence
      clc
      adc  #$01
      sta  $DE
      and  #$0F
Vol1:
      sta  $DD18                ; play sample
      dex
      bne  againP1

      dey                       ; select next delay in table
      bpl  nextDelayP1

      lda  #$DD
      sta  Vol1+2               ; mask again D418 address
PNothing:
      rts

; table of operation
operTable:
  .byte $20, $40, $60, $49, $4C

delayTabP3:
  .byte $20, $10, $08, $04, $02, $01

; low pointer of operations
lowOper:
  .byte <Inst_20_sample
  .byte <inst_40_sample
  .byte <inst_60_sample
  .byte <inst_49_sample
  .byte <inst_4C_sample

delayTabP5:
  .byte $23, $14, $0C, $09, $06, $03

delayTabP4:
  .byte $05, $1E, $19, $14, $0F, $0A

delayTabP6:
  .byte $28, $0A, $14, $0A, $0F, $0A

delayTabP2:
  .byte $20, $10, $08, $04, $02, $01, $28
  .byte $0A, $3C, $0A, $14, $0A, $20, $40

delayTabP1:                      ; table of delay for Play sample 1
  .byte $40, $1E, $3E, $19, $3C, $14


;================================
; Generate sample
; If one routine of sample is
; available, it will be called
;================================
SampleGeneration:
      lda  $DF                  ; load sample generation routine index
```

```
      beq  exitSample            ; if 0 exit
      lda  #$00
      sta  $DF
callPSample:
      jsr  PSample3              ; execute play sample routine
      lda  #$0F                  ; turn the volume to max
      ora  TEMP                  ; turn filter to the store value
      sta  $D418                 ; Select volume and filter mode
      rts

;================================
; Play sample engine routine
;================================
playSample:
      dec  $DC                    ; dec sample duration
      bne  exitSample

nextPattern:
      ldy  #$00
      lda  ($DA),y                ; read sample pattern
      bmi  isPlay                 ; jump if >=80
      ldx  #$FF

nextOper:
      inx
      cmp  operTable,x            ; compare value to table for getting index of instr.
      bne  nextOper

      lda  lowOper,x              ; read low address of routine
      sta  decoded+1
      lda  highOper,x             ; read high address of routine
      sta  decoded+2
decoded:
      jsr  inst_60_sample         ; execute decoded instruction
      jmp  nextPattern

isPlay:
      sta  $DF                    ; store readed sample generator routine
      tax
      iny
      lda  ($DA),y                ; read sample duration
      sta  $DC                    ; store sample duration
      lda  $DA
      clc
      adc  #$02                   ; adjust pattern pointer low
      sta  $DA
      lda  $DB
      adc  #$00                   ; adjust pattern pointer high
      sta  $DB
      lda  lowPSample-$81,x       ; low address of play sample routines
      sta  callPSample+1
      lda  highPSample-$81,x      ; high address of play sample routine
      sta  callPSample+2

exitSample:
      rts
```

Conclusion

Are this the end about the Martin engine?

Maybe not, but now I would like to talk about the rip itself. It is a very clean rip, where the ripper had manage a complete code initialization that take order especially into sound effects generation. However in the rip there other part of Martin Engine (a voice 1 and 2 part), maybe present in the game but that I haven' t investigated why it is present (probably they are part of the relocation of the code into upper memory).

Din 4 end